POLYTECH
NICE-SOPHIA

Systèmes à Microprocesseurs

Cycle Ingénieur Troisième Année

Sébastien Bilavarn

Plan

- Ch1 – Représentation de l'information
- Ch2 – ARM Instruction Set Architecture
- Ch3 – Accès aux données
- Ch4 – Programmation structurée
- Ch5 – Cycle d'exécution
- Ch6 – Codage binaire
- Ch7 – Microcontrôleur ARM Cortex-M

POLYTECH
NICE-SOPHIA

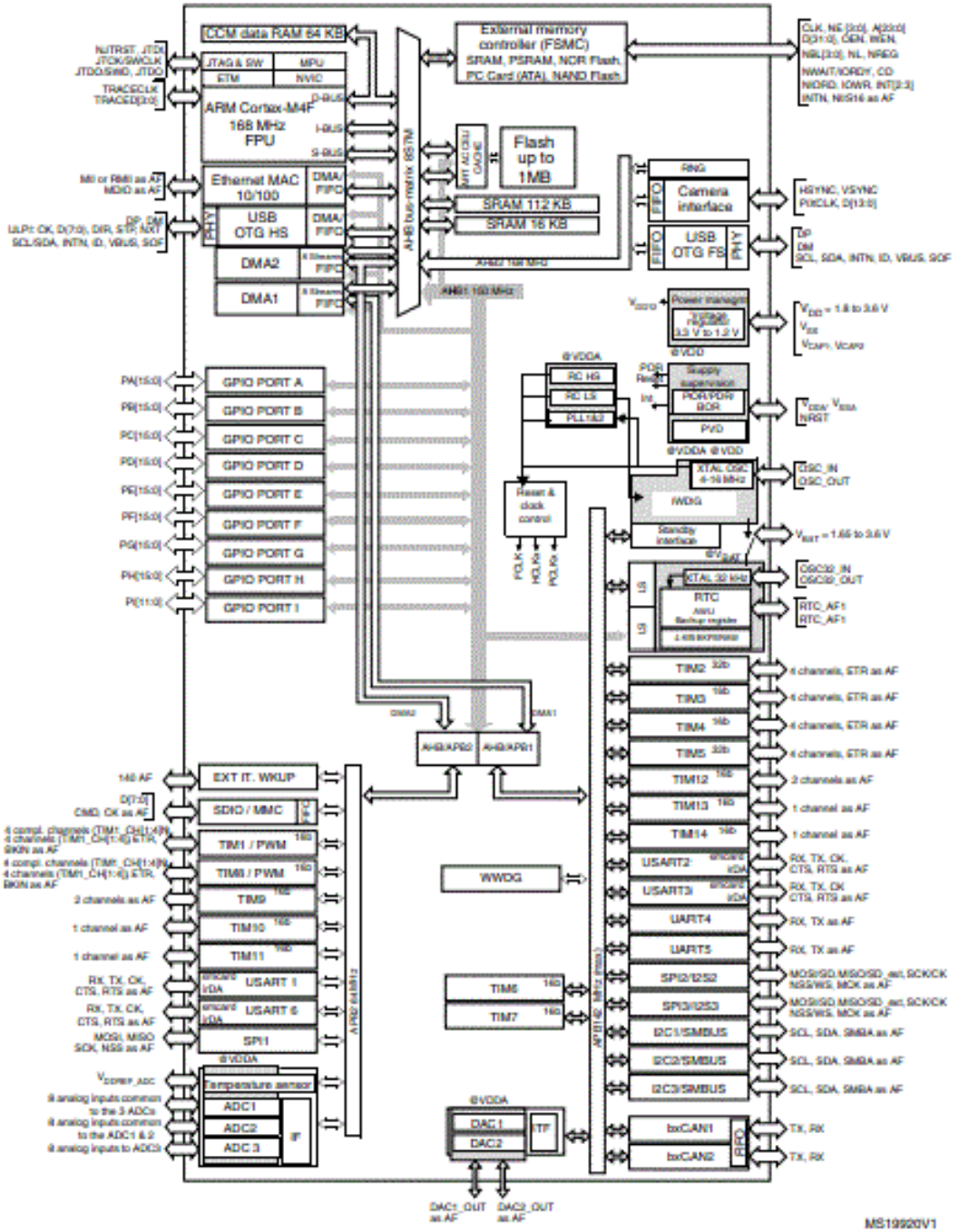
Microcontrôleur ARM Cortex-M

- Microcontrôleur
- Les périphériques
- Interruptions
- Les interruptions du processeur ARM

Qu'est-ce qu'un microcontrôleur ?

