M2 - ESE

Cours Architecture des Processeurs pour l'Embarqué

2025-2026

Chapitre 3. Exploitation des processeurs ARM-Cortex

- Configuration des ports d'entrées/sorties
- Gestion des interruptions
- Communication UART
- Gestion de la DMA
- Gestion de l'horloge
- Utilisation des Timers
- Conversion analogique numérique
- Conversion numérique analogique
- Communication I²C
- Communication SPI
- Calcul des CRC (Cyclic Redundancy Check)
- Utilisation du Watchdog Timer
- L'horloge temps réel

1. Configuration des ports d'entrées/sorties

A. Objectif

Programmer chaque broche (pin) du microcontrôleur pour qu'elle fonctionne comme :

- **Entrée :** lire niveau logique 0/1 (bouton, capteur) : vérifier si tension = **0V (0)** ou **3.3V (1)**

Bouton pressé → Masse (0V) → GPIO lit 0

Bouton relâché → Pull-up (3.3V) → GPIO lit 1

- **Sortie :** piloter 0/1 (LED, relais)
- **Fonctions alternatives :** UART, SPI, I2C...

GPIO = General Purpose Input/Output (Entrée/Sortie à Usage Général)

Le rôle fondamental des ports E/S

Les ports d'entrées/sorties représentent l'interface physique entre le microcontrôleur et le monde réel. Ce sont des broches électroniques capables de lire des signaux électriques (boutons, capteurs) ou d'en générer (LED, relais, moteurs). Sur un STM32F407, ces ports sont organisés en 7 groupes (A, B, C, D, E, F, G) contenant chacun 16 broches numérotées de 0 à 15, soit plus de 140 connexions possibles vers l'extérieur.

La métamorphose programmable d'une broche

Chaque broche possède une capacité unique : elle peut changer de fonction par simple programmation. Une broche PA0 peut être configurée comme entrée pour lire un capteur de température, puis reprogrammée comme sortie pour commander une LED, ou encore comme entrée analogique pour mesurer une tension. Cette flexibilité universelle est la signature des microcontrôleurs modernes face aux circuits fixes traditionnels.

La séquence électrique obligatoire

Avant toute utilisation, il faut d'abord activer l'alimentation électrique du port concerné via le contrôleur d'horloge (RCC). Sans cette étape, les broches restent inactives, comme une maison sans courant. Ensuite, chaque broche doit être explicitement configurée dans son mode de fonctionnement : entrée numérique, sortie numérique, fonction alternative (UART, SPI) ou entrée analogique. Cette configuration se fait bit par bit dans un registre de 32 bits contrôlant les 16 broches du port.

Entrée vs Sortie : deux mondes opposés

En mode entrée, la broche devient un détecteur sensible aux variations de tension : 0V (logique 0) ou 3.3V (logique 1). Un bouton connecté à la masse génère un 0, tandis qu'un capteur actif produit un 1. En mode sortie, la broche devient un générateur actif capable d'imposer sa tension sur le circuit externe : 0V pour éteindre une LED, 3.3V pour l'allumer. La vitesse de commutation peut aussi être ajustée selon l'usage (2MHz pour boutons, 100MHz pour signaux rapides).

Les résistances de pull-up/pull-down internes

Pour éviter les états flottants parasites, chaque broche dispose de résistances internes pilotables par logiciel. Le pull-up connecte la broche à 3.3V par une résistance de 40kΩ, parfait pour les boutons : relâché=1, pressé=0. Le pull-down la relie à la masse. Ces résistances éliminent le besoin de composants externes, simplifiant les prototypes et réduisant les coûts industriels.

L'interrogation et la commande des états

Une broche entrée se lit via un registre contenant l'état logique de toutes les 16 broches du port. Pour isoler une broche spécifique, on utilise un masque binaire qui ne laisse passer que l'information désirée. Une broche sortie se commande en écrivant directement 0 ou 1 dans un registre de sortie, avec possibilité d'inverser l'état (toggle) en un seul cycle processeur.

Définition simple

Les GPIO sont les "broches intelligentes" d'un microcontrôleur qui peuvent être configurées par logiciel pour :
text

ENTRÉE → Lire 0/1 (bouton, capteur)
SORTIE → Écrire 0/1 (LED, relais)
Architecture 1 GPIO (exemple PA0)
text
[Broche physique PA0]
↑↓
[Buffer entrée] ← Tension 0V/3.3V → [Registre IDR bit 0]
↑↓ ↑
[Registre MODER] ← CPU configure → [Registre ODR bit 0] → [Buffer sortie]
"00"=Entrée "01"=Sortie
PA0 = Broche 0 du Port A (GPIO Port A Pin 0)
text
PA0 = Première broche (pin) du Port GPIOA
P = Port
A = Port A (1er port)
0 = Broche numéro 0 (de 0 à 15)
Position Physique STM32F407
text
STM32F407VGT6 LQFP100 :
Pin physique #23 = PA0
Pin 23 = PA0
PA0
Pin 24 = PA1
Pin 25 = PA2
Caractéristiques PA0
Fonction Description
GPIO Entrée/Sortie générale
ADC Canal ADC12_IN0 (capteur analogique)
UART USART2_CTS (contrôle flux)
Timer TIM2_CH1 (PWM/capture)
Tension 0-3.3V (FT = 5V tolérant)
Code exemple PA0
c
```
// PA0 = ENTRÉE (bouton)
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
GPIOA->MODER &= ~0x00000003;      // 00 = entrée
GPIOA->PUPDR |= 0x00000001;       // Pull-up

// Lire PA0
if(GPIOA->IDR & 0x00000001) {     // Bit 0 = 1 ?
   // Bouton relâché
}

// PA0 = SORTIE (LED)
GPIOA->MODER |= 0x00000001;       // 01 = sortie
```

