Ecole Nationale
Supérieure
de l'Electronique
ENSEA et de ses Applications

Noyaux Temps-réel

Modèles de drivers

Laurent Fiack
Bureau D212/D060 – laurent.fiack@ensea.fr

Les polled-based drivers

Exemple : Un polled-based driver

```c
char polled_uart_receive()
{
    while(!(USART2->ISR & USART_ISR_RXNE_Msk));
    return USART2->RDR;
}

void t_uart_print_out(void* unused)
{
    char chr;

    STM_uart_init(USART2, 9600, NULL, NULL);

    for(;;)
    {
        chr = polled_uart_receive();
        printf("%c", chr);
    }
}
```

- Avantages
  - Facile à programmer
  - Accès immédiat
- Problèmes
  - Doit être une des tâches les plus prioritaires
  - Risque de rater des données
  - Forte utilisation du CPU
- Utilisation
  - Vérification/Initialisation du système
  - Cas particulier :
    - ex. Besoin de réactivité (rare)

Exemple : Ajout d'une tâche et d'une queue

```
static QueueHandle_t q_uart2_received = NULL;

void t_polled_uart_receive(void* unused)
{
    char chr;

    STM_uart_init(USART2, 9600, NULL, NULL);
    q_uart2_received = xQueueCreate(Q_LENGTH, Q_SIZE);

    for(;;)
    {
        while(!(USART2->ISR & USART_ISR_RXNE_Msk));
        chr = USART2->RDR;
        xQueueSend(q_uart2_received, (void*)(&chr), 0);
    }
}

void t_uart_print_out(void* unused)
    char chr;
    for(;;)
    {
        xQueueReceive(q_uart2_received, (void*)(&chr), portMAX_DELAY);
        printf("%c", chr);
    }
}
```

- Utilisation d'une queue
  - Peut stocker plusieurs caractères avant de les traiter
- Problèmes
  - Toujours faire très attention aux priorités
  - Compromis compliqué à justifier

Exemple : Encapsulation de la queue

```c
static QueueHandle_t q_uart2_received = NULL;

void t_polled_uart_receive(void* unused)
{
    char chr;

    STM_uart_init(USART2, 9600, NULL, NULL);
    q_uart2_received = xQueueCreate(Q_LENGTH, Q_SIZE);

    for(;;)
    {
        while(!(USART2->ISR & USART_ISR_RXNE_Msk));
        chr = USART2->RDR;
        xQueueSend(q_uart2_received, (void*)(&chr), 0);
    }
}

char polled_uart_receive()
{
    char chr;
    xQueueReceive(q_uart2_received, (void*)(&chr), portMAX_DELAY);
    return chr;
}
```

```c
void t_uart_print_out(void* unused)
{
    char chr;
    for(;;)
    {
        chr = polled_uart_receive();
        printf("%c", chr);
    }
}
```

- Encapsulation d'une queue
  - Rend l'utilisation du driver plus lisible
  - Attention au blocage
  - Bien documenter les fonctions
  - Timeout réglable
    - Paramètre ou define

Les ISR-based drivers

Les queue-based drivers

Exemple : Un queue-based driver

```c
static QueueHandle_t q_uart2_received = NULL;
static uint8_t rx_in_progress = 0;

void uart_start_receive(void)
{
    q_uart2_received = xQueueCreate(Q_LENGTH, Q_SIZE);
    rx_in_progress = 1;

    USART2->CR3 |= USART_CR3_EIE;
    USART2->CR1 |= (USART_CR1_UE | USART_CR1_RXNEIE);
    NVIC_SetPriority(USART2_IRQn, 6);
    NVIC_EnableIRQ(USART2_IRQn);
}

void t_uart_print_out(void* unused) {
    char chr;

    STM_UartInit(USART2, 9600, NULL, NULL);

    uart_start_receive();

    for(;;)
    {
        chr = uart_receive();
        printf("%c", chr);
    }
}
```

```c
char uart_receive()
{
    char chr;
    xQueueReceive(q_uart2_received, (void*)(&chr), portMAX_DELAY);
    return chr;
}

void USART2_IRQHandler(void)
{
    portBASE_TYPE hptw = pdFALSE;

    //error flag clearing omitted for brevity
    if(USART2->ISR & USART_ISR_RXNE_Msk)
    {
        char chr = (uint8_t) USART2->RDR;

        if(rx_in_progress)
        {
            xQueueSendFromISR(q_uart2_received, &chr, &hptw);
        }

        portYIELD_FROM_ISR(hptw);
    }
}
```