- Un microcontrôleur est un circuit intégré qui rassemble dans un même boitier
  - Un microprocesseur
  - Des mémoires
  - Des périphériques
- Les périphériques sont des circuits capables d'effectuer des tâches spécifiques
  - Conversion A/N, N/A
  - Génération de signaux (PWM)
  - Compteur de temps ou d'évènements (Timer)
  - Communications, contrôleurs de bus (UART, IIC)

STM32F407VGT6



- Horloges périphériques
  - APB1
  - APB2
  - AHB1
  - AHB2

- CMSIS
  - Cortex Microcontroller Software Interface Standard
    - http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php
  - API pour la famille de coeurs Cortex-M et leurs périphériques.
- Adresse de base des périphériques
  - `stm32f4xx.h`
  - `#define PERIPH_BASE ((uint32_t)0x40000000)`
- Horloges
  - `#define APB1PERIPH_BASE PERIPH_BASE`
  - `#define APB2PERIPH_BASE (PERIPH_BASE + 0x00010000)`
  - `#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000)`
  - `#define AHB2PERIPH_BASE (PERIPH_BASE + 0x10000000)`

POLYTECH
NICE-SOPHIA

Microcontrôleur ARM Cortex-M

- Microcontrôleur
- Les périphériques
- Interruptions
- Les interruptions du processeur ARM

Les périphériques

- GPIO: General Purpose Input/Outpout
- UART: Universal Asynchronous Receiver Transmitter
- I^2C: Inter Integrated Circuit
- SPI: Serial Peripheral Interface
- Timer: minuteur
- PWM: Pulse-width modulation

GPIO

- Entrées/sorties pour usage général
- Comporte un ensemble de ports d'entrée/sortie qui peuvent être configurées pour jouer soit le rôle d'une entrée, soit le rôle d'une sortie.
  - Lorsqu'un port GPIO est configuré en tant que sortie, on peut écrire dans un registre interne afin de modifier l'état d'une sortie.
  - Lorsqu'il est configuré en tant qu'entrée, on peut détecter son état en lisant le contenu d'un registre interne.
- Exemple: un GPIO est fréquemment utilisé pour contrôler des LEDs.

