NSY107
Cours 8 DMA

Matthias Puech

Master 1 SEMS — Cnam

29 mai 2017

Principe du DMA

Configuration et utilisation avec HAL

Principe du DMA

Configuration et utilisation avec HAL

Rappel transmission d'un message en UART

Pour envoyer des trames

1. configurer CR1,2,3 et BRR
   (suivant les caractéristiques de la ligne)
2. attendre que TXE soit à 1
   (prêt à envoyer)
3. placer les N bits à envoyer dans TDR
4. TXE se met à 0
   (en cours d'envoi)
5. s'il reste des trames à envoyer, GOTO 2.
6. attendre que TC soit à 1
   (envoi terminé)

Rappel transmission d'un message en UART

Problème

Le processeur est interrompu tous les N bits envoyés :

1. il met la donnée à envoyer dans TDR
2. ... les N bits sont envoyés par l'UART
3. l'UART met son IRQ à 1
4. le processeur est interrompu ; il :
   - 4.1 sauvegarde son contexte ($\simeq$ 14 cycles)
   - 4.2 met la nouvelle donnéee à envoyer dans TDR (2-3 cycles)
   - 4.3 restaure son contexte ($\simeq$ 14 cycles)
5. ... les N bits suivants sont envoyés par l'UART ...

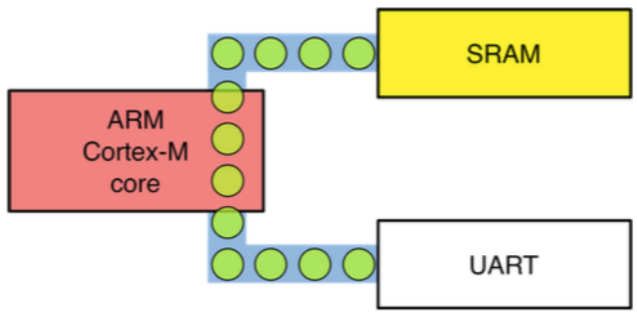
Rappel transmission d'un message en UART

Problème

Le processeur est interrompu tous les N bits envoyés :

1. il met la donnée à envoyer dans TDR
2. ... les N bits sont envoyés par l'UART
3. l'UART met son IRQ à 1
4. le processeur est interrompu ; il :
   - 4.1 sauvegarde son contexte ($\simeq$ 14 cycles)
   - 4.2 met la nouvelle donnéee à envoyer dans TDR (2-3 cycles)
   - 4.3 restaure son contexte ($\simeq$ 14 cycles)
5. ... les N bits suivants sont envoyés par l'UART ...

$\rightsquigarrow$ l'interruption répétée a un coût en cycles CPU



Le DMA

Périphérique de copie périphérique $\longleftrightarrow$ mémoire

- un registre pour stocker le mot à copier
- des adresses source et cible, taille en octet à copier
- une source de déclenchement de la copie
  (software, ou hardware : périphériques)

Le DMA

Périphérique de copie périphérique ⟷ mémoire

- un registre pour stocker le mot à copier
- des adresses source et cible, taille en octet à copier
- une source de déclenchement de la copie
  (software, ou hardware : périphériques)

Applications

- *buffering* : réception/émission de/vers périphérique par bloc
  (amortit le coût de l'interruption)
- *double buffering* (voir plus loin)