GPIOA->ODR |= 0x00000001; // PA0 = 1 (3.3V)
Comment retenir tous les PAx ?
text
PORT A : PA0 PA1 PA2 PA3 ... PA15
↑
16 broches

Masque binaire :
PA0 : 0x00000001 (bit 0 = 0b1)
PA1 : 0x00000002 (bit 1 = 0b10)
PA2 : 0x00000004 (bit 2 = 0b100)
...
PA15: 0x00008000 (bit 15)
Sur STM32F4-Discovery
text
PA0 = Broche accessible (header CN10 pin 1)
Idéal pour : Capteur température, potentiomètre

4 Modes possibles (MODER)

Mode	Bits MODER	Usage	Exemple STM32
00	Entrée	Lire boutons	GPIOA->MODER &= ~0x3;
01	Sortie	Piloter LED	`GPIOA->MODER
10	Alternate	UART/SPI	PA9=USART1_TX
11	Analogique	ADC capteur	PA0=ADC_IN0

Exemple concret STM32F407
text
PA0 configurée ENTRÉE :
1. GPIOA->MODER &= ~0x00000003; // Bits 1:0 = 00
2. uint32_t etat = GPIOA->IDR & 1; // Lecture bit 0
text
PA0 configurée SORTIE :
1. GPIOA->MODER |= 0x00000001; // Bits 1:0 = 01
2. GPIOA->ODR |= 1; // PA0=1 (3.3V)

Applications réelles (votre domaine)

Composant	GPIO Mode	Broche exemple
LED état	Sortie	PD12 (verte Discovery)
Bouton	USER	Entrée Pull-up PC13
Capteur temp°		Entrée PA0
Relais irrigation	Sortie	PB5
Fin de course robot	Entrée Pull-down	PA1

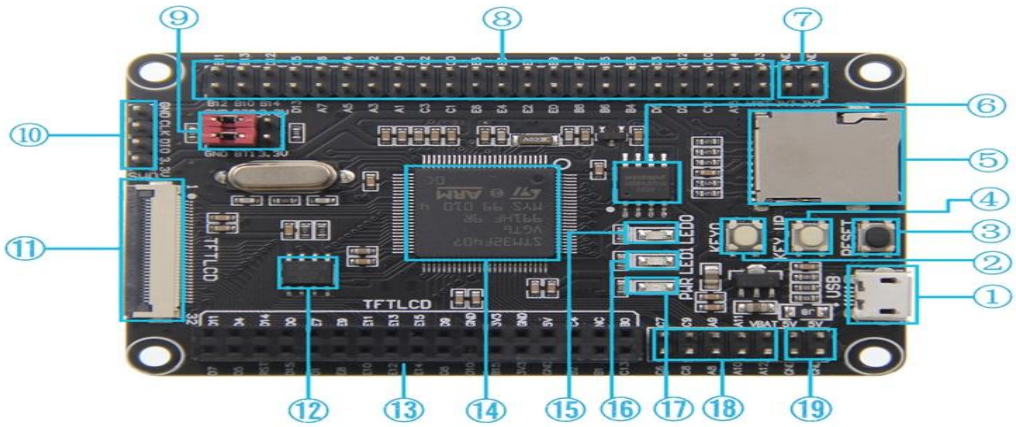
Pourquoi "General Purpose" ?
text
1 PIN GPIO = 4 FONCTIONS possibles
= Économie broches = Compacité carte
= Flexibilité maximale projet

B. Vue d'ensemble STM32F407VGT6

Caractéristiques du produit

- Petit et élégant, pratique pour divers projets de bricolage
- Compatible avec deux processeurs : STM32F407VGT6 et STM32F407VET6
- La fréquence principale du processeur atteint 168 MHz, ce qui le rend rapide et puissant.
- Ressources de stockage importantes : SRAM interne : 192 Ko, mémoire flash interne : 1 Mo (STM32F407VGT6), 512 Ko (STM32F407VET6), mémoire flash SPI externe : 8 Mo, EEPROM externe : 256 octets
- La carte SD permet d'étendre l'espace de stockage.
- Prise en charge de l'insertion des broches LCD et de la connexion par câble flexible FPC
- Prise en charge de l'utilisation du bus FSMC pour piloter l'écran LCD, vitesse d'affichage rapide
- Prise en charge du téléchargement SWD et par port série, débogage simplifié
- Prise en charge de la communication USB
- Étendez le port GPIO pour connecter facilement divers périphériques.
- Les ressources de développement sont nombreuses, faciles à apprendre et faciles à mettre en œuvre.
- Fournir des informations de développement riches

C. Description du matériel



Nombre	Nom du matériel	Description
①	interface de communication USB	Elle sert à la communication USB entre la carte mère minimale et le PC, notamment en mode esclave (la carte mère minimale faisant office de clé USB) et en mode hôte (la carte mère minimale faisant office d'hôte). Cette interface peut également servir d'interface d'alimentation (sauf en mode hôte).

