



Tutoriels STM32F0

Direct Access Memory – DMA

USART – Mode Rx

Arouna DARGA – enseignant chercheur Sorbonne Université
arouna.darga@sorbonne-universite.fr

Table des matières

1.	Introduction.....	3
2.	Qu'est-ce que l'accès direct à la mémoire (DMA) ?	3
3.	Mise place du projet pour le tutoriel	5
4.	Utilisation de l'USART RX avec DMA	6
4.1	Configuration de USART2 avec DMA	6
4.2	Comprendre le comportement du DMA	10
4.3	Gestion du buffer de réception	15
4.3.1	Utilisation de la scrutation et des pointeurs	15
4.3.2	Utilisation des interruptions DMA.....	16
4.4	Résumé	22

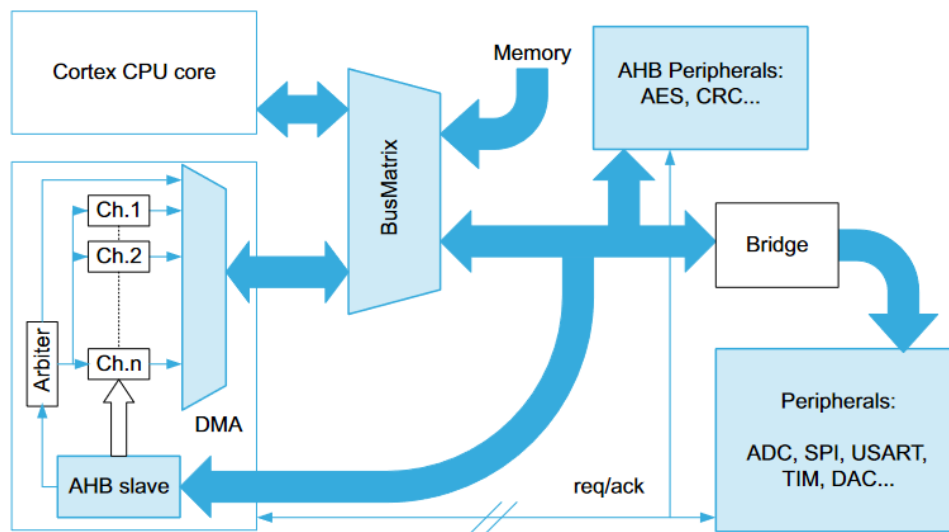
1. Introduction

Dans ce tutoriel, nous allons aborder le périphérique **DMA**, **D**irect **M**emory **A**ccess (accès direct à la mémoire) des microcontrôleurs STM32. Nous commencerons par une introduction sur ce qu'est un DMA, quand et pourquoi l'utiliser. Ensuite, nous discuterons du hardware du DMA STM32, de ses caractéristiques et de la façon de le configurer à travers deux exemples d'applications que nous allons construire tout au long de ce tutoriel.

2. Qu'est-ce que l'accès direct à la mémoire (DMA) ?

Une unité d'accès direct à la mémoire (DMA) est un élément logique numérique de l'architecture informatique qui peut être utilisé conjointement avec le microprocesseur principal sur la même puce afin de décharger le microprocesseur des opérations de transfert de mémoire. Cela réduit considérablement la charge de l'unité centrale. Le contrôleur DMA peut effectuer des transferts de données de mémoire à mémoire ainsi que des transferts de données de périphérique à mémoire ou vice versa. L'existence du DMA avec un CPU peut accélérer son débit de plusieurs ordres de grandeur.

Sans le DMA, le CPU (processeur principal) doit effectuer tout le travail de récupération des instructions (code) de la mémoire flash, exécuter les instructions décodées et déplacer les données vers et depuis les périphériques et la mémoire. Imaginez un périphérique comme l'USART (liaison série) qui reçoit un flux de données que le CPU doit immédiatement transférer vers un buffer local en mémoire afin de ne perdre aucun paquet de données. Cela se traduit par un nombre considérable d'interruptions par seconde déclenchées. Avec un flux de données de 10kB/s, un CPU sans DMA peut être très occupé et manquer les contraintes de temps pour l'application. Cette tâche de transfert de données doit être confiée à une autre unité, et c'est là que l'unité DMA intervient pour décharger le CPU de ces transactions de données très lourdes.



Comme vous pouvez le voir dans le diagramme ci-dessus, l'existence de l'unité DMA peut maintenant diriger le flux de données provenant des périphériques directement vers la mémoire pendant que le CPU effectue d'autres tâches et calculs. Cette coopération parallèle entre le CPU et le DMA est à l'origine de l'accélération.

Il existe deux types de transfert distincts :

- Mémoire à périphérique ou périphérique à mémoire :**
 Lorsqu'un périphérique est configuré en **mode DMA**, chaque donnée transférée est gérée de manière autonome au niveau matériel et au niveau des données entre le **DMA** et le périphérique concerné via un protocole dédié de demande de **DMA** et d'acquiescement. Le mappage des différents signaux de requête **DMA**, d'un périphérique donné à un **canal DMA donné**, est listé soit dans la section **DMA du manuel de référence**, soit dans la section d'implémentation **DMAMUX** du manuel de référence des produits.
- Mémoire à mémoire :**
 Le transfert ne nécessite aucun signal de commande supplémentaire, il est activé par le logiciel. Un canal est alloué par le logiciel, par exemple pour initialiser un gros bloc de données en RAM à partir de la mémoire Flash. Il est alors susceptible d'entrer en concurrence pour l'accès à la mémoire Flash avec la récupération des instructions du CPU. Dans tous les cas, l'arbitrage DMA entre les canaux est reconsidéré entre chaque donnée transférée.

3. Mise place du projet pour le tutorial

Pour ce tutorial, nous allons apporter des modifications majeures au code développé dans les tutoriels précédents. Il faut donc un projet dédié à ce tutorial. Cloner le projet « **Tutoriels Interruptions** » et nommer le projet cloné par « **Tutoriels_DMA** ». Pour la procédure de clonage suivre celle décrite dans le tutorial « **Tutoriels STM32F0-Environnement de développement v05022021.pdf** » (à partir de la page 30), disponible sur moodle.