Les bus systèmes

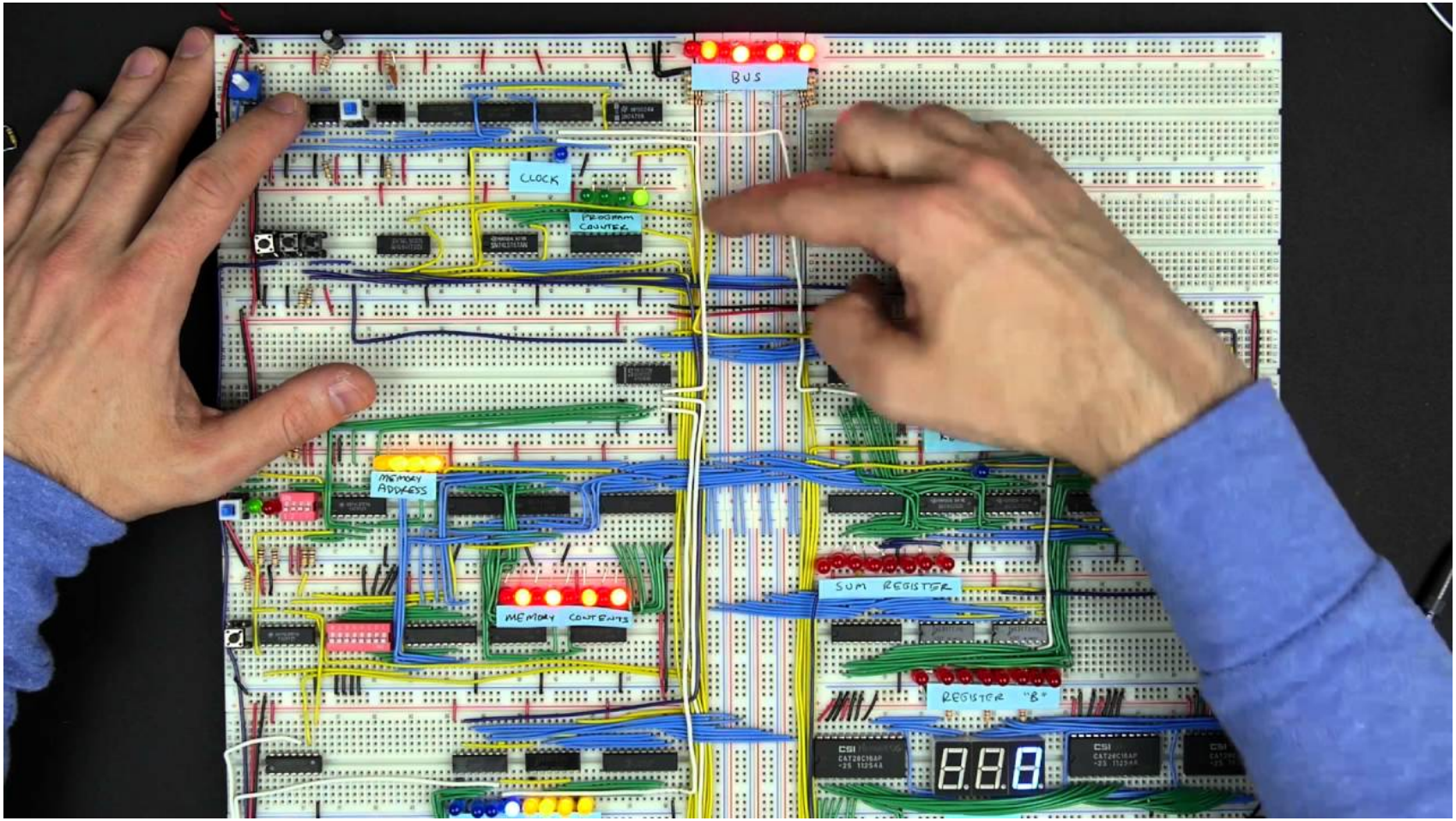
- un ordinateur est un assemblage d'unités communicantes
  (mémoire, registres, additionneurs, décodeur d'instruction…)
- il serait trop coûteux de relier chaque paire d'unités

⇝ communication par *bus* parallèle
(*N* lignes communes à toutes les unités, que chacun peut lire/écrire)

Exemple

- le contenu du registre PC est placé sur le bus
- la mémoire lit cette adresse, et place son contenu sur le bus
- l'unité de contrôle lit l'instruction et la décode
- …

Les bus systèmes



https://youtu.be/HyznrdDSSGM

Bus systèmes Cortex

- Les bus systèmes font partie de la spécification ARM (“AMBA”)
- Protocole maître-esclave ; chaque message a une adresse
  (seuls les maîtres initient la communication, comme I^2C)
- le coeur M4 est un maître, les autres unités sont esclaves
  (parmi les esclaves, la RAM, la flash, les GPIOs)
- comment le DMA peut-il initier une copie ?