②	Bouton de test Key0	En tant que bouton de test standard, il peut être utilisé pour la fonction de saisie de touches dans le programme.
③	Bouton de réinitialisation	Il sert à réinitialiser la puce principale. De plus, il permet également de réinitialiser l'écran LCD TFT lorsqu'il est inséré directement ou lorsqu'un câble flexible FPC est utilisé.
④	Touche de test/réveil	Permet de sortir la puce principale du mode veille. Si la fonction de sortie de veille n'est pas utilisée, cette touche peut servir d'entrée normale.
⑤	emplacement pour carte SD	Elle sert à insérer une carte SD et à étendre l'espace de stockage de données de la plus petite carte mère.
⑥	Flash SPI W25Q64	Il sert à stocker les polices de caractères, les images et autres données utilisateur, et à étendre l'espace de stockage des données.
⑦	Broche d'entrée/sortie d'alimentation 3,3 V	Il sert à fournir une alimentation de 3,3 V à un périphérique externe et peut également connecter une alimentation de 3,3 V à la carte mère minimale depuis l'extérieur.
⑧	extension d'E/S	Utilisé pour connecter des périphériques externes.
⑨	Port de sélection de démarrage BT0 / BT1	Il sert à sélectionner le mode de démarrage après la réinitialisation de la puce principale.
⑩	Interface de simulation de téléchargement SWD	Il sert à télécharger et à simuler la configuration minimale de la carte mère.
⑪	Interface en ligne pour câble plat TFT_LCD FPC	Utilisé pour connecter un module LCD via un câble flexible FPC.
⑫	EEPROM 24C02	Il est utilisé pour accéder à des données qui ne peuvent pas être perdues après une panne de courant et nécessite moins de précision en termes de vitesse et de durée de lecture.
⑬	Interface en ligne TFT_LCD	Pour l'insertion directe d'un module LCD via un réseau de broches
⑭	Puce principale STM32F407VGT6/STM 32F407VET6	Utilisé pour exécuter le programme. Sa fréquence principale est de 168 MHz, sa mémoire SRAM interne est de 192 Ko et sa mémoire flash interne est de 1 Mo (STM32F407VGT6) ou de 512 Ko (STM32F407VET6).
⑮	Lampe de test LED0 (bleue)	Il peut être utilisé pour tester les sorties GPIO, indiquer l'état d'exécution d'un programme ou réaliser certains effets lumineux.
⑯	Lampe de test LED1 (bleue)	Il peut être utilisé pour tester les sorties GPIO, indiquer l'état d'exécution d'un programme ou réaliser certains effets lumineux.

⑰	Indicateur d'alimentation (rouge)	Utilisé pour indiquer la mise sous tension minimale de la carte mère.
⑱	extension d'E/S	Utilisé pour connecter des périphériques externes.
⑲	Broche d'entrée/sortie d'alimentation 5V	Il sert à fournir une alimentation 5V à un périphérique externe et peut également connecter une alimentation 5V à la carte mère minimale depuis l'extérieur.

D. Modèles d'écrans LCD compatibles

- 2,4 pouces - MRB2408
- 2,8 pouces - MRB2801
- 2,8 pouces - MRB2802
- 3,2 pouces - MRB3205
- 3,5 pouces - MRB3503
- 3,5 pouces - MRB3511
- 3,5 pouces - MRB3512
- 3,97 pouces - MRB3973
- 4 pouces - MRB3951
- 4 pouces - MRB3952

E. Architecture GPIO STM32

text
STM32F407 : 7 ports GPIO (A,B,C,D,E,F,G)
Port A : PA0 ↔ PA15 (16 broches)
Port B : PB0 ↔ PB15
Total : 140+ broches GPIO

Étapes configuration (SÉQUENCE OBLIGATOIRE)

1. Activation Clock du port (RCC)

c
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Clock GPIOA ON
// Bits équivalents :
RCC_AHB1ENR_GPIOBEN // Port B
RCC_AHB1ENR_GPIOCEN // Port C
ERREUR FATALE : Oubli = GPIO ne répond pas !

2. Mode des broches (REGISTRE MODER)

32 bits = 16 broches × 2 bits chacune
text
MODER15[1:0] MODER14[1:0] ... MODER0[1:0]
| 2 bits | 2 bits | |
Codage (par broche) :
text

00 = Entrée (Input)
01 = Sortie (Output 2MHz)
10 = Fonction alternative (UART/SPI...)
11 = Analogique (ADC)
Exemple PA0-PA1 :
c
```
GPIOA->MODER |= 0x00000001;       // PA0 = 01 (Sortie)
                                  // Bits 1:0 = 01
GPIOA->MODER &= ~0x0000000C;      // PA1 = 00 (Entrée)
                                  // Bits 3:2 = 00 (masque 1100)
```
Vitesse sortie (OSPEEDR) :
text
00 = 2MHz 01 = 25MHz
10 = 50MHz 11 = 100MHz

3. Sortie logique (ODR)

c
```
GPIOA->ODR |= 0x00000001;         // PA0 = 1 (LED ON)
GPIOA->ODR &= ~0x00000001;        // PA0 = 0 (LED OFF)
```
4. Lecture entrée (IDR)
c
```
if (GPIOA->IDR & 0x00000002) {    // PA1 = 1 ?
  // Bouton pressé
}
```
Exemple COMPLET LED + Bouton
c

// 1. Clock

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN;
```

// 2. PA0 = Sortie LED (PC13 sur STM32F4DISCOVERY)

```
GPIOA->MODER &= ~0x00000003;      // Reset bits PA0
GPIOA->MODER |= 0x00000001;       // PA0 sortie
```

// 3. PC13 = Entrée bouton (Pull-up)

```
GPIOC->MODER &= ~0x00003000;      // PC13 entrée
GPIOC->PUPDR &= ~0x00003000;      // No pull-up
GPIOC->PUPDR |= 0x00001000;       // Pull-up PC13
```

// 4. Boucle principale

```
while(1) {
  if (GPIOC->IDR & 0x2000) {      // Bouton relâché ?
    GPIOA->ODR |= 0x00000001;     // LED ON
  } else {
    GPIOA->ODR &= ~0x00000001;    // LED OFF
```

```
  }
}
```

Applications pratiques

Broche	Usage	Configuration
PA0	LED	Sortie 2MHz
PB0	Bouton	Entrée Pull-up
PC13	Bouton USER	Entrée Pull-up
PB5	Fin de course	Entrée Pull-down
PA8	Relais	Sortie 50MHz

Astuces avancées

Toggle rapide :

```
c
GPIOA->ODR ^= 0x00000001;  // Inverse PA0 (0→1 ou 1→0)
```

Bit-banding (accès ultra-rapide) :