4. Utilisation de l'USART RX avec DMA

4.1 Configuration de USART2 avec DMA

Dans cet exemple de tutoriel, nous verrons comment configurer le contrôleur **DMA** et le périphérique **USART2** pour que les octets entrants de l'USART soient automatiquement stockés dans une mémoire tampon sans avoir à interrompre le processeur.

Pour comprendre l'organisation du DMA, vous devez ouvrir le **manuel de référence du STM32F0** et visiter la section **DMA** (pages 197 à 228). Le tableau 3.1 fournit un résumé utile des requêtes **DMA** de périphériques disponibles, et sur quels canaux DMA ces requêtes sont effectuées. On peut voir ici que **USART2_RX** appelle le contrôleur **DMA** sur son **canal 5** ou **6**.

Utilisons le **canal 5** car aucun remappage **DMA** n'est nécessaire pour ce canal.

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC	ADC ⁽¹⁾	ADC ⁽²⁾	Reserved	Reserved	Reserved	Reserved	Reserved
SPI	Reserved	SPI1_RX	SPI1_TX	SPI2_RX ⁽¹⁾	SPI2_TX ⁽¹⁾	SPI2_RX ⁽²⁾	SPI2_TX ⁽²⁾
USART	Reserved	USART1_TX ⁽¹⁾ USART3_TX ⁽²⁾	USART1_RX ⁽¹⁾ USART3_RX ⁽²⁾	USART1_TX ⁽²⁾ USART2_TX ⁽¹⁾	USART1_RX ⁽²⁾ USART2_RX ⁽¹⁾	USART2_RX ⁽²⁾ USART3_RX ⁽¹⁾ USART4_RX	USART2_TX ⁽²⁾ USART3_TX ⁽¹⁾ USART4_TX
I2C	Reserved	I2C1_TX ⁽¹⁾	I2C1_RX ⁽¹⁾	I2C2_TX	I2C2_RX	I2C1_TX ⁽²⁾	I2C1_RX ⁽²⁾
TIM1	Reserved	TIM1_CH1 ⁽¹⁾	TIM1_CH2 ⁽¹⁾	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_CH3 ⁽¹⁾ TIM1_UP	TIM1_CH1 ⁽²⁾ TIM1_CH2 ⁽²⁾ TIM1_CH3 ⁽²⁾	Reserved
TIM2	TIM2_CH3	TIM2_UP	TIM2_CH2 ⁽¹⁾	TIM2_CH4 ⁽¹⁾	TIM2_CH1	Reserved	TIM2_CH2 ⁽²⁾ TIM2_CH4 ⁽²⁾
TIM3	Reserved	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1 ⁽¹⁾ TIM3_TRIG ⁽¹⁾	Reserved	TIM3_CH1 ⁽²⁾ TIM3_TRIG ⁽²⁾	Reserved
TIM6 / DAC	Reserved	Reserved	TIM6_UP DAC_Channel1	Reserved	Reserved	Reserved	Reserved
TIM7 / DAC	Reserved	Reserved	Reserved	TIM7_UP DAC_Channel2	Reserved	Reserved	Reserved
TIM15	Reserved	Reserved	Reserved	Reserved	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM	Reserved	Reserved
TIM16	Reserved	Reserved	TIM16_CH1 ⁽¹⁾ TIM16_UP ⁽¹⁾	TIM16_CH1 ⁽²⁾ TIM16_UP ⁽²⁾	Reserved	TIM16_CH1 ⁽³⁾ TIM16_UP ⁽³⁾	Reserved
TIM17	TIM17_CH1 ⁽¹⁾ TIM17_UP ⁽¹⁾	TIM17_CH1 ⁽²⁾ TIM17_UP ⁽²⁾	Reserved	Reserved	Reserved	Reserved	TIM17_CH1 ⁽³⁾ TIM17_UP ⁽³⁾

La configuration d'un canal **DMA** (c'est-à-dire le routage) n'est pas aussi compliquée qu'il n'y paraît à première vue. Le **DMA** est chargé de prendre des données à une **adresse source** (mémoire ou périphérique) et de les transmettre à une **adresse de destination** (périphérique ou mémoire). En outre, le **DMA** doit savoir si l'adresse source ou de destination est un buffer (c'est-à-dire un tableau de données) et quelle est la taille (8/16/32) des données à transmettre. C'est presque tout.

Dans notre cas particulier :

- A une des extrémités du transfert (Périphérique) se trouve le registre **RDR** de **USART2**. Il contient des **données de 8 bits** et son adresse est toujours la même, donc **nous ne voulons pas qu'il soit incrémenté après chaque transfert**.
- A l'autre extrémité du transfert (Mémoire) se trouve un tableau de données de 8 bits défini comme variable globale **rx_dma_buffer[8]**. Le **DMA doit incrémenter l'adresse de destination après chaque transfert** afin de remplir le buffer de manière séquentielle.
- Le sens de transfert est **Périphérique → Mémoire**
- La quantité de données à transférer est de **8**
- Nous voulons un remplissage circulaire de la mémoire (on reprend au début quand la mémoire est pleine)
- En plus :
 - Nous souhaitons faire fonctionner le **DMA** sans aucune interruption. Vous pouvez désactiver l'interruption **RXNE de USART2** (commenter la ligne concernée).
 - Vous devez dire à **USART2** d'appeler (requête) un transfert **DMA** à chaque fois qu'un nouvel octet est reçu.

