DMA & I2S

1 Introduction DMA

Le DMA permet aux périphériques d'**accéder directement à la mémoire** système **sans impliquer la CPU**.
Lors de l'utilisation du DMA, la CPU initie un transfert entre la mémoire et le périphérique. Le transfert est entièrement pris en charge par le contrôleur DMA et le processeur est libre d'effectuer d'autres tâches. Lorsque le transfert DMA est terminé, la CPU reçoit une interruption et peut alors configurer d'autres données à transmettre.

Les petits buffers DMA impliquent que le processeur doit faire plus de travail car il est interrompu plus souvent. Les tampons de grande taille impliquent que le processeur doit faire moins de travail car il recoit moins d'interruptions.

Supposons que nous échantillonnions à 25 kHz avec 8 bits par échantillon - cela donne un taux de transfert de données d'environ 25 Ko par seconde.

Si nous avions une taille de tampon DMA de 32 échantillons, nous interromprions le processeur toutes les 40*32 soit 1280 microsecondes. La taille du tampon doit être comprise entre **8** et **1024** échantillons.

2 Pourquoi plusieurs tampons DMA ?

Le problème avec un seul tampon est qu'il ne nous laisse pas le temps de charger les données. Le tampon DMA ne peut être utilisé que par la CPU ou le contrôleur DMA. Ils ne peuvent pas accéder au même tampon en même temps.

Si nous n'avons qu'un seul tampon, nous devons terminer notre traitement et rendre le tampon au contrôleur DMA avant de devoir transférer de nouvelles données.

Le résultat de ceci signifie qu'il faut généralement définir le **dma_buf_count** sur au moins 2. Avec deux tampons, on peut traiter un tampon avec le CPU tandis que l'autre tampon est vidé par le contrôleur DMA.

3 Les périphériques I2S

ESP32 contient deux périphériques I2S. Ces périphériques peuvent être configurés pour entrer et sortir des échantillons de données via le pilote I2S.

Le périphérique I2S prend en charge le DMA, ce qui signifie qu'il peut diffuser des échantillons de données sans que chaque échantillon soit lu ou écrit individuellement par le processeur. Toutefois pour éviter toute discontinuité du signal, le processeur doit être en mesure de recharger les tampons plus rapidement qu'ils ne sont déchargés par le DMA.

La sortie I2S peut également être acheminée directement vers les canaux de sortie du convertisseur numérique/analogique (GPIO 25 et GPIO 26) pour produire une sortie analogique.

configuration d'I2S pour utiliser le DAC interne comme sortie analogique

```
#include  "pilote/i2s.h"
#include  "freertos/queue.h"

statique  const  int  i2s_num  =  0 ;  // numéro de port i2s

statique  const  i2s_config_t  i2s_config  =  {
     .mode  =  I2S_MODE_MASTER  |  I2S_MODE_TX  |  I2S_MODE_DAC_BUILT_IN
,
     .sample_rate  =  44100 ,
     .bits_per_sample  =  16 ,  /*le DAC ne prendra que les 8 bits de
MSB */
     .channel_format  =  I2S_CHANNEL_FMT_ONLY_RIGHT, // only right
channel
     .communication_format  =  I2S_COMM_FORMAT_I2S_MSB ,
     .intr_alloc_flags  =  0 , // priorité d'interruption par défaut
     .dma_buf_count  =  8 ,   // Nb de buffers
     .dma_buf_len  =  64 ,    // Taille d'un buffer
     .use_apll  =  0
};
```

```
//installer et démarrer le pilote i2s
i2s_driver_install ( i2s_num ,  &i2s_config ,  0 ,  NULL );


//Vous pouvez appeler i2s_set_dac_mode pour définir le mode de sortie
DAC //intégré. Les fonctions DAC intégrées ne sont prises en charge que sur i2s_num = 0
// I2S_DAC_CHANNEL_RIGHT_EN : gpio 25
//  I2S_DAC_CHANNEL_LEFT_EN  : gpio 26
//  I2S_DAC_CHANNEL_BOTH_EN : les deux (pour du son stéréo)

i2s_set_dac_mode(I2S_DAC_CHANNEL_RIGHT_EN);

// Commence à utiliser le mode ADC intégré I2S. ???

i2s_adc_enable(I2S_NUM_0);
```

Écrire les données dans le tampon de transmission I2S DMA

Préparer les données pour l'envoi (voir fiche dds)

Appelez la fonction i2s_write() et transmettez-lui l'adresse du tampon de données et la longueur des données

```
i2s_write(I2S_NUM_0, (char*) bufferOut, NUM_SAMPLES * sizeof (uint16_t), &bytesWrite, portMAX_DELAY);
```

bufferOut Adresse source des données à écrire

NUM_SAMPLES * sizeof (uint16_t) Taille des données en octets

&bytesWrite Nombre d'octets écrits, si le timeout est atteint , le résultat sera inférieur à la taille passée.

portMAX_DELAY Délai d'attente du tampon TX portMAX_DELAY pour pas de délai d'attente