```
text
0x40013C07  // Bit-band PA0 (équiv. ODR bit 0)
```

Pull-up/Pull-down :

```
text
PUPDR : 00=None, 01=Pull-up, 10=Pull-down
```

Ce module = BASE de 90% des projets embarqués

2. Gestion des interruptions (NVIC) :

l'objectif est de Réagir instantanément aux événements (bouton pressé, timer overflow)

Qu'est-ce qu'une interruption ?

Une interruption est un signal électrique qui stoppe immédiatement le programme principal pour exécuter une tâche urgente. Imaginez que vous conduisez et soudain votre téléphone sonne : vous arrêtez de conduire (programme principal), répondez (interruption), puis reprenez la route. Dans un microcontrôleur, un bouton pressé ou un capteur qui détecte un danger déclenche cette interruption. Le processeur Cortex-M4 réagit en 1 microseconde au lieu d'attendre en boucle (polling).

Pourquoi utiliser les interruptions ?

Sans interruptions, le microcontrôleur passe 99% de son temps à vérifier inutilement "le bouton est-il pressé ?". Les interruptions libèrent le CPU pour d'autres tâches importantes comme calculer des données médicales ou surveiller une irrigation. Résultat : le système devient réactif (réaction immédiate) et efficace (CPU disponible).

Rôle du NVIC (Nested Vectored Interrupt Controller)

Le NVIC est le "chef d'orchestre" des interruptions dans le Cortex-M4. Il gère 81 interruptions différentes simultanément avec 16 niveaux de priorité. Si un capteur ECG détecte une crise (priorité 1) pendant que l'utilisateur appuie sur un bouton

(priorité 5), le NVIC exécute d'abord l'ECG. Il "vecte" automatiquement vers la bonne routine sans perdre de temps.

Les 4 étapes obligatoires de configuration

Premièrement, activez l'alimentation électrique (clock) des périphériques concernés via le RCC. Deuxièmement, configurez la ligne d'interruption EXTI pour associer une broche physique (PA0) à un signal d'interruption. Troisièmement, informez le NVIC qu'il doit surveiller cette interruption avec une priorité spécifique. Enfin, écrivez la routine d'interruption qui s'exécutera automatiquement quand l'événement se produit.

Comment détecte-t-on un événement ?

Les interruptions EXTI surveillent les fronts (changements d'état) sur les broches : front montant (0→1) ou front descendant (1→0). Pour un bouton avec pull-up, le front descendant (bouton pressé = 0V) déclenche l'interruption. Le SYSCFG relie physiquement la broche PA0 à la ligne logicielle EXTI0.

La routine d'interruption (ISR)

Quand l'interruption se déclenche, le Cortex-M4 sauvegarde automatiquement l'état du programme principal, saute à l'adresse de votre routine EXTI0_IRQHandler, exécute votre code (toggle LED), puis revient exactement là où il s'était arrêté. Vous devez impérativement effacer le flag d'interruption pour éviter les boucles infinies.

Gestion des priorités

Les 16 niveaux de priorité (0=urgente, 15=peu urgente) permettent d'arbitrer les conflits. Dans un système de monitoring santé, l'interruption ECG (priorité 0) primerait sur l'interruption UART debug (priorité 10). Le NVIC interrompt même une interruption en cours si une plus prioritaire arrive.

Contenu :

text

```
NVIC_EnableIRQ(EXTI0_IRQn);          // Activation IT EXTI0
EXTI->IMR |= EXTI_IMR_MR0;           // Masque ligne 0
```

IRQ Handler :

c