Modifiez la fonction **BSP_Console_Init()** afin d'ajouter la fonctionnalité DMA pour les octets entrants (RX).
Le code est ci-dessous :

```
/*
 * BSP_Console_Init()
 * USART2 @ 115200 Full Duplex
 * 1 start - 8-bit - 1 stop
 * TX -> PA2 (AF1)
 * RX -> PA3 (AF1)
 */
extern uint8_t rx_dma_buffer[8];

void BSP_Console_Init()
{
    //activer horloge du peripherique GPIOA
    // mettre le bit b17 du registre RCC_AHBENR a '1'
    // voir page 128 du manuel technique (User Manuel) du Microcontrôleur STM32F072RB
    // le bit b17 de RCC_AHBENR est également défini = RCC_AHBENR_GPIOAEN dans le fichier stm32f0xx.h
    RCC->AHBENR |= (1<<17); // ou RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    // Choisir le mode Alternate Function (AF) pour les broches PA2 et PA3
    // pour PA2 : ecrire "10" sur les bits b5b4 du registre GPIOA_MODER
    GPIOA->MODER |= (1<<5);
    GPIOA->MODER &= ~(1<<4);
    // pour PA3 : ecrire "10" sur les bits b7b6 du registre GPIOA_MODER
    GPIOA->MODER |= (1<<7);
    GPIOA->MODER &= ~(1<<6);
    // Choisir la fonction AF1 (USART2) pour les broches PA2 et PA3
    // pour PA2 : ecrire "0001" sur les bits b11b10b9b8 du registre GPIOA_AFR1 = GPIO->AFR[0]
    GPIOA->AFR[0] &= ~(1<<11);
    GPIOA->AFR[0] &= ~(1<<10);
    GPIOA->AFR[0] &= ~(1<<9);
    GPIOA->AFR[0] |= (1<<8);
    // Choisir la fonction AF1 (USART2) pour les broches PA2 et PA3
    // pour PA3 : ecrire "0001" sur les bits b15b14b13b12 du registre GPIOA_AFR2 = GPIO->AFR[1]
    GPIOA->AFR[1] &= ~(1<<15);
    GPIOA->AFR[1] &= ~(1<<14);
    GPIOA->AFR[1] &= ~(1<<13);
    GPIOA->AFR[1] |= (1<<12);
    //activer horloge du peripherique USART2
    // mettre '1' le bit b17 du registre (RCC_APB1ENR)
    // voir page 131 du manuel technique (User Manuel) du Microcontrôleur STM32F072RB
    RCC -> APB1ENR |= (1<<17);
```

```

//Reset de la configuration de USART2 : Mise a zero des registres de control de USART2
//USART2_CR1, USART2_CR2 , USART2_CR3
// On utilise les valeurs par defaut
// 8-bits de donnees
// 1 bit START
// 1 bit STOP
// desactivation de CTS/RTS
USART2->CR1 = 0x00000000;
USART2->CR2 = 0x00000000;
USART2->CR3 = 0x00000000;
// Choisir la source PCLK (APB1) comme source horloge de USART2 : Valeur par defaut
// PCLK -> 48 MHz
// mettre "00" sur les bits b17b16 du registre (RCC_CFGR3)
// voir page 140 du reference manual
RCC->CFGR3 &= ~(1<<17);
RCC->CFGR3 &= ~(1<<16);
// Configuration du Baud Rate = 115200
// sans oversampling 8 bits (OVER8=0) et Fck=48MHz, USARTDIV = 48E6/115200 = 416.6666
// BRR = 417 -> Baud Rate = 115107.9137 -> 0.08% erreur
// avec oversampling 8 bits (OVER8=1) and Fck=48MHz, USARTDIV = 2*48E6/115200 = 833.3333
// BRR = 833 -> Baud Rate = 115246.0984 -> 0.04% error (Meilleur choix)
// choix oversampling 8 bits (OVER8=1)
// mettre a '1' le bit b15 de USART2_CR1
USART2->CR1 |= (1<<15);
// ecrire la valeur du Baud Rate dans le registre USART2_BRR
USART2->BRR = 833;
// Activer la transmission : ecrire '1' sur le bit b3 de USART2_CR1
USART2->CR1 |= (1<<3);

// Activer la reception : ecrire '1' sur le bit b2 de USART2_CR1
USART2->CR1 |= (1<<2);
// Action la demande interruption de evenement RXNE : caractere recu
//USART2->CR1 |= USART_CR1_RXNEIE;
// Configuration de RX sur le canal 5 du DMA
// Activer horloge du DMA
RCC->AHBENR |= RCC_AHBENR_DMA1EN;
// Réinitialisation de la configuration du canal 5 du DMA1
DMA1_Channel5->CCR = 0x00000000;
// Définir la direction Peripherique -> Memoire
DMA1_Channel5->CCR &= ~DMA_CCR_DIR;
// Définir USART2 RDR comme etant le périphérique
DMA1_Channel5->CPAR = (uint32_t)&USART2->RDR;
// Indiquer que la taille des donnees du peripherique est de 8 bits (octet).
DMA1_Channel5->CCR |= (0x00 << DMA_CCR_PSIZE_Pos);
// Desactiver auto-incrementation de Adresse du peripherique
DMA1_Channel5->CCR &= ~DMA_CCR_PINC;
// Définir le tableau rx_dma_buffer comme etant la memoire
DMA1_Channel5->CMAR = (uint32_t)rx_dma_buffer;
// Indiquer que la taille des donnees de la memoire est de 8 bits (octet).
DMA1_Channel5->CCR |= (0x00 << DMA_CCR_MSIZE_Pos);
// activer auto-incrementation de Adresse de la memoire
DMA1_Channel5->CCR |= DMA_CCR_MINC;
// Définir la taille de la memoire tampon
DMA1_Channel5->CNDTR = 8;
// Activer le mode circulaire du DMA
DMA1_Channel5->CCR |= DMA_CCR_CIRC;
// Activer le canal 5 du DMA
DMA1_Channel5->CCR |= DMA_CCR_EN;
// Activer la requete DMA pour la reception USART RX
USART2->CR3 |= USART_CR3_DMAR;
// activer le peripherique USART2 en dernier
// mettre a '1' le bit b0 de USART2_CR1
USART2->CR1 |= (1<<0);
}