Les queue-based driver

- Utilisation d'interruption
  - Réduction de la charge du CPU
- Utilisation d'une queue entre l'interruption et l'application
  - Peut exécuter d'autres tâches pendant la réception
  - Grâce à la queue, les données sont stockées dans une FIFO
- Points de vigilence
  - Attention au remplissage de la queue.
    - Pas possible de bloquer depuis une interruption
    - Ajouter de la gestion d'erreur !
  - Attention au débit
    - Lecture/Écriture dans une queue est relativement lent
    - Attention s'il y a beaucoup d'interruption par secondes

Les ISR-based drivers

Les buffer-based drivers

Exemple : Un buffer-based driver (1/2)

```c
static uint8_t rx_in_progress = 0;
static uint16_t rx_len = 0;
static uint8_t * rx_buffer = NULL;
static uint16_t rxItr = 0;

int uart_start_receive(uint8_t * buffer, uint16_t len)
{
    if ((!rx_in_progress) && (buffer != NULL))
    {
        rx_in_progress = 1;
        rx_len = len;
        rx_buffer = buffer;
        rx_itr = 0;
        USART2->CR3 |= USART_CR3_EIE;
        USART2->CR1 |= (USART_CR1_UE | USART_CR1_RXNEIE);
        NVIC_SetPriority(USART2_IRQn, 6);
        NVIC_EnableIRQ(USART2_IRQn);
        return 0;
    }
    return -1;
}
```

```c
void USART2_IRQHandler(void)
{
    BaseType_t hptw = pdFALSE;
    if (CHECK_ERROR_IRQ_MASKS())
    {
        RESET_ERROR_IRQ_MASKS();
        if (rx_in_progress)
        {
            rx_in_progress = 0;
            xSemaphoreGiveFromISR(sem_rx_done, &hptx);
        }
    }
    if (CHECK_RXNE_IRQ_MASK())
    {
        uint8_t chr = (uint8_t) USART2->RDR;
        if (rx_in_progress)
        {
            rx_buffer[rx_itr++] = chr;

            if (rx_itr >= rx_len)
            {
                rx_in_progress = 0;
                xSemaphoreGiveFromISR(sem_rx_done, &hptw);
            }
        }
    }
    portYIELD_FROM_ISR(hptw);
}
```

Exemple : Un buffer-based driver (2/2)

```
void t_uart_print_out (void * unused)
{
    uint8_t rx_data[20];
    uint8_t expected_length = 16;

    STM_UartInit(USART2, 9600, NULL, NULL);

    for(;;)
    {
        uart_start_receive(rx_data, expected_length);
        if (xSemaphoreTake(sem_rx_done, 100) == pdPASS)
        {
            if (expected_length == rx_itr)
            {
                printf("Received: %s\r\n", rx_data);
            }
            else
            {
                printf("Expected %d bytes, received %d\r\n", expected_length, rx_itr);
            }
        }
        else
        {
            printf("Timeout error\r\n");
        }
    }
}
```

Les buffer-based drivers

- Utile si la taille du transfert est connue à l'avance
  - Limité aux cas où la longueur de transmission est connue.
- Moins de cycles dans l'ISR
  - Un seul sémaphore pour toute une *transaction*
- Norme dans les applications bare-metal
  - Facilement adaptable avec FreeRTOS
- Limitation
  - Pas de stockage en FIFO, besoin de réactivité de la tâche

Les DMA-based drivers

Rappels DMA : Exemple STM32G070RBT6

Figure 1. System architecture

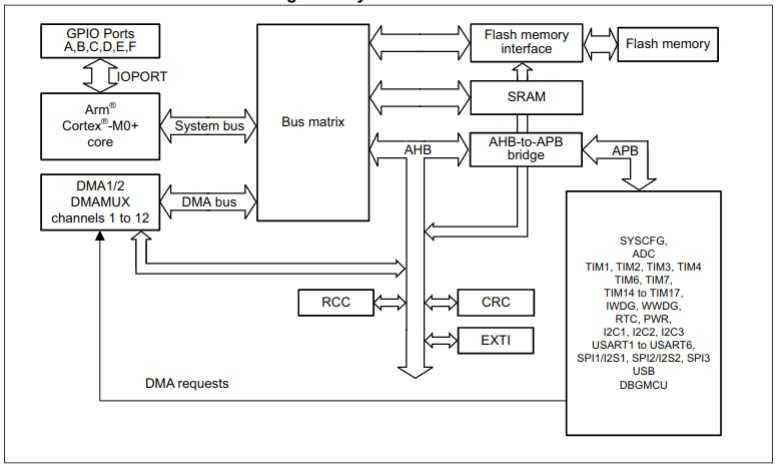
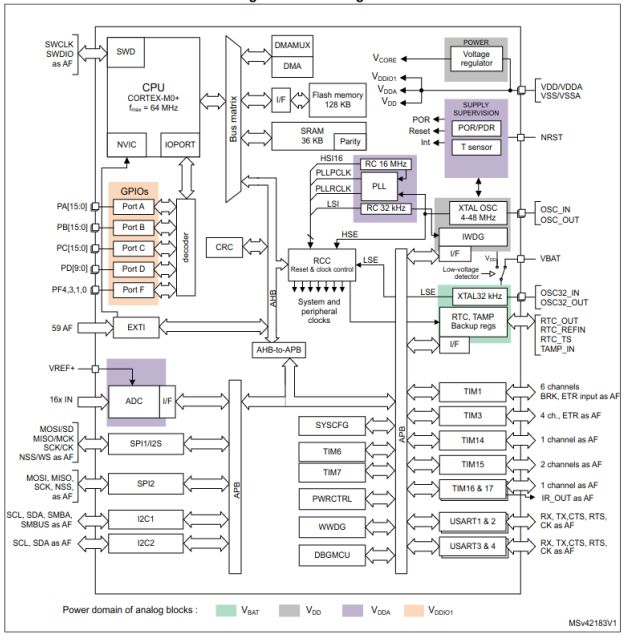