```
void EXTI0_IRQHandler(void) {
  if(EXTI->PR & EXTI_PR_PR0) {
    // Action bouton pressé
    EXTI->PR |= EXTI_PR_PR0;        // Clear flag
  }
}
```

Priorités : NVIC 16 niveaux

3. Communication UART (USART)

Objectif : Communication série PC-microcontrôleur (115200 bauds)

Le langage universel des machines

L'UART, ou Universal Asynchronous Receiver Transmitter, est le système de communication série le plus répandu dans l'électronique. Il permet à deux appareils électroniques de converser en envoyant des caractères (A, B, C...) un par un via deux fils seulement : un fil TX (transmission) et un fil RX (réception). Contrairement à une conversation téléphonique synchronisée, l'UART fonctionne sans horloge commune, d'où le terme "asynchrone". Chaque caractère est précédé d'un bit de départ et suivi de bits de parité et d'arrêt pour la synchronisation.

Anatomie d'un octet voyageur

Chaque caractère envoyé commence par un bit de départ à 0 qui aligne les récepteurs, suivi de 8 bits de données (un octet complet), un bit de parité optionnel pour détecter les erreurs, et un ou deux bits d'arrêt à 1. Cette trame de 10 à 12 bits voyage à une vitesse fixe appelée baudrate, typiquement 115200 bits/seconde pour les communications PC-microcontrôleur. À cette vitesse, un octet complet transite en environ 87 microsecondes.

La cadence des échanges bidirectionnels

L'UART fonctionne en duplex intégral : TX et RX opèrent simultanément et indépendamment. Le microcontrôleur peut envoyer "HELLO" vers un terminal PC pendant qu'il reçoit simultanément des commandes utilisateur. Cette duplexité est cruciale pour les applications de débogage où l'on envoie des données de diagnostic tout en recevant des ordres de configuration en temps réel.

Configuration de la danse temporelle

Avant toute communication, émetteur et récepteur doivent s'accorder sur trois paramètres essentiels : le baudrate (vitesse), la longueur des données (7 ou 8 bits), et le format de trame (parité, bits d'arrêt). Le STM32F407 génère cette synchronisation en divisant sa fréquence d'horloge interne (souvent 16MHz) par un diviseur précis pour obtenir le baudrate désiré. Une erreur de configuration provoque une réception de caractères illisibles ou du silence total.

Applications critiques en systèmes embarqués

Dans les projets de santé par exemple, l'UART transmet les données ECG vers un terminal médical ou un smartphone pour analyse temps réel. En agriculture intelligente, il envoie les mesures d'humidité et de température vers un serveur central. Pour le débogage, l'UART affiche les variables internes du microcontrôleur sur un terminal PC (PuTTY, TeraTerm), permettant de diagnostiquer des pannes sans oscilloscope coûteux.

Buffer circulaire et gestion du flux

Pour éviter la perte de données lors d'échanges rapides, l'UART intègre des tampons FIFO internes (tampon d'entrée/sortie). Le microcontrôleur vérifie si le tampon d'émission est vide avant d'envoyer et lit le tampon de réception dès qu'un nouveau caractère arrive. En cas de surcharge, des drapeaux de débordement signalent les

données perdues, garantissant la robustesse des communications critiques comme la télémédecine.

USART vs UART : la version avancée

Le STM32F407 embarque des périphériques USART (Universal Synchronous Asynchronous Receiver Transmitter), évolution de l'UART simple. L'USART ajoute la capacité synchrone (avec horloge) et des débits plus élevés, mais conserve la compatibilité UART pour les usages traditionnels. Les USART1/2/3 du STM32F407 offrent jusqu'à 10.5Mbit/s en mode synchrone, idéal pour les communications industrielles rapides.
L'UART = système nerveux du microcontrôleur : il relie le cerveau électronique au monde extérieur !
Configuration :
text
USART1->BRR = 0x0683; // Baudrate 115200@16MHz
USART1->CR1 |= USART_CR1_TE | USART_CR1_RE | USART_CR1_UE;
TX/RX :
c
while(!(USART1->SR & USART_SR_TXE)); // Attendre TX vide
USART1->DR = 'A'; // Envoi
Applications : Debug, terminal Putty, commandes PC

4. Gestion de la DMA

Objectif : Transfert mémoire-mémoire/périphérique sans CPU
Avantages : CPU 100% disponible pendant UART/SPI/ADC

Le DMA libère le processeur

La DMA, ou Direct Memory Access, est un périphérique hardware qui transfère des données directement entre la mémoire et les périphériques sans intervention du CPU. Imaginez un commis de cuisine qui remplit automatiquement les assiettes depuis la chambre froide pendant que le chef (CPU) prépare la sauce.

Transferts automatiques en arrière-plan

Contrairement au CPU qui doit lire un capteur ADC octet par octet, la DMA configure une fois le canal de transfert puis opère seule. Elle peut déplacer 65535 échantillons d'un capteur vers un buffer mémoire en quelques millisecondes, laissant le processeur libre pour l'intelligence artificielle ou l'affichage. Le DMA signale sa fin de transfert par une interruption, réveillant le CPU seulement quand les données sont prêtes à être analysées.

Canaux spécialisés par périphérique

Le STM32F407 possède deux contrôleurs DMA (DMA1/DMA2) avec 8 et 12 canaux respectivement, chacun dédié à un périphérique spécifique : canal 4 pour UART1 TX, canal 6 pour ADC1, canal 0 pour SPI1... Cette spécialisation hardware garantit des

performances optimales sans conflit. Un même buffer peut être transféré simultanément vers l'UART (debug) et le SPI (carte SD) via deux canaux différents.

Modes de transfert flexibles

La DMA supporte trois modes : mémoire-mémoire (copie rapide tableau), périphérique-mémoire (acquisition ADC continue), et mémoire-périphérique (envoi UART automatique). Le mode circulaire recycle automatiquement le buffer quand il est plein, parfait pour l'enregistrement continu de données médicales ou agricoles. Les compteurs de transfert et d'incrémentation gèrent automatiquement les adresses sources et destinations.

Réduction drastique de la charge CPU

Sans DMA, une acquisition ADC à 1MHz occupe le CPU à 100% pour déplacer les données. Avec DMA circulaire, le CPU n'intervient qu'une fois par seconde pour analyser le buffer complet. Cette économie de cycles processeur multiplie par 10 la capacité de calcul disponible pour les algorithmes d'intelligence artificielle appliquée à la santé ou à l'agriculture intelligente.

Synchronisation maître-esclave

La DMA peut être synchronisée par un périphérique maître (timer, ADC) qui déclenche chaque transfert. Par exemple, un timer génère une impulsion toutes les 100µs qui ordonne à la DMA de lire un échantillon ADC. Cette précision temporelle est cruciale pour les systèmes temps réel comme la détection de crises d'épilepsie où chaque échantillon doit être acquis à intervalles réguliers.

La DMA transforme le CPU en cerveau stratégique au lieu d'esclave de transfert !

text

```
DMA2_Stream7->CR |= DMA_SxCR_EN;     // Activation canal UART TX
DMA2_Stream6->CR |= DMA_SxCR_EN;     // Canal UART RX
```

Applications : Acquisition ADC continue, buffer UART circulaire

5. Gestion de l'horloge (RCC)

Objectif : Configurer HSI/PLL/HSE (8MHz→168MHz Cortex-M4)

text