```


Notez que puisque nous avons désactivé l'interruption RXNE, dans le fichier bsp.c, dans la fonction **BSP_NVIC_Init()**, il faut commenter les lignes : (i) autorisation interruption USART ; (ii) configuration de la priorité de la demande d'interruption de l'USART

```
/*
 * BSP_NVIC_Init()
 * Configuration du controleur NVIC pour autoriser et accepter les sources interruptions activer
 */

void BSP_NVIC_Init()
{
    // Mettre en priorite maximum les lignes interruptions externes EXTI 4 a 15
    NVIC_SetPriority(EXTI4_15_IRQn, 0);
    // Autoriser les lignes interruptions externes EXTI 4 a 15
    // le bouton Bleu est sur PC13 (ligne 13)
    NVIC_EnableIRQ(EXTI4_15_IRQn);
    // Mettre la priorite 1 pour les interruptions du TIM6
    NVIC_SetPriority(TIM6_DAC_IRQn, 1);
    // Autoriser les demandes interruptions du TIM6
    NVIC_EnableIRQ(TIM6_DAC_IRQn);

    // Mettre la priorite 1 pour les interruptions USART2
    // NVIC_SetPriority(USART2_IRQn, 1);

    // Autoriser les demandes interruptions de caractere USART2
    // NVIC_EnableIRQ(USART2_IRQn);
}
```

Vous pouvez garder l'ISR associée **USART2_IRQHandler()** dans le fichier **stm32f0xx_it.c** mais elle ne sera jamais appelée.

4.2 Comprendre le comportement du DMA

Maintenant, modifiez la fonction main() de façon à ce que le CPU fasse très peu de choses:

```
/*
 * main.c
 *
 * Created on: Feb 3, 2021
 * Author: darga
 */
#include <math.h>
#include "stm32f0xx.h"
#include "bsp.h"
#include "delay.h"
#include "main.h"
static void SystemClock_Config(void);

// variable globale
uint8_t button_irq = 0;
uint8_t timebase_irq = 0;
uint8_t console_rx_byte [10];
uint8_t console_rx_irq;

uint8_t rx_dma_buffer[8];

int main(void)
{
    // Configuration horloge du systeme : 48MHz avec 8MHz HSE
    SystemClock_Config();

    // Initialisation USART pour console liaison serie
    BSP_Console_Init();
    // test console : message accueil
    mon_printf("La Console est Ready!\r\n");
    // boucle principale des applications
    while(1)
    {
        // Le CPU ne fait rien

    }
}
```

Compilez le projet, flashez le MCU STM32F072RB, lancez une session de débogage et ouvrez une console de terminal série.


```

23
24 int main(void)
25 {
26
27     // Configuration horloge du système : 48MHz avec 8MHz HSE
28     SystemClock_Config();
29
30     // Initialisation USART pour console liaison série
31     BSP_Console_Init();
32     // test console : message accueil
33     mon_printf("La Console est Ready!\r\n");
34
35     // boucle principale des applications
36     while(1)
37     {
38         // Le CPU ne fait rien
39     }
40 }
41
42




























```


Dans la fenêtre **Expressions** , ajoutez la variable 'rx_dma_buffer', c'est le buffer pour les octets entrants.

Pour le moment, tout est à zéro :

Expression	Type	Value
(x)= SystemCoreClock	uint32_t	8000000
▼  rx_dma_buffer	uint8_t [8]	0x20000030 <rx...
(x)= rx_dma_buffer[0]	uint8_t	0 '\0'
(x)= rx_dma_buffer[1]	uint8_t	0 '\0'
(x)= rx_dma_buffer[2]	uint8_t	0 '\0'
(x)= rx_dma_buffer[3]	uint8_t	0 '\0'
(x)= rx_dma_buffer[4]	uint8_t	0 '\0'
(x)= rx_dma_buffer[5]	uint8_t	0 '\0'
(x)= rx_dma_buffer[6]	uint8_t	0 '\0'
(x)= rx_dma_buffer[7]	uint8_t	0 '\0'
+ Add new expression		

Dans la fenêtre **SFR**, déployez **STM32F0x2→DMA1→CNDTR5** et observez la valeur **NDT**. Il s'agit du compteur de transfert restant du DMA.

Register	Address	Value
▼  DMA1		
>  ISR	0x40020000	0x0
>  IFCR	0x40020004	0x0
>  CCR1	0x40020008	0x0
>  CNDTR1	0x4002000c	0x0
>  CPAR1	0x40020010	0x0
>  CMAR1	0x40020014	0x0
>  CCR2	0x4002001c	0x0
>  CNDTR2	0x40020020	0x0
>  CPAR2	0x40020024	0x0
>  CMAR2	0x40020028	0x0
>  CCR3	0x40020030	0x0
>  CNDTR3	0x40020034	0x0
>  CPAR3	0x40020038	0x0
>  CMAR3	0x4002003c	0x0
>  CCR4	0x40020044	0x0
>  CNDTR4	0x40020048	0x0
>  CPAR4	0x4002004c	0x0
>  CMAR4	0x40020050	0x0
>  CCR5	0x40020058	0x0
▼  CNDTR5	0x4002005c	0x0
 NDT	[0:16]	0x0
>  CPAR5	0x40020060	0x0
>  CMAR5	0x40020064	0x0
>  CCR6	0x4002006c	0x0
>  CNDTR6	0x40020070	0x0
>  CPAR6	0x40020074	0x0


Cliquez ensuite deux fois sur  afin d'exécuter la fonction **BSP_Console_Init()** :

```

22 uint8_t rx_dma_buffer[8];
23
24 int main(void)
25 {
26
27     // Configuration horloge du systeme : 48MHz avec 8MHz HSE
28     SystemClock_Config();
29
30     // Initialisation USART pour console liaison serie
31     BSP_Console_Init();
32     // test console : message accueil
33     mon_printf("La Console est Ready!\r\n");
34
35     // boucle principale des applications
36     while(1)
37     {
38         // Le CPU ne fait rien
39     }
40 }
41
42


```


Maintenant USART2 et DMA ont été initialisés. SFR view indique le compteur DMA actuel :

> 1010 0101 CCR5	0x40020058	0xa7
▼ 1010 0101 CNDTR5	0x4002005c	0x8 
1010 0101 NDT	[0:16]	0x8
> 1010 0101 CPAR5	0x40020060	0x40004424
> 1010 0101 CMAR5	0x40020064	0x20000030


Maintenez l'exécution suspendue (c'est-à-dire ne touchez à rien dans le débogueur) et mettez le terminal série (Putty) au premier plan.