Bus systèmes Cortex

- Les bus systèmes font partie de la spécification ARM (“AMBA”)
- Protocole maître-esclave ; chaque message a une adresse
  (seuls les maîtres initient la communication, comme I^2C)
- le coeur M4 est un maître, les autres unités sont esclaves
  (parmi les esclaves, la RAM, la flash, les GPIOs)
- comment le DMA peut-il initier une copie ?
  $\rightsquigarrow$ le DMA est aussi maître

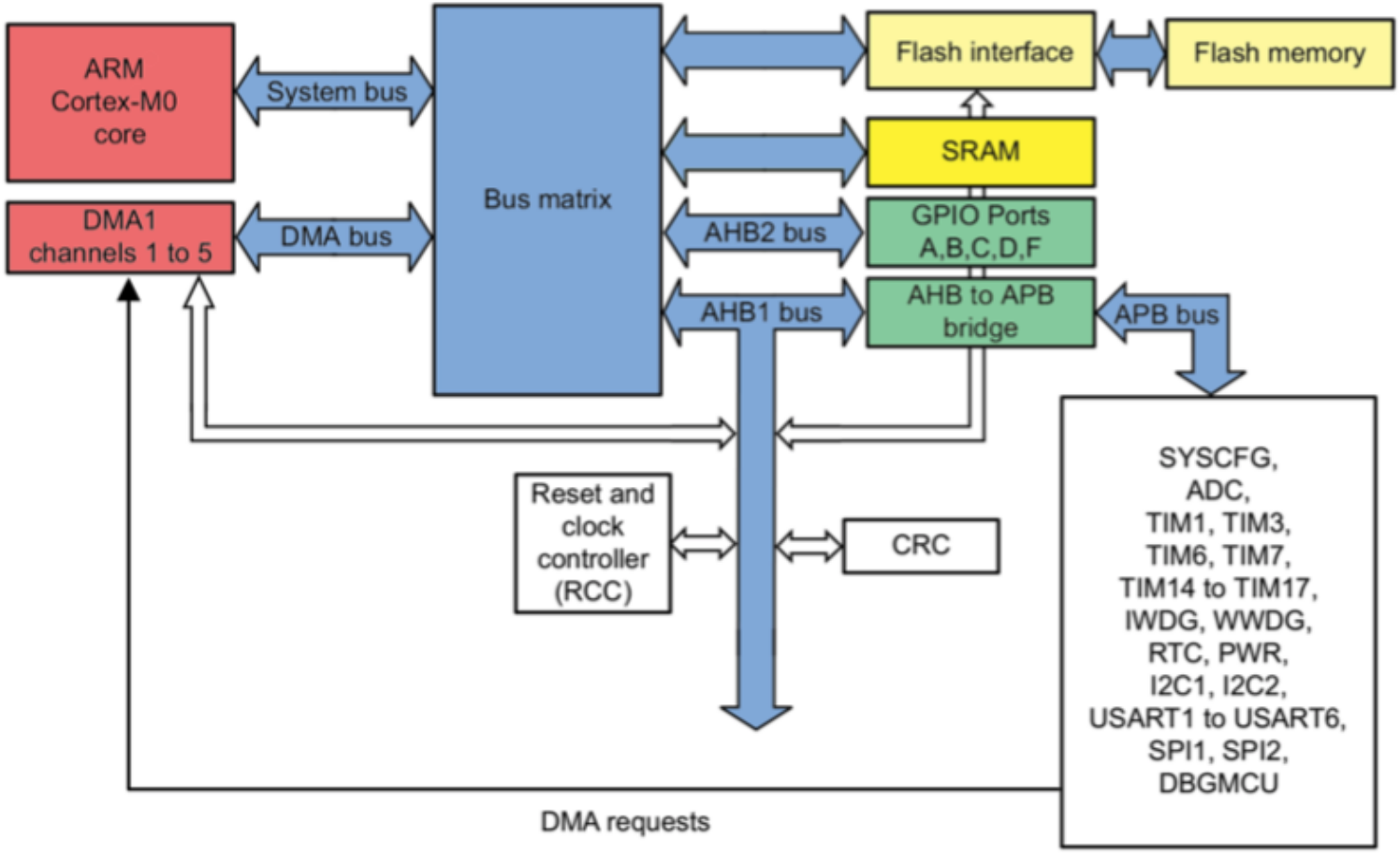
Bus systèmes Cortex

- Les bus systèmes font partie de la spécification ARM (“AMBA”)
- Protocole maître-esclave ; chaque message a une adresse
  (seuls les maîtres initient la communication, comme I^2C)
- le coeur M4 est un maître, les autres unités sont esclaves
  (parmi les esclaves, la RAM, la flash, les GPIOs)
- comment le DMA peut-il initier une copie ?
  $\rightsquigarrow$ le DMA est aussi maître
- une matrice de bus
  - arbitre les requêtes venant du coeur et du DMA
  - suivant leur adresse, les distribue vers : (espace d’adressage)
    - la RAM
    - la flash
    - AHB{1,2} (*Advanced High-performance Bus*) servant les GPIOs
    - APB (*Advanced Peripheral Bus*) relié à AHB1 par un pont
- comment un périphérique peut prévenir le DMA qu’il est prêt ?