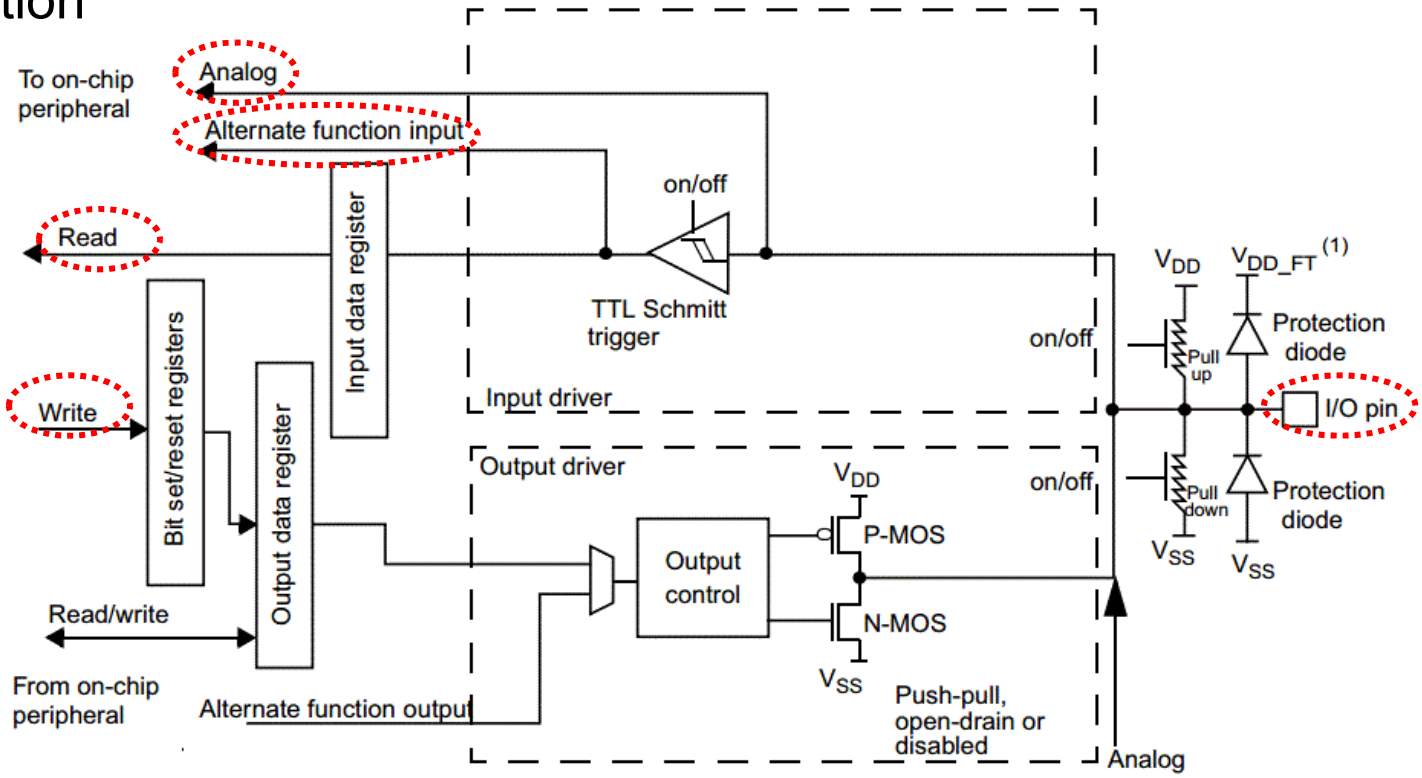
Registres GPIO STM32

- stm32f4xx.h

```
#define PERIPH_BASE           ((uint32_t)0x40000000)
#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000)
#define GPIOH_BASE            (AHB1PERIPH_BASE + 0x1C00)
typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint16_t BSRRL;    /*!< GPIO port bit set/reset low register,  Address offset: 0x18      */
  __IO uint16_t BSRRH;    /*!< GPIO port bit set/reset high register, Address offset: 0x1A      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

Exemple

- Port mode register (MODER)
  - Input
  - General purpose output
  - Alternate function
  - Analog



- Exemple
  - `GPIOSTM32F4xx.h`
  - Configurer le port H.3 en sortie
    - `GPIOH_MODER = 0x40  // @40021C00 <- 0x40`

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIO STM32

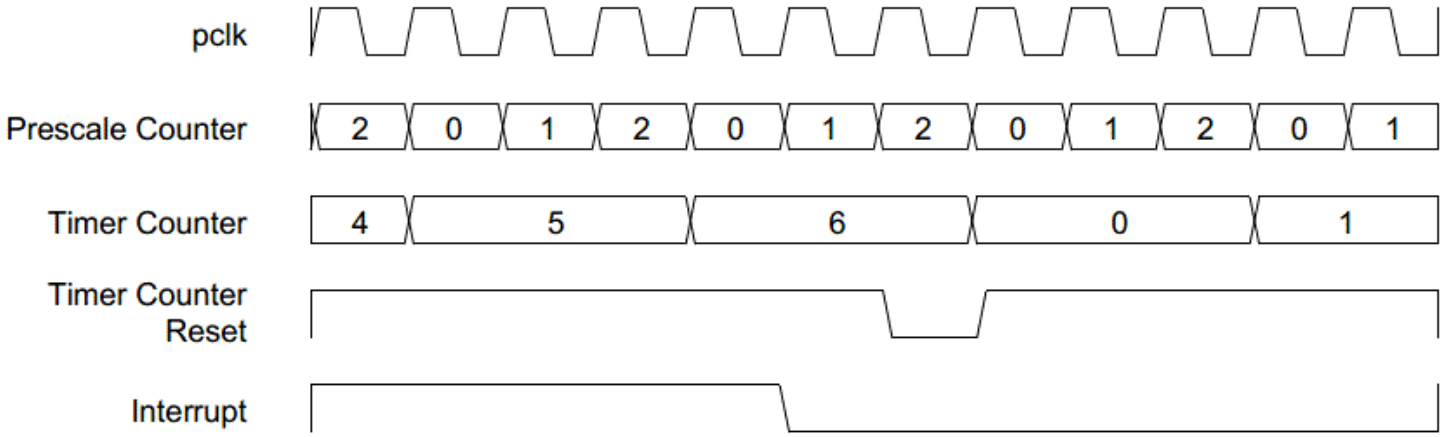
- Exemple
  - Configurer le port H.3 en sortie, push-pull, 50MHz, Pull-down