Figure 1. Block diagram



Exemple : Un DMA-based driver

```
void setupUSART2DMA(void)
{
    __HAL_RCC_DMA1_CLK_ENABLE();
    NVIC_SetPriority(DMA1_Stream5_IRQn, 6);
    NVIC_EnableIRQ(DMA1_Stream5_IRQn);

    usart2DmaRx = DMA_CONFIG_CHANNEL(DMA_CHANNEL_4);

    HAL_StatusTypeDef retVal = HAL_DMA_Init(&usart2DmaRx);
    assert_param(retVal == HAL_OK);

    //enable transfer complete interrupts
    DMA1_Stream5->CR |= DMA_SxCR_TCIE;

    //set the DMA receive mode flag in the USART
    USART2->CR3 |= USART_CR3_DMAR_Msk;
}
```

```
void DMA1_Stream5_IRQHandler(void) {
    BaseType_t hptw = pdFALSE;

    if(rxInProgress && (DMA1->HISR & DMA_HISR_TCIF5)) {
        rxInProgress = false;
        DMA1->HIFCR |= DMA_HIFCR_CTCIF5;
        xSemaphoreGiveFromISR(rxDone, &hptw);
    }

    portYIELD_FROM_ISR(hptw);
}
```

Les DMA-based drivers

- Très proche du buffer-based driver
- Avantages
  - Plus rapide
  - Un peu plus complexe
  - Très utile pour des transferts longs
  - Ou très rapides
- Limites
  - Un peu plus long à coder
  - Rend le code *légèrement* moins lisible
  - DMA non-illimités

Choix d'un modèle de driver

Choix d'un modèle de driver

- Dépend de différents facteurs
  - Comment fonctionne le code de l'application ?
  - Quelle latence est acceptable ?
  - Quel est le débit de données ?
  - Quel type de périphérique ?

Comment fonctionne le code de l'application ?

- Est-ce qu'il doit traiter les octets/charactères dès qu'ils arrivent ?
- Est-ce qu'il faut attendre un certain nombre de données pour effectuer des traitements de haut niveau ?

- Queue-based drivers pour traiter des quantité indéfinies de données, ou des streams
- Buffer-based drivers pour transférer des structures de données

Quel est le débit des données

- Les queues sont pratiques, mais son utilisation peut être assez lourde.
  - Exemple :
    - À 9600 baud chaque caractère doit être traité en 40µs.
    - À 115200 baud, il n'y a plus que 9µs
    - L'écriture dans une Queue est trop lente.
- Un driver avec un DMA double-buffered peut être une bonne alternative.
- Plus compliqué (= moins lisible/réutilisable)

Quel est le type de périphérique

- Les queues sont bien adaptées aux périphériques asynchrones
  - UARTs,
  - USB,
  - Réseau,
  - Timer capture...
- Les block-based drivers sont bien adaptés aux périphériques synchrones
  - SPI,
  - I2C...

En résumé

Quand utiliser une queue

- Périphérique/application reçoivent des données d'une taille inconnue
- Données doivent être reçues de manière asynchrone
- Driver doit recevoir des données de plusieurs sources
- Débit suffisamment lent (1∼10µs entre les interruptions)

Quand utiliser un buffer

- Besoin de gros buffers, grandes quantités de données d'un coup
- Protocoles de communication transaction-based
- Quand les longueurs sont connues à l'avance

Organisation du code et bonnes pratiques

TD : Initialisation de l'UART avec la HAL

- Ouvrez le projet du TD Shell
- Activez les interruptions de l'USART1
- Dans le fichier `main.c`, repérez la fonction `MX_USART1_UART_Init()`
- Allez voir le code de cette fonction
- Quel est le paramètre passé à la fonction `HAL_UART_Init()` ? Pourquoi utiliser un pointeur ?
- Quels sont les membres de la structure ?
- Allez voir le code de la fonction `HAL_UART_Init()`
- Où est définie `HAL_UART_MspInit()` ?
- Que fait la fonction `UART_SetConfig()` ?
- Quel est le rôle de gState ? Quel est sa valeur au retour de la fonction `HAL_UART_Init()` ?