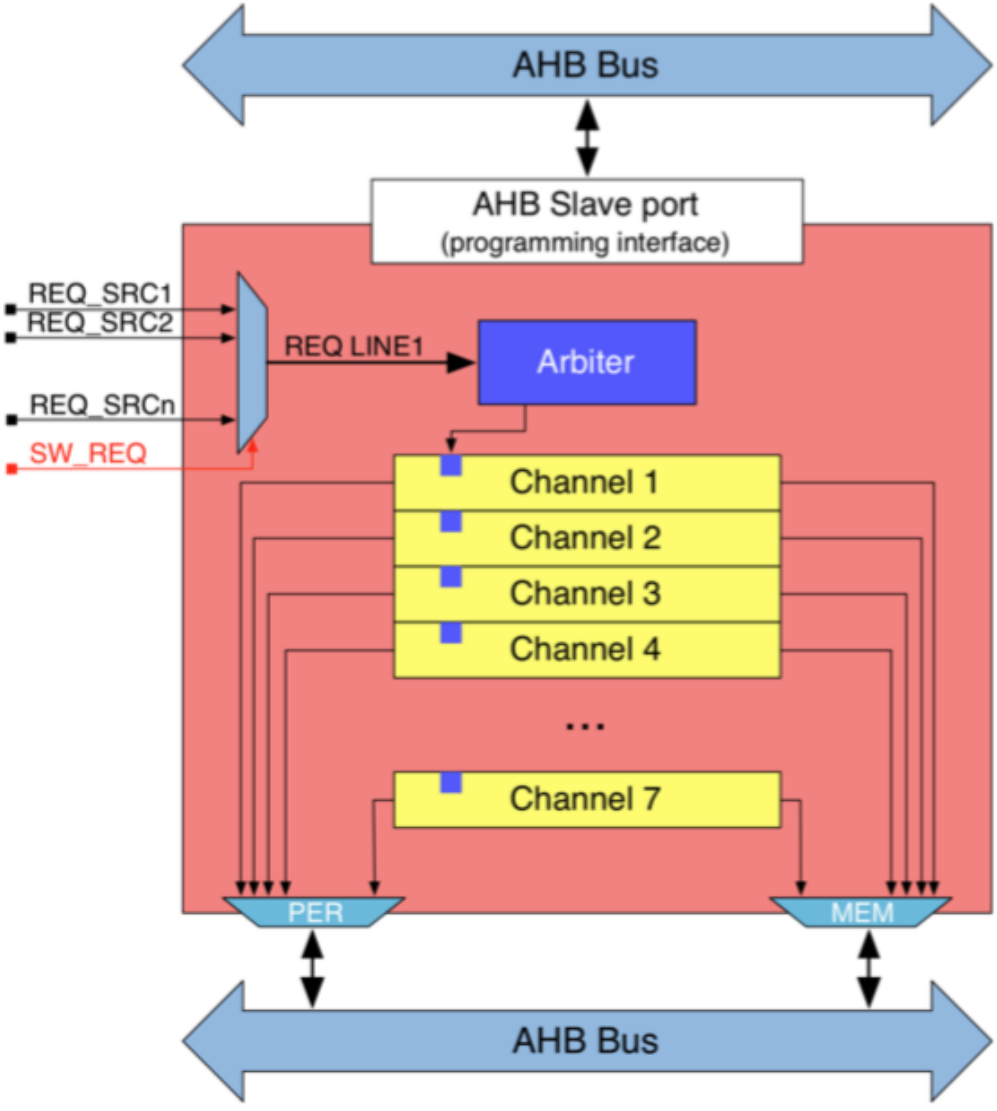
Bus systèmes Cortex

- Les bus systèmes font partie de la spécification ARM (“AMBA”)
- Protocole maître-esclave ; chaque message a une adresse
  (seuls les maîtres initient la communication, comme I^2C)
- le coeur M4 est un maître, les autres unités sont esclaves
  (parmi les esclaves, la RAM, la flash, les GPIOs)
- comment le DMA peut-il initier une copie ?
  ⇝ le DMA est aussi maître
- une matrice de bus
  - arbitre les requêtes venant du coeur et du DMA
  - suivant leur adresse, les distribue vers : (espace d’adressage)
    - la RAM
    - la flash
    - AHB{1,2} (*Advanced High-performance Bus*) servant les GPIOs
    - APB (*Advanced Peripheral Bus*) relié à AHB1 par un pont
- comment un périphérique peut prévenir le DMA qu’il est prêt ?
  ⇝ lignes dédiées périphériques → DMA (“*request lines*”)

Bus systèmes Cortex



Périphériques DMA sur STM32F3



Périphériques DMA sur STM32F3

Périphériques DMA1 et DMA2 :

- maîtres sur AHB, s'adressent à mémoire et à périphériques
- esclaves sur le bus AHB
  (pour configuration via écriture dans registres, comme d'habitude)
- chaque DMA a 8 canaux
  (registre de 32 bits, reçoit un mot de la source, l'envoit vers la cible)
- la copie est déclenchée :
  - en software, ou
  - par une ou plusieurs *DMA request line* venant de périphériques
    (à chaque périphérique correspond une *request line*)