Ensuite, appuyez sur les touches du clavier 'a', 'z', 'e', 'r', 't', 'y'. Vous ne verrez rien se produire. C'est normal.

Revenez dans le débogueur et passez sur la ligne suivante . Le message de bienvenue est affiché dans la console, mais plus intéressant encore : les touches que vous avez frappées auparavant, **alors que le CPU ne tournait pas, SONT** dans le **rx_dma_buffer** !

Expression	Type	Value
▼  rx_dma_buffer	uint8_t [8]	0x20000028 <rx_dma_buffer>
(x)= rx_dma_buffer[0]	uint8_t	97 'a'
(x)= rx_dma_buffer[1]	uint8_t	122 'z'
(x)= rx_dma_buffer[2]	uint8_t	101 'e'
(x)= rx_dma_buffer[3]	uint8_t	114 'r'
(x)= rx_dma_buffer[4]	uint8_t	116 't'
(x)= rx_dma_buffer[5]	uint8_t	121 'y'
(x)= rx_dma_buffer[6]	uint8_t	0 '\0'
(x)= rx_dma_buffer[7]	uint8_t	0 '\0'
+ Add new expression		

Le compteur **DMA (CNDTR5)** est maintenant à 2. Le compteur **DMA** fonctionne comme un compteur à rebours, contenant le nombre de transferts restant à effectuer.

> 1010 0101 CCR5	0x40020058	0xa7
▼ 1010 0101 CNDTR5	0x4002005c	0x2 
1010 0101 NDT	[0:16]	0x2
> 1010 0101 CPAR5	0x40020060	0x40004424
> 1010 0101 CMAR5	0x40020064	0x20000030

Alors, que s'est-il passé ? Après que USART2 et DMA aient été initialisés, ces périphériques deviennent actifs indépendamment du CPU. Même avec un CPU suspendu (c'est-à-dire une exécution en pause dans le débogueur), le processus de USART2 recevant des données, puis appelant le DMA pour stocker des octets dans la mémoire est actif. Lorsque vous avez appuyé sur le clavier du terminal série, les octets envoyés ont été transférés en mémoire instantanément. Il a juste fallu passer une fois avec le débogueur pour rafraîchir la vue de l'expression, mais en fait elle était déjà là.

Appuyez sur deux autres touches 'u', 'i', dans la fenêtre du terminal (CPU toujours suspendu). Puis lancez/suspendez (▶ / ⏸) le programme pour rafraîchir les vues Expression et SFR :

Expression	Type	Value
▼ rx_dma_buffer	uint8_t [8]	0x20000028 <rx_dma_buffer>
(x)= rx_dma_buffer[0]	uint8_t	97 'a'
(x)= rx_dma_buffer[1]	uint8_t	122 'z'
(x)= rx_dma_buffer[2]	uint8_t	101 'e'
(x)= rx_dma_buffer[3]	uint8_t	114 'r'
(x)= rx_dma_buffer[4]	uint8_t	116 't'
(x)= rx_dma_buffer[5]	uint8_t	121 'y'
(x)= rx_dma_buffer[6]	uint8_t	117 'u'
(x)= rx_dma_buffer[7]	uint8_t	105 'i'
➕ Add new expression		

> 1010 0101 CCR5	0x40020058	0xa7
▼ 1010 0101 CNDTR5	0x4002005c	0x8
1010 0101 NDT	[0:16]	0x8
> 1010 0101 CPAR5	0x40020060	0x40004424
> 1010 0101 CMAR5	0x40020064	0x20000030

Vous pouvez voir que les deux derniers octets ont été ajoutés dans le tableau, et que le compteur **DMA** a été remis au compte initial de **8**. Appuyez sur deux autres touches 'o', 'p', dans la fenêtre du terminal. Puis exécutez/suspendez à nouveau (▶ / ⏸) le programme pour rafraîchir les vues Expression et SFR :

Expression	Type	Value
▼ rx_dma_buffer	uint8_t [8]	0x20000028 <rx_dma_buffer>
(x)= rx_dma_buffer[0]	uint8_t	111 'o'
(x)= rx_dma_buffer[1]	uint8_t	112 'p'
(x)= rx_dma_buffer[2]	uint8_t	101 'e'
(x)= rx_dma_buffer[3]	uint8_t	114 'r'
(x)= rx_dma_buffer[4]	uint8_t	116 't'
(x)= rx_dma_buffer[5]	uint8_t	121 'y'
(x)= rx_dma_buffer[6]	uint8_t	117 'u'
(x)= rx_dma_buffer[7]	uint8_t	105 'i'

Comme nous avons configuré le **DMA** pour qu'il fonctionne en **mode circulaire**, les nouveaux octets ont été stockés au début du tampon et le compteur du **DMA** est maintenant de **6**.

> 1010 0101 CCR5	0x40020058	0xa7
▼ 1010 0101 CNDTR5	0x4002005c	0x6
1010 0101 NDT	[0:16]	0x6
> 1010 0101 CPAR5	0x40020060	0x40004424
> 1010 0101 CMAR5	0x40020064	0x20000030

En résumé :

- Aucun octet entrant n'est perdu tant que le buffer **rx_dma_buffer[]** n'est pas complètement rempli. Lorsque le **rx_dma_buffer[]** est entièrement rempli (c'est-à-dire lorsque le compteur **DMA** atteint 0), le compteur DMA reprend sa valeur initiale (8) et les nouveaux octets entrants sont à nouveau stockés depuis le début du **rx_dma_buffer[]**. C'est le comportement du mode circulaire.
- Toutes ces opérations sont effectuées sans aucune aide de la **CPU**. **USART2**, **DMA** et la mémoire fonctionnent en totale indépendance, même lorsque le **CPU** est arrêté par le débogueur.