```
GPIOH->MODER = 0x40;        // @40021C00 <- 0x40
GPIOH->OTYPER = 0x0;        // @40021C04 <- 0x0
GPIOH->OSPEEDR = 0x80;      // @40021C08 <- 0x80
GPIOH->PUPDR = 0x80;        // @40021C0C <- 0x80
```

  - En utilisant les fonctions CMSIS

```
GPIO_PinConfigure(  GPIOH,
                    3,
                    GPIO_MODE_OUTPUT,
                    GPIO_OUTPUT_PUSH_PULL,
                    GPIO_OUTPUT_SPEED_50MHz,
                    GPIO_PULL_DOWN);
```

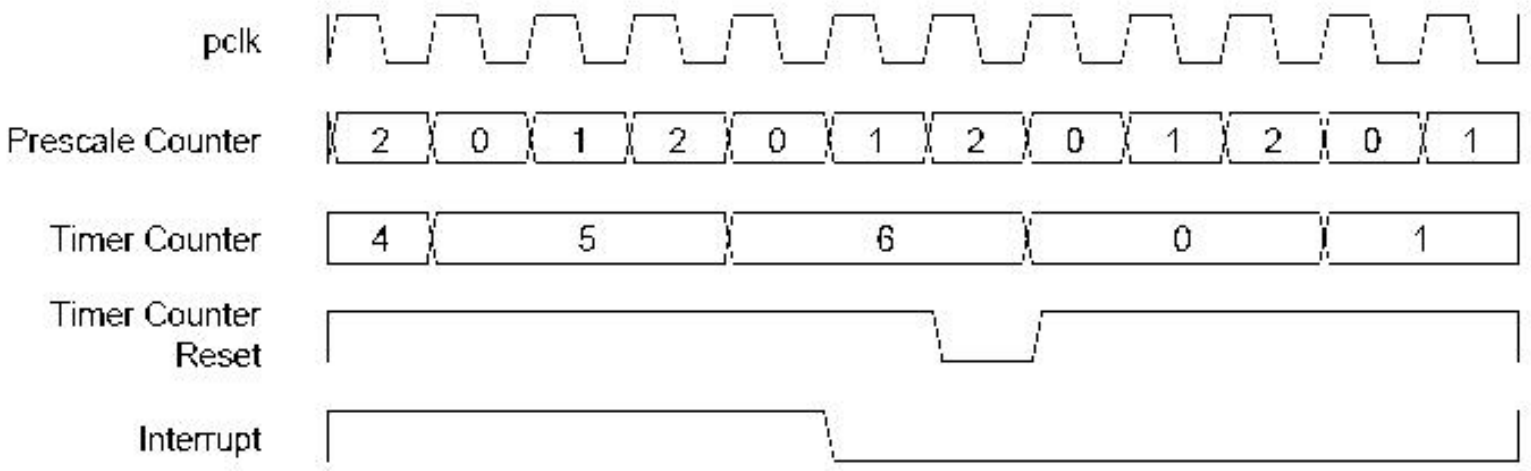
Toutes les macros sont définies dans `GPIO_STM32F4xx.h`



Timer

- Périphérique matériel permettant de mesurer les durées
- Temporisateur programmable: compteur que l'on peut programmer
  - Exemple: pour compter périodiquement de 0 à 10000 pour réaliser une temporisation
- Il permet
  - De mesurer des intervalles de temps
  - De réaliser des temporisations
  - De générer des signaux PWM

- Prescaler (prédiviseur)



  - Prescaler = 2: le compteur (Timer Counter) s'incrémente tous les trois cycles d'horloge (pclk)
  - Period = 6: le compteur repart à 0 (reset) et génère une interruption lorsqu'il atteint la valeur 6

Timer STM32

- Le timer fonctionne à 84MHz

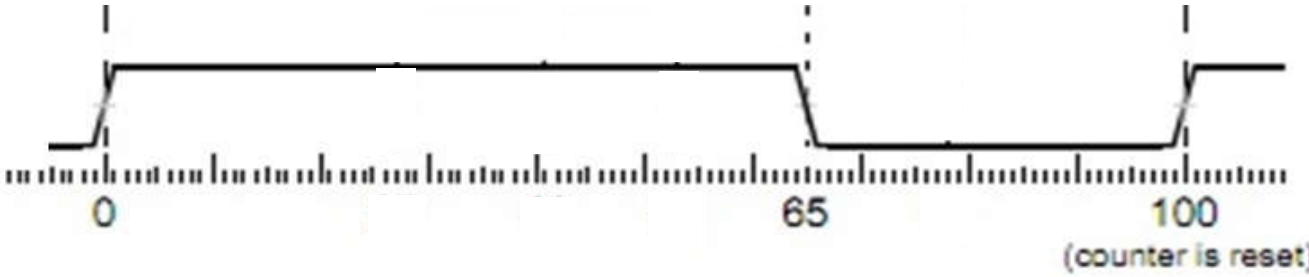
- Exemple
  - Configurer le timer 5 avec : prescaler=0, mode comptage croissant, période 10KHz, division d'horloge=1