- en cas de compétition entre les *request lines*, un arbitre décide quelle copie sera déclenchée en premier
  (chaque canal a une priorité programmable)

Périphériques DMA sur STM32F3

Caractéristiques

- la configuration comprend notamment :
  - adresse source, adresse cible
  - taille de la donnée (8/16/32 bits)
  - nombre de données N à transférer
  - si l'adresse source/cible doit être incrémentée après copie
- DMA1 propose aussi la copie mémoire → mémoire
- chaque DMA a son IRQ, et peut lever une interruption quand :
  - fin du transfert
  - moitié du transfert effectué
  - erreur ou annulation du transfert
- deux modes :
  - normal (copie de N mots et stop)
  - circulaire (copie de N mots en boucle, sans interruption)

Principe du DMA

Configuration et utilisation avec HAL

HAL Utilisation du DMA

Résumé

Dans l'ordre, on va :

1. allumer et configurer le périphérique auquel le DMA va parler
   (pour enclencher la *request line* quand prêt pour la copie)
2. consulter quel DMA et canal est rattaché au périphérique
   (dans le manuel de référence)
3. allumer et configurer ce DMA, en déclarant une *handle*
   (direction, normal/circulaire, priorité, incrément., taille de mot)
4. "lier" cette *handle* à celui du périphérique
   (spécifité de HAL, permet au périphérique d'installer ses callbacks)
5. activer les interruptions du DMA dans le NVIC
   (et écrire un *handler*)
6. initier le transfert, côté périphérique puis côté DMA