4.3 Gestion du buffer de réception

4.3.1 Utilisation de la scrutation et des pointeurs

Disons que nous voulons un programme qui "répercute" simplement les touches frappées dans la console du terminal série (le PC envoie au MCU et le MCU lui renvoie le même caractère). Nous utilisons le DMA pour gérer les requêtes USART RX et une mémoire `rx_dma_buffer[]` permettant à la tâche d'être exécutée seulement une fois par seconde (c'est-à-dire en laissant du temps pour d'autres tâches) sans perdre aucun octet entrant. Une implémentation possible serait celle-ci :

```
static void SystemClock_Config(void);

// variable globale
uint8_t button_irq = 0;
uint8_t timebase_irq = 0;
uint8_t console_rx_byte [10];
uint8_t console_rx_irq;

uint8_t rx_dma_buffer[8];

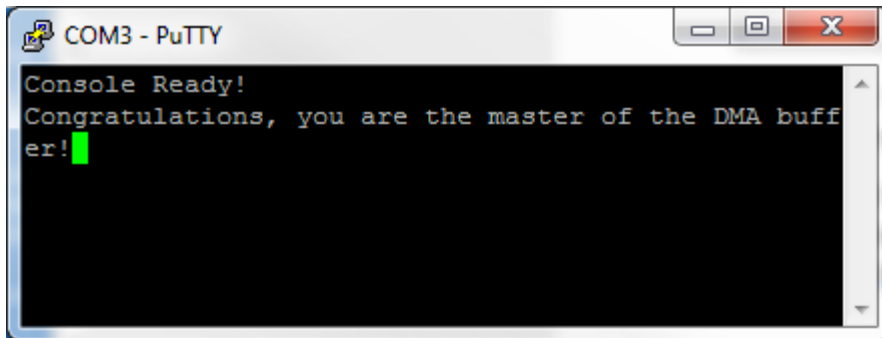
int main(void)
{
    uint8_t DMA_Counter;
    uint8_t index;

    // Configuration horloge du systeme : 48MHz avec 8MHz HSE
    SystemClock_Config();
    // Initialisation USART pour console liaison serie
    BSP_Console_Init();
    // test console : message accueil
    mon_printf("La Console est Ready!\r\n");
    // Initialiser le TIM6 pour generer une interruption toute les 1s
    BSP_TIMER_Timebase_Init();
    BSP_NVIC_Init();
    // Initialiser les variables
    DMA_Counter = DMA1_Channel5->CNDTR;
    index = DMA_Counter;
    // boucle principale des applications
    while(1)
    {
        // Faire toutes les 1s
        if (timebase_irq == 1)
        {
            //Obtenir la valeur actuelle du compteur DMA
            DMA_Counter = DMA1_Channel5->CNDTR;
            //Pour tous les nouveaux octets recus
            while (index != DMA_Counter)
            {
                // Envoyer un octet au PC pour etre afficher sur la console PuTTY
                while ( (USART2->ISR & USART_ISR_TC) != USART_ISR_TC);
                USART2->TDR = rx_dma_buffer[8-index];

                // Mise a jour de index du buffer circulaire
                index--;
                if (index == 0) index = 8;
            }
            // reset du drapeau interruption TIM6
            timebase_irq = 0;
        }
    }
}
```

Prenez le temps de comprendre le code ci-dessus. Fondamentalement, la variable `index` est utilisée pour adresser les octets individuels dans le `rx_dma_buffer[]` et se trouve après le compteur **DMA**.

Vous pouvez tester le programme dans la console. Tapez n'importe quoi sur le clavier et vous devriez voir votre entrée être mise à jour toutes les secondes sans aucune perte... à moins que vous ne frappiez plus de 8 touches par seconde (ce qui n'est pas si difficile).



Dans cet exemple, la fonction `main()` interroge le `rx_dma_buffer[]` toutes les 1s. Que se passe-t-il si vous recevez plus de 8 octets par seconde ?

Eh bien, certains octets seraient perdus, écrasés par le DMA dans le `rx_dma_buffer[]`, avant que vous ne trouviez le temps de les traiter (c'est-à-dire de les afficher).

Bien sûr, vous pouvez traiter ce problème en :

- Augmentant la taille de la mémoire tampon
- En raccourcissant la période de requête

Notez qu'en faisant cela, vous vérifierez l'état du DMA même dans le cas où aucun octet n'a été transféré...

Une alternative est de s'appuyer sur une fonctionnalité supplémentaire du contrôleur DMA : Les interruptions DMA !

4.3.2 Utilisation des interruptions DMA

Le contrôleur **DMA** est capable d'envoyer des signaux d'interruption basés sur plusieurs **événements DMA**. On pourrait par exemple envoyer un **signal d'interruption** à chaque fois qu'un **octet est transféré de l'USART vers la mémoire**, mais ce serait stupide puisque le but de l'implication du DMA dans le processus RX est d'éviter les interruptions multiples d'autres tâches importantes.

En travaillant avec des buffers, le DMA est capable de déclencher deux événements très utiles :

- L'événement Half-Transfer (HT), lorsque le DMA atteint la première moitié du tampon.
- L'événement Transfer-Complete (TC), lorsque le DMA atteint la fin du tampon et recommence au début.