```
RCC->CR |= RCC_CR_HSION;             // HSI 16MHz ON
RCC->CFGR = RCC_CFGR_SW_PLL;         // Source PLL
RCC->PLLCFGR = 0x00020000;           // PLL 168MHz
```

Clock tree : AHB/APB1/APB2 diviseurs

6. Utilisation des Timers (TIM)

Objectif : PWM (moteurs/servos), décompte, capture

Les timers, horlogers du microcontrôleur

Les timers sont des compteurs matériels spécialisés qui mesurent le temps avec précision microseconde ou génèrent des signaux périodiques. Chaque STM32F407 embarque 14 timers différents, chacun capable de compter les impulsions d'horloge à des fréquences précises. Ils fonctionnent comme des horloges suisses intégrées qui continuent de tourner indépendamment du programme principal, parfait pour créer des délais, mesurer des durées ou piloter des moteurs.

Le mécanisme du compteur ascendant

Un timer est fondamentalement un compteur 16 ou 32 bits qui s'incrémente à chaque impulsion d'horloge. Un prédiviseur ralentit cette fréquence : par exemple, diviser 16MHz par 16000 donne 1kHz (1ms par tick). Quand le compteur atteint une valeur limite (ARR), il se remet à zéro et génère un signal d'overflow. Cette période programmable permet de créer des interruptions parfaitement régulières, comme un métronome électronique.

Génération PWM pour le contrôle analogique

Le mode PWM (Pulse Width Modulation) transforme le timer en générateur d'impulsions modulées dont la largeur varie proportionnellement à une consigne. Pour piloter un servomoteur, un timer génère une impulsion de 1ms à 2ms toutes les 20ms : 1ms=position gauche, 1.5ms=centre, 2ms=droite. Pour un moteur DC, le rapport cyclique (duty cycle) 0-100% contrôle la vitesse moyenne, transformant une sortie logique en signal analogique virtuel.

Capture de signaux externes

En mode capture, le timer fige sa valeur courante quand un signal externe (bouton, capteur) change d'état. Cette mesure précise la durée entre deux fronts montants : vitesse d'un encodeur rotatif, période d'un signal ultrason, ou délai entre deux impulsions laser. Avec deux canaux par timer, on mesure simultanément période ET rapport cyclique d'un signal unique, technique essentielle pour l'analyse spectrale temps réel.

Synchronisation maître-esclave

Les timers peuvent s'enchaîner : un timer maître génère des impulsions qui déclenchent des conversions ADC ou des transferts DMA dans d'autres périphériques. Dans un système d'acquisition ECG, un timer central pilote l'échantillonnage simultané de 8 canaux ADC à 1kHz précis, garantissant la cohérence temporelle des données pour l'analyse FFT. Cette architecture synchronisée est la clé des systèmes temps réel professionnels.

Applications multiples par timer

Un seul timer peut multitâcher : générer une PWM sur canal 1 (servomoteur), capturer un signal sur canal 2 (encodeur), et déclencher une interruption toutes les 10ms sur canal 3 (supervision). Le STM32F407 permet même d'enchaîner 4 timers pour créer un générateur de signaux arbitraires complexe, technique utilisée dans les générateurs de fonctions audio ou les contrôleurs moteur 3 phases.

Timers spécialisés pour missions critiques

Outre les timers généraux, le STM32F407 inclut des timers avancés (complémentaires pour moteurs brushless), des timers de base (délais simples), et le SysTick (calibrage système). Le timer de watchdog redémarre automatiquement le système si un programme plante. Dans les projets médicaux par exemple, ces timers garantissent une précision temporelle vitale : 1ms d'erreur peut changer un diagnostic d'épilepsie.

Les timers = cœur battant du système embarqué : ils donnent vie aux signaux et au temps réel !

text

```
TIM2->PSC = 15999;              // Prédiviseur 16MHz/16kHz
TIM2->ARR = 999;                // Période 1ms
TIM2->CR1 |= TIM_CR1_CEN;       // Compteur ON
```

Applications : PWM LED/moteur, timeout, heartbeat

7. Conversion Analogique-Numérique (ADC)

Objectif : Lecture capteurs (température, lumière, potentiomètre)

Le pont entre analogique et numérique

L'ADC transforme une tension continue (0-3.3V) en un nombre entier numérique que le microcontrôleur comprend. Un capteur de température produit une tension proportionnelle à 25°C ; l'ADC la convertit en valeur 2048 (sur 12 bits), que le processeur peut ensuite transformer en degrés Celsius. Cette conversion essentielle relie le monde physique analogique continu aux calculs numériques discrets du Cortex-M4.

Résolution et précision de mesure

La résolution définit la finesse de la mesure : un ADC 12 bits du STM32F407 divise 0-3.3V en 4096 niveaux (3.3V/4096 = 0.8mV par niveau). Un ADC 8 bits ne distingue que 256 niveaux (13mV), insuffisant pour mesurer des signaux ECG subtils. Plus la résolution est élevée, plus la précision est fine, mais le temps de conversion augmente proportionnellement.

Pipeline de conversion rapide

L'ADC fonctionne par échantillonnage puis quantification. Pendant un temps d'acquisition court (quelques µs), il prélève un instantané de la tension d'entrée et le maintient stable dans un condensateur interne. Ensuite, un convertisseur successif-approché (SAR) compare cette tension à des références internes par dichotomie binaire, déterminant chaque bit en 1-2 cycles. Un échantillon complet prend environ 12 cycles d'horloge (1µs à 16MHz).

Multiplexage intelligent des entrées

Le STM32F407 possède 16 voies ADC connectées à des broches spécifiques (PA0=ADC1_IN0, PA1=ADC1_IN1...). Un multiplexeur analogique sélectionne dynamiquement quelle broche mesurer : température en 5ms, puis humidité, puis lumière. En mode scan continu, l'ADC parcourt automatiquement toutes les voies assignées et stocke les résultats dans un buffer unique, libérant le CPU.

Synchronisation avec timers pour échantillonnage précis

Pour des mesures périodiques parfaites, l'ADC se synchronise sur un timer qui déclenche chaque conversion à intervalles réguliers. Dans un monitoring ECG à 1kHz, un timer génère une impulsion toutes les 1ms qui ordonne à l'ADC de capturer un échantillon. Cette précision temporelle uniforme est cruciale pour les analyses spectrales FFT ou la détection d'anomalies médicales.

Calibration automatique contre les dérives

Les imperfections hardware (offset, gain) s'accumulent avec la température et le vieillissement. L'ADC intègre une calibration automatique qui mesure ses propres erreurs internes et les compense par logiciel. Cette auto-ajustement garantit une précision stable sur 10 ans, vitale pour des équipements médicaux certifiés qui ne peuvent pas être recalibrés sur site.

L'ADC donne des yeux analogiques au microcontrôleur numérique : il voit le monde réel !

text