1. Configuration du périphérique en mode DMA

On doit informer le périphérique d'émettre des requêtes DMA (allumer la *DMA request line* quand il a fini de lire/écrire)

1. Configuration du périphérique en mode DMA

On doit informer le périphérique d'émettre des requêtes DMA (allumer la *DMA request line* quand il a fini de lire/écrire)

Exemple (U(S)ART)

- dans le registre de configuration CR3, un bit à allumer : DMAR
- on peut directement utiliser les fonctions HAL :

  `HAL_UART_Receive_DMA()`
  `HAL_UART_Transmit_DMA()`

  en place de celles déjà vues ; elles :
  - activent le bit DMAR,
  - mettent en place les même *callbacks* que l'on a vu :

    `HAL_UART_{T|R}xCpltCallback()`
    `HAL_UART_{T|R}xHalfCpltCallback()`
    `HAL_UART_ErrorCallback()`
  - lancent la transmission

2,3. Configuration du DMA

Comme d'habitude :

- on déclare une *handle*
  (contient configuration + état courant du DMA)
- on initialise ses champs de configuration :

  `Instance` `DMAx_Channel_y`
  `Init.Direction` de mémoire vers périphérique ou l'inverse
  `Init.{Periph|Mem}Inc` doit-on incrémenter l'adresse source/cible de copie après chaque copie ?
  `Init.{Periph|Mem}DataAlignment` quelle taille pour chaque copie ? 8/16/32 bits
  `Init.Mode` normal (1 copie) ou circulaire (en continu)
  `Init.Priority` priorité du canal DMA (4 niveaux)
- on la passe à `HAL_DMA_Init()`
- on spécifiera adresse source/cible et taille de copie plus tard
  (dans `HAL_DMA_Start()`)

4. Lien de la *handle*

Lien de la *handle*

Spécificité de HAL : on doit copier l'adresse de la *handle* DMA dans un champ de la *handle* périphérique. HAL fournit une macro pour cela.

Exemple

Pour l'UART :

```
__HAL_LINKDMA(&huart, hdmatx, hdma);
```

(`huart` = handle UART ; `hdma` = handle DMA ; `hdmatx` = champ)

5. Activation de l'interruption DMA

Activation des interruptions DMA

- activation :
  `HAL_NVIC_EnableIRQ(DMAx_Channely_IRQn);`
- définition d'un *handler* qui appelle le handler HAL

```
void DMA1_Channel4_IRQHandler() {
  HAL_DMA_IRQHandler(&hdma);
}
```

- ce handler HAL fait le *dispatch* et appelle à son tour les *callbacks* définis pour le périphérique

6. Initiation du transfert

- ... côté périphérique
  (ex : pour UART, on appelle `HAL_UART_Transmit_DMA()`)
- et côté DMA avec la fonction

```
HAL_StatusTypeDef HAL_DMA_Start(
  DMA_HandleTypeDef *hdma,
  uint32_t SrcAddress,
  uint32_t DstAddress,
  uint32_t DataLength)
```

`hdma` le *handle* du DMA à démarrer

`SrcAddress` l'adresse source de copie
(ex : l'adresse d'un tableau en RAM à envoyer)

`DstAddress` l'adresse cible de copie
(ex : l'adresse du registre de données à transférer de l'UART : `USART1->TDR`)

`DataLength` le nombre de copies à effectuer

6. Initiation du transfert

Attention, bug de HAL !