Modifiez la fonction **BSP_Console_Init()** avec la configuration suivante :

```
/*
 * BSP_Console_Init()
 * USART2 @ 115200 Full Duplex
 * 1 start - 8-bit - 1 stop
 * TX -> PA2 (AF1)
 * RX -> PA3 (AF1)
 */
extern uint8_t rx_dma_buffer[8];

void BSP_Console_Init()
{
    //activer horloge du peripherique GPIOA
    // mettre le bit b17 du registre RCC_AHBENR a '1'
    // voir page 128 du manuel technique (User Manuel) du Microcontrôleur STM32F072RB
    // le bit b17 de RCC_AHBENR est egalement defini = RCC_AHBENR_GPIOAEN dans le fichier stm32f0xx.h
    RCC->AHBENR |= (1<<17); // ou RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    // Choisir le mode Alternate Function (AF) pour les broches PA2 et PA3
    // pour PA2 : ecrire "10" sur les bits b5b4 du registre GPIOA_MODER
    GPIOA->MODER |= (1<<5);
    GPIOA->MODER &= ~(1<<4);
    // pour PA3 : ecrire "10" sur les bits b7b6 du registre GPIOA_MODER
    GPIOA->MODER |= (1<<7);
    GPIOA->MODER &= ~(1<<6);
    // Choisir la fonction AF1 (USART2) pour les broches PA2 et PA3
    // pour PA2 : ecrire "0001" sur les bits b11b10b9b8 du registre GPIOA_AFR1 = GPIO->AFR[0]
    GPIOA->AFR[0] &= ~(1<<11);
    GPIOA->AFR[0] &= ~(1<<10);
    GPIOA->AFR[0] &= ~(1<<9);
    GPIOA->AFR[0] |= (1<<8);
    // Choisir la fonction AF1 (USART2) pour les broches PA2 et PA3
    // pour PA3 : ecrire "0001" sur les bits b15b14b13b12 du registre GPIOA_AFR1 = GPIO->AFR[0]
    GPIOA->AFR[0] &= ~(1<<15);
    GPIOA->AFR[0] &= ~(1<<14);
    GPIOA->AFR[0] &= ~(1<<13);
    GPIOA->AFR[0] |= (1<<12);
    //activer horloge du peripherique USART2
    // mettre '1' le bit b17 du registre (RCC_APB1ENR)
    // voir page 131 du manuel technique (User Manuel) du Microcontrôleur STM32F072RB
    RCC -> APB1ENR |= (1<<17);
    //Reset de la configuration de USART2 : Mise a zero des registres de control de USART2
    //USART2_CR1, USART2_CR2 , USART2_CR3
    // On utilise les valeurs par default
    // 8-bits de donnees
    // 1 bit START
    // 1 bit STOP
    // desactivation de CTS/RTS
    USART2->CR1 = 0x00000000;
    USART2->CR2 = 0x00000000;
    USART2->CR3 = 0x00000000;
    // Choisir la source PCLK (APB1) comme source horloge de USART2 : Valeur par default
    // PCLK -> 48 MHz
    // mettre "00" sur les bits b17b16 du registre (RCC_CFGR3)
    // voir page 140 du reference manual
    RCC->CFGR3 &= ~(1<<17);
    RCC->CFGR3 &= ~(1<<16);

    // Configuration du Baud Rate = 115200
    // sans oversampling 8 bits (OVER8=0) et Fck=48MHz, USARTDIV = 48E6/115200 = 416.6666
    // BRR = 417 -> Baud Rate = 115107.9137 -> 0.08% erreur
    // avec oversampling 8 bits (OVER8=1) and Fck=48MHz, USARTDIV = 2*48E6/115200 = 833.3333
    // BRR = 833 -> Baud Rate = 115246.0984 -> 0.04% error (Meilleur choix)

    // choix oversampling 8 bits (OVER8=1)
    // mettre a '1' le bit b15 de USART2_CR1
    USART2->CR1 |= (1<<15);
}
```

```

// ecrire la valeur du Baud Rate dans le registre USART2_BRR
USART2->BRR = 833;
// Activer la transmission : ecrire '1' sur le bit b3 de USART2_CR1
USART2->CR1 |= (1<<3);
// Activer la reception : ecrire '1' sur le bit b2 de USART2_CR1
USART2->CR1 |= (1<<2);
// Action la demande interruption de evenement RXNE : caractere reçu
//USART2->CR1 |= USART_CR1_RXNEIE;
// Configuration de RX sur le canal 5 du DMA
// Activer horloge du DMA
RCC->AHBENR |= RCC_AHBENR_DMA1EN;
// Reinitialisation de la configuration du canal 5 du DMA1
DMA1_Channel5->CCR = 0x00000000;
// Définir la direction Peripherique -> Memoire
DMA1_Channel5->CCR &= ~DMA_CCR_DIR;
// Définir USART2 RDR comme etant le peripherique
DMA1_Channel5->CPAR = (uint32_t)&USART2->RDR;
// Indiquer que la taille des donnees du peripherique est de 8 bits (octet)
DMA1_Channel5->CCR |= (0x00 << DMA_CCR_PSIZE_Pos);
// Desactiver auto-incrementation de Adresse du peripherique
DMA1_Channel5->CCR &= ~DMA_CCR_PINC;
// Définir le tableau rx_dma_buffer comme etant la memoire
DMA1_Channel5->CMAR = (uint32_t)rx_dma_buffer;
// Indiquer que la taille des donnees de la memoire est de 8 bits (octet)
DMA1_Channel5->CCR |= (0x00 << DMA_CCR_MSIZE_Pos);
// activer auto-incrementation de Adresse de la memoire
DMA1_Channel5->CCR |= DMA_CCR_MINC;
// Définir la taille de la memoire tampon
DMA1_Channel5->CNDTR = 8;
// Activer le mode circulaire du DMA
DMA1_Channel5->CCR |= DMA_CCR_CIRC;
// Activation des interruptions DMA HT & TC
DMA1_Channel5->CCR |= DMA_CCR_HTIE | DMA_CCR_TCIE;
// Activer le canal 5 du DMA
DMA1_Channel5->CCR |= DMA_CCR_EN;
// Activer la requete DMA pour la reception USART RX
USART2->CR3 |= USART_CR3_DMAR;
// activer le peripherique USART2 en dernier
// mettre a '1' le bit bit b0 de USART2_CR1
USART2->CR1 |= (1<<0);
}