```
ADC1->SQR3 = 0;                    // Canal 0 (PA0)
ADC1->CR2 |= ADC_CR2_ADON;         // ADC ON
ADC1->CR2 |= ADC_CR2_SWSTART;      // Conversion
valeur = ADC1->DR;                 // 12 bits (0-4095)
```

Applications : Capteurs santé (EEG épilepsie), irrigation

8. Conversion Numérique-Analogique (DAC)

Objectif : Génération signaux analogiques (audio, servos)

Du nombre à la tension continue

Le DAC effectue l'opération inverse de l'ADC : il transforme un nombre numérique stocké en mémoire en une tension analogique continue proportionnelle. Le microcontrôleur écrit la valeur 2048 (moitié de 4096) et le DAC génère exactement 1.65V sur sa sortie (moitié de 3.3V). Cette conversion essentielle permet de piloter des composants analogiques comme des servomoteurs, des amplificateurs audio ou des circuits de commande précis.

Génération d'ondes arbitraires

Contrairement à la PWM qui approxime un signal analogique par modulation de largeur, le DAC produit un vrai signal analogique continu sans artefacts numériques. En écrivant séquentiellement les valeurs [0, 256, 512, ..., 4095, ...] à 10kHz, le DAC génère une onde sinusoïdale pure parfaite pour les tests audio, les générateurs de

fonctions ou les signaux de calibration médicaux. La fluidité du signal dépend uniquement de la fréquence de mise à jour.

Deux canaux indépendants

Le STM32F407 embarque deux canaux DAC physiquement séparés sur des broches dédiées (PA4= DAC_OUT1, PA5=DAC_OUT2). Chaque canal peut générer simultanément des signaux différents : un canal pilote un servomoteur d'irrigation tandis que l'autre génère un signal de test 1kHz pour vérifier le bon fonctionnement du système. Cette dualité double les possibilités sans consommer de timers supplémentaires.

Synchronisation avec timers et DMA

Pour des signaux périodiques impeccables, le DAC se synchronise sur un timer qui rafraîchit automatiquement la sortie à intervalles précis. Mieux encore, couplé à la DMA en mode circulaire, le DAC lit seul un tableau de 1024 valeurs sinusoïdales et les convertit en continu pendant que le CPU dort. Cette autonomie hardware produit des signaux parfaitement réguliers pendant des heures sans intervention logicielle.

Buffer interne pour fluidité maximale

Chaque canal DAC intègre un tampon de tenue qui maintient la tension stable entre deux écritures. Sans ce buffer, toute latence logicielle créerait des à-coups audibles dans l'audio ou des saccades dans les servomoteurs. Le buffer garantit une sortie parfaitement lisse même si le CPU est occupé par d'autres tâches critiques comme l'analyse temps réel d'ECG.

Applications précises dans l'industrie

Dans les projets médicaux par exemple, le DAC génère des signaux de calibration pour tester les capteurs ECG ou des tensions de référence stables pour les comparateurs de seuil. En agriculture intelligente, il pilote finement les vannes proportionnelles d'irrigation (0-100% ouverture) ou génère des signaux de diagnostic pour vérifier l'état des capteurs de sol. Dans les prototypes audio embarqués, il produit des sons de guidage ou d'alarme de qualité professionnelle.

Lien symétrique avec l'ADC

Le DAC et l'ADC forment un couple complémentaire parfait pour les boucles de régulation. L'ADC mesure la tension d'un capteur d'humidité (1.2V), le CPU calcule l'erreur par rapport à la consigne (1.5V), et le DAC génère la tension de commande de la vanne (2.1V). Cette chaîne analogique-numérique-analogique fermée maintient automatiquement l'humidité optimale avec une précision de 0.1%.

Le DAC redonne une voix analogique au microcontrôleur : des nombres deviennent des tensions utiles !

text

```
DAC->CR |= DAC_CR_EN1;              // DAC canal 1 ON
DAC->DHR12R1 = 2048;                // 1.65V (sur 3.3V)
```

Applications : Sortie audio buzzer, commande servo

9. Communication I²C

Objectif : Bus 2 fils (SDA/SCL) multi-esclaves

Le bus économique deux fils seulement

Le protocole I²C (Inter-Integrated Circuit) révolutionne la communication en utilisant uniquement deux fils pour connecter plusieurs appareils : SCL (horloge série) et SDA (données série). Contrairement à l'UART qui nécessite des connexions point à point, l'I²C crée un bus multi-maîtres multi-esclaves où un microcontrôleur peut interroger simultanément un accéléromètre, une EEPROM, un écran OLED et un capteur de température sur les mêmes deux broches.

Adresses uniques pour chaque esclave

Chaque appareil esclave possède une adresse I²C unique sur 7 ou 8 bits (0x68 pour MPU6050, 0x50 pour EEPROM 24Cxx). Le maître (STM32F407) commence chaque conversation en envoyant cette adresse sur le bus. Seul l'esclave correspondant répond en abaissant la ligne SDA (ACK). Cette sélection intelligente permet de partager les deux fils entre 128 appareils maximum sans conflit.

Séquence de dialogue maître-esclave

Une communication I²C débute par un start condition : SDA passe de 1 à 0 tandis que SCL reste à 1. Le maître envoie 7 bits d'adresse + 1 bit lecture/écriture, l'esclave répond ACK. Ensuite transitent les données octet par octet (8 bits + ACK), terminées par un stop condition (SDA de 0 à 1 pendant SCL=1). Chaque octet est synchronisé par 8 impulsions SCL générées par le maître.

Vitesse adaptable aux besoins

L'I²C supporte trois vitesses : standard 100kHz (fiable), fast 400kHz (courant), et high-speed 3.4MHz (rare). À 400kHz, un octet transite en 25µs. Le STM32F407 ajuste dynamiquement cette fréquence via un registre de contrôle d'horloge, s'adaptant automatiquement à la capacité du bus et aux longueurs de câbles. Les résistances de pull-up 4.7kΩ sur SCL/SDA garantissent l'état haut par défaut.

Lecture et écriture directionnelles

En mode écriture, le maître configure un registre distant (adresse 0x37 octet 0x47). En mode lecture, l'esclave pousse ses données vers le maître (mesure capteur). Les communications longues utilisent un repeated start : le maître recommence sans stop, changeant de direction lecture→écriture pour d'abord configurer puis lire une mesure. Cette technique optimise les échanges avec les capteurs intelligents.

Détection robuste des erreurs

L'esclave confirme chaque octet reçu par un ACK (SDA=0). Un NACK (SDA=1) signale une erreur ou la fin de transfert. Le maître détecte les collisions (deux maîtres simultanés) et arbitre automatiquement la priorité. La synchronisation SCL/SDA

empêche les faux départs. Ces mécanismes rendent l'I²C remarquablement robuste face aux bruits électriques industriels.

L'I²C = économie maximale de câbles pour richesse maximale de capteurs !

text