TD : Transmission d'une chaine sans interruption

- Que se passe-t-il si la HAL n'est pas dans l'état `HAL_UART_STATE_READY` ?
- Peut-on utiliser un autre UART pendant ce temps-là ?
- Expliquez le rôle de chacun des paramètres suivants :
  - pData
  - Size
  - Timeout
- Peut-on appeler la fonction `HAL_UART_Transmit()` dans deux fonctions différentes en même temps ?
- Est-ce que le code est réentrant ?

TD : Réception d'une chaine avec interruptions

- Avant de regarder le code, devinez le fonctionnement de la fonction `HAL_UART_Receive_IT`
- Listez les fonctions utilisée par la réception
- Où et par qui est appelée la fonction `UART_RxISR_8BIT()` ?
- Quand la fonction `HAL_UART_RxCpltCallback()` est-elle appelée ?
- Après analyse, vérifiez si votre supposition était correcte

Qu'est-ce qu'on peut en tirer ?

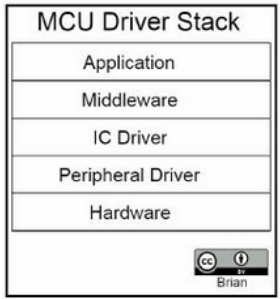
- Toutes les variables de travail sont stoquées dans une structure
  - Les fonctions sont génériques = les mêmes quel que soit le périphérique choisi
- On travaille avec des pointeurs, les buffers appartiennent à l'utilisateur
- Il y a des pointeurs de fonctions
- Il y a des structures dans des structures
- Utilisation d'une machine à état pour les interruptions

Différents niveaux de drivers

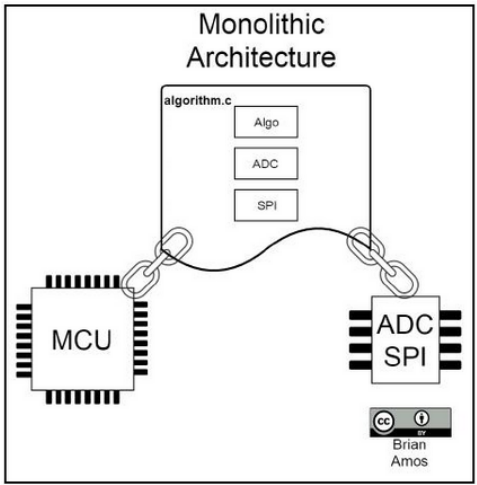
Exemple : l'ADXL345 est un accéléromètre SPI

- Il faut des fonctions d'accès au bus SPI
- Et des fonctions spécifiques à l'ADXL345

- Modèle en couches
- Peripheral Driver accède directement au matériel du MCU (eg. SPI)
- IC driver est juste au dessus.
  - Se base sur le Peripheral driver.
  - Doit fonctionner indépendamment du hardware du MCU.

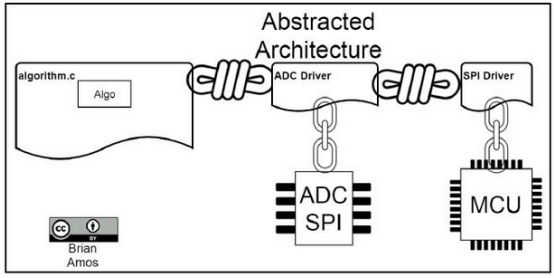


Il faut éviter des architectures monolithiques



- Ici, l'accès au registre se fait directement dans l'application
- Le code est lié au MCU et à l'IC
- Code difficile à maintenir
- MCU et IC difficile à remplacer

Et préférer des architectures avec plus d'abstraction



- SPI driver est fortement lié au MCU
- IC driver dépend de l'IC, mais peut s'adapter sur d'autres MCU
- L'algo est indépendant du matériel

Guide des bonnes pratiques

- Paire de fichier .c/.h par driver
  - On peut séparer en plus pour des très gros drivers
- Identifiez les fonctions internes et les fonctions accessibles à l'utilisateur
  - Les fonctions internes sont en static
- Utilisez des structures pour stocker toutes les variables de travail
  - Le code doit être réentrant
  - Où planter proprement
- Si possible, écrivez le code d'interruption dans des callbacks, dans le main
- De préférence, une seule fonction appelée dans l'interruption
  - Fonctions bien identifiées (et documentées) dans le .h
  - Éventuellement, ça peut être des fonctions satic inline dans le .h
- Dans la structure du driver IC, utilisez des pointeurs de fonction pour passer les fonctions du peripheral driver