```

Puis activer l'interruption **DMA** pour passer par le **NVIC avec la priorité 1** :

```
void BSP_NVIC_Init()
{
    // Mettre en priorite maximum les lignes interruptions externes EXTI 4 a 15
    NVIC_SetPriority(EXTI4_15_IRQn, 0);

    // Autoriser les lignes interruptions externes EXTI 4 a 15
    // le bouton Bleu est sur PC13 (ligne 13)
    NVIC_EnableIRQ(EXTI4_15_IRQn);

    // Mettre la priorite 1 pour les interruptions du TIM6
    NVIC_SetPriority(TIM6_DAC_IRQn, 1);

    // Autoriser les demandes interruptions du TIM6
    NVIC_EnableIRQ(TIM6_DAC_IRQn);

    // Mettre la priorite 1 pour les interruptions USART2
    // NVIC_SetPriority(USART2_IRQn, 1);

    // Autoriser les demandes interruptions de caractere USART2
    // NVIC_EnableIRQ(USART2_IRQn);

    // Definir le niveau de priorite 1 pour les interruptions DMA1_Channel5
    NVIC_SetPriority(DMA1_Channel4_5_6_7_IRQn, 1);

    // Autoriser les demandes interruptions du DMA1_Channel5 interrupts
    NVIC_EnableIRQ(DMA1_Channel4_5_6_7_IRQn);
}
```

À ce stade, les signaux d'interruption TC et HT sont générés par le contrôleur DMA et transmis par le contrôleur NVIC. Il faut écrire la fonction de traitement :

```
/*
 * Cette fonction declencler surles interruptions du canal 5 du DMA1 (USART2 RX).
 */

extern uint8_t rx_dma_irq;

void DMA1_Channel4_5_6_7_IRQHandler()
{
    // Test pour le demi-transfert du canal 5
    if ((DMA1->ISR & DMA_ISR_HTIF5) == DMA_ISR_HTIF5)
    {
        // Effacer le bit ou drapeau d'interruption en attente
        DMA1->IFCR |= DMA_IFCR_CHTIF5;

        // Definir une variable globale
    }
```

```

    rx_dma_irq = 1;
}

// Test pour le transfert du canal 5 terminé
if ((DMA1->ISR & DMA_ISR_TCIF5) == DMA_ISR_TCIF5)
{
    // Effacer le bit d'interruption en attente
    DMA1->IFCR |= DMA_IFCR_CTCIF5;

    // Définir une variable globale
    rx_dma_irq = 2;
}
}

```

La variable globale **rx_dma_irq** est maintenant fixée à 1 ou 2 selon le cas d'interruption (HT ou TC) chaque fois que 4 octets ont été acheminés par **DMA** depuis le registre **RDR** de **USART2**. La fonction **main()** peut donc exploiter de cette situation :

```

static void SystemClock_Config(void);

// variable globale
uint8_t button_irq = 0;
uint8_t timebase_irq = 0;
uint8_t console_rx_byte [10];
uint8_t console_rx_irq;

uint8_t rx_dma_buffer[8];
uint8_t rx_dma_irq = 0;

int main(void)
{
    uint8_t index;

    // Configuration horloge du systeme : 48MHz avec 8MHz HSE
    SystemClock_Config();

    // Initialisation USART pour console liaison serie
    BSP_Console_Init();
    // test console : message accueil
    mon_printf("La Console est Ready!\r\n");
    // Initialiser le TIM6 pour generer une interruption toute les 1s
    BSP_TIMER_Timebase_Init();
    BSP_NVIC_Init();

    // boucle principale des applications
    while(1)
    {
        // Faire toutes les 1s
        if (timebase_irq == 1)
        {

```

```

switch(rx_dma_irq)
{
    case 1:// Une interruption de demi transfert (HT) s'est
produite
    {
        // Affichage des octets [0-3] pour affichage sur la
console de PC
        for (index=0; index<4; index++)
        {
            while ( (USART2->ISR & USART_ISR_TC) !=
USART_ISR_TC);

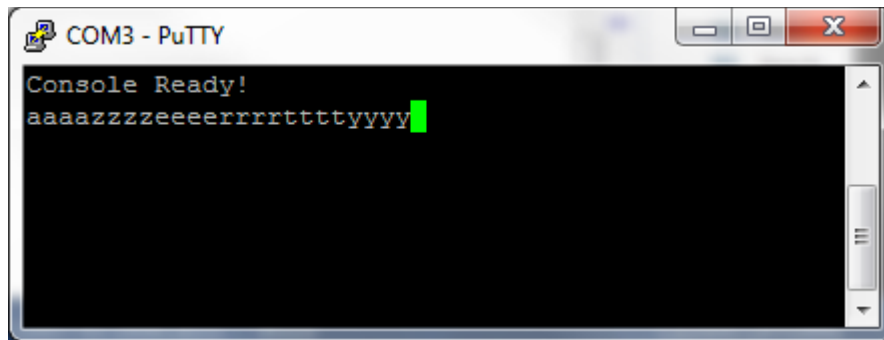
            USART2->TDR = rx_dma_buffer[index];
        }

        rx_dma_irq = 0;
        break;
    }
    case 2:// interruption du transfert complet (TC) s'est produite
    {
        // // Envoi des octets [4-7] pour affichage sur la
console de PC
        for (index=4; index<8; index++)
        {
            while ( (USART2->ISR & USART_ISR_TC) !=
USART_ISR_TC);

            USART2->TDR = rx_dma_buffer[index];
        }
        rx_dma_irq = 0;
        break;
    }
}
// reset du drapeau interruption TIM6
timebase_irq = 0;
}
}
}

```

Compilez le projet et exécutez le programme dans la console. Vous obtiendrez un rafraîchissement de l'affichage toutes les 1s, et seulement après avoir appuyé sur 4 touches :



Cette approche de la gestion des octets entrants est particulièrement intéressante lorsque vous devez analyser un flux continu d'octets entrants (par exemple, les messages NEMA d'un capteur GPS) tout en gagnant du temps pour d'autres tâches. Dans ce cas, vous pouvez gérer un tampon de données beaucoup plus grand. Lorsque vous êtes averti que la moitié d'un tampon est pleine, vous savez que vous devez le traiter avant que la seconde moitié ne soit remplie, sinon les données seront perdues.

4.4 Résumé

Dans ce tutoriel, vous avez appris à recevoir des octets du périphérique USART en utilisant le contrôleur DMA pour transférer automatiquement les octets entrants vers une variable en mémoire.

En outre, vous avez appris à utiliser les interruptions DMA dans un scénario de buffer d'entrée circulaire.