```
I2C1->CR1 |= I2C_CR1_PE;            // I2C ON
// Écriture EEPROM 0x50
I2C1->CR2 |= (0x50<<1);             // Adresse esclave
```

Applications : EEPROM 24Cxx, accéléromètres, OLED

10. Communication SPI

Objectif : Bus synchrone rapide (SD card, écrans TFT)

Le bus synchrone ultra-rapide

Le SPI (Serial Peripheral Interface) est un protocole synchrone qui excelle dans les transferts de données massives à très haute vitesse. Contrairement à l'I²C limité à 400kHz, le SPI atteint facilement 10 à 50 MHz, parfait pour les cartes SD, écrans TFT, ou capteurs haute définition. Il utilise quatre fils : SCK (horloge), MOSI (maître vers esclave), MISO (esclave vers maître), et NSS (sélection esclave).

Sélection individuelle des esclaves

Chaque appareil esclave possède sa propre ligne NSS (chip select) active bas. Seul l'esclave dont NSS est mis à 0 répond pendant la transaction. Un STM32F407 peut ainsi communiquer simultanément avec une carte SD (NSS=PA4), un écran LCD (NSS=PA3), et un codec audio (NSS=PB0) sur les mêmes lignes SCK/MOSI/MISO. Cette sélection individuelle garantit l'absence de collisions.

Transfert bidirectionnel synchrone

À chaque impulsion SCK, un bit circule dans chaque sens simultanément : le maître envoie une donnée sur MOSI tandis que l'esclave renvoie une réponse sur MISO. Cette duplexité intégrale double le débit effectif. Le maître contrôle totalement le rythme via sa fréquence SCK, l'esclave n'ayant qu'à suivre. Un octet complet (8 bits) transite donc en 200ns à 50MHz.

Modes de synchronisation cruciaux

Le SPI définit quatre modes selon le niveau initial de SCK (0 ou 1) et son moment de changement (front montant ou descendant). Les capteurs comme l'ADXL345 préfèrent généralement le mode 0 (SCK bas initial, échantillonnage front montant). Une incompatibilité de mode provoque des données corrompues. Le STM32F407 configure ces polarité/phase par deux bits dans son registre de contrôle.

Polarité maître/esclave stricte

Le STM32F407 en mode maître génère SCK et initie toutes les transactions. Les esclaves (carte SD, LCD) attendent passivement les impulsions. En mode esclave, le STM32F407 synchronise ses données sur SCK externe, utile pour recevoir des

commandes d'un maître principal. La gestion NSS manuelle permet des protocoles hybrides maître/esclave complexes dans les architectures distribuées.

Transferts DMA pour performances maximales

Couplé à la DMA, le SPI transfère des blocs entiers (pages EEPROM 256 octets, images LCD 320x240) sans intervention CPU. À 20MHz avec DMA, une photo 100Ko s'écrit sur carte SD en 50ms. Cette autonomie hardware libère le Cortex-M4 pour l'analyse temps réel pendant que le SPI remplit automatiquement les buffers d'acquisition ou d'affichage.

SPI = autoroute 10x plus rapide que l'I²C pour les gros transferts !

text
```
SPI1->CR1 |= SPI_CR1_MSTR | SPI_CR1_SSI | SPI_CR1_SSM;
SPI1->CR1 |= SPI_CR1_SPE;           // SPI ON
```
Applications : Cartes SD (data logging), LCD Nokia 5110

11. Calcul des CRC (Cyclic Redundancy Check)

Objectif : Détection erreurs trames UART/SPI/I²C

text
```
CRC->CR |= CRC_CR_RESET;            // Reset CRC
CRC->DR = 0x12345678;               // Donnée 32bits
resultat = CRC->DR;                 // CRC calculé
```
Applications : Protocoles robustes, Modbus RTU

12. Utilisation du Watchdog Timer

Objectif : Reset automatique si programme bloqué

text
```
IWDG->KR = IWDG_KEY_RELOAD;         // Reload watchdog (toutes les 30s)
```
Applications : Sécurité systèmes embarqués critiques

13. L'horloge temps réel (RTC)

Objectif : Horodatage, alarmes, calendriers

text
```
RTC->ISR |= RTC_ISR_INIT;           // Mode init
RTC->TR = 0x20324548;               // 20:32:45
RTC->CR |= RTC_CR_TSE;              // Timestamp ON
```
Applications : Logs horodatés, alarmes irrigation