Toujours appeler `HAL_DMA_Start()` *après* avoir initié le transfert côté périphérique

Problème

On cherche à envoyer sur l'UART1 les octets correspondants à la suite de fibonacci modulo 256 :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 121, 98, 219, 61, 24, 85,
109, 194, 47, 241, 32, 17, 49, 66, 115, 181, 40, 221, 5, 226, 231, 201...
```

Problème

On cherche à envoyer sur l'UART1 les octets correspondants à la suite de fibonacci modulo 256 :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 121, 98, 219, 61, 24, 85,
109, 194, 47, 241, 32, 17, 49, 66, 115, 181, 40, 221, 5, 226, 231, 201...
```

- à 921 600 baud (1Mo/s)

Problème

On cherche à envoyer sur l'UART1 les octets correspondants à la suite de fibonacci modulo 256 :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 121, 98, 219, 61, 24, 85,
109, 194, 47, 241, 32, 17, 49, 66, 115, 181, 40, 221, 5, 226, 231, 201...
```

- à 921 600 baud (1Mo/s)
- à vitesse constante !

Problème

On cherche à envoyer sur l'UART1 les octets correspondants à la suite de fibonacci modulo 256 :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 121, 98, 219, 61, 24, 85,
109, 194, 47, 241, 32, 17, 49, 66, 115, 181, 40, 221, 5, 226, 231, 201...
```

- à 921 600 baud (1Mo/s)
- à vitesse constante !

```
void fibonacci(uint8_t *buffer, int length) {
  static uint8_t a=0, b=1;
  while (length--) {
    uint8_t tmp = b;
    b += a;
    *buffer++ = a = tmp;
  }
}
```

Problème

Non-solution 1

Envoyer octet par octet, avec interruptions :

```
HAL_UART_Transmit_IT(huart, &counter++, 1);
```

Problème

Non-solution 1

Envoyer octet par octet, avec interruptions :

```
HAL_UART_Transmit_IT(huart, &counter++, 1);
```

⇝ impossible : une interruption coûte ≈ 20 cycles

Problème

Non-solution 1

Envoyer octet par octet, avec interruptions :

`HAL_UART_Transmit_IT(huart, &counter++, 1);`

$\rightsquigarrow$ impossible : une interruption coûte $\approx$ 20 cycles

Non-solution 2

Envoyer par bloc de 64 octets, avec DMA en mode normal :

```
int main() {
  /* ... */
  HAL_UART_Transmit_DMA(&huart, buffer, 64)
  HAL_DMA_Start(&hdma, buffer, USART1->TDR, 64);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
  /* calcul des 64 valeurs suivantes dans buffer */
  fibonacci(buffer+0, 64);
  HAL_DMA_Start(&hdma, buffer, USART1->TDR, 64);
}
```

Problème

Non-solution 1

Envoyer octet par octet, avec interruptions :

`HAL_UART_Transmit_IT(huart, &counter++, 1);`

⇝ impossible : une interruption coûte ≈ 20 cycles

Non-solution 2

Envoyer par bloc de 64 octets, avec DMA en mode normal :

```
int main() {
  /* ... */
  HAL_UART_Transmit_DMA(&huart, buffer, 64)
  HAL_DMA_Start(&hdma, buffer, USART1->TDR, 64);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
  /* calcul des 64 valeurs suivantes dans buffer */
  fibonacci(buffer+0, 64);
  HAL_DMA_Start(&hdma, buffer, USART1->TDR, 64);
}
```

⇝ vitesse non constante : pause quand recalcul

Problème

Non-solution 3

Envoyer par bloc de 64 octets, avec DMA en mode continu :

```
int main() {
  /* ... */
  HAL_UART_Transmit_DMA(&huart, buffer, 64)
  HAL_DMA_Start(&hdma, buffer, USART1->TDR, 64);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
  /* calcul des 64 valeurs suivantes */
  fibonacci(buffer+0, 64);
}
```

⇝ le DMA continue à lire l'ancien buffer pendant qu'on écrit les valeurs suivantes dedans

Solution Double buffering

- Envoyer par bloc de 128 octets
- avec le DMA en mode continu
- à la fin d'un transfert (dans `HAL_UART_TxCpltCallback`), on commence à calculer `fibonacci(buffer+64, 64)`
- à la moitié du transfert, (dans `HAL_UART_TxHalfCpltCallback`) on commence à calculer `fibonacci(buffer+0, 64)`

⇝ on alterne :

  - quand le DMA est en train de lire la première moitié du buffer, on écrit dans la seconde
  - quand le DMA est en train de lire la seconde moitié du buffer, on écrit dans la première