```
TIM_TimeBaseInitTypeDef TIM_BaseStruct;

TIM_BaseStruct.TIM_Prescaler = 0;
TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_BaseStruct.TIM_Period = 8399; /* 10kHz PWM */
TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInit(TIM5, &TIM_BaseStruct);
```

Toutes les macros sont définies dans `stm32f4xx_tim.h`

PWM

- Pulse Width Modulation (Modulation de largeur d'Impulsion)
  - Le principe est de générer un signal logique (valant 0 ou 1), à fréquence fixe mais dont le rapport cyclique est contrôlé numériquement.



  - L'exemple ci-dessus définit un signal de période 100.
  - Le timer fonctionne à 84MHz, la fréquence du signal PWM est donc 84 10^6/100 = 840 KHz.
  - Il est programmé pour passer de 1 à 0 sur la valeur 65, le rapport cyclique (pulse) est donc de 65%.

UART

- L'UART est un émetteur-récepteur asynchrone universel.
- Composant utilisé pour faire le lien entre un microprocesseur et le port série.
- Tous les bits de données transitent en série (à travers un même fil)
- Vitesse de transmission:
  - Bauds (1 bd = 1 temps bit, 9600 bauds = 9600 bps)
  - Vitesses normalisées: 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1843200, 3686400
  - Relativement lent

UART

- Composition d'une trame:

  - 1 bit de *start* toujours à 0, pour la synchronisation du récepteur
  - Les données (la taille peut varier entre 5 et 8 bits)
  - Éventuellement un bit de parité, paire ou impaire (détection d'erreur de transmission)
  - 1 bit de *stop* toujours à 1, la durée peut varier entre 1, 1.5 et 2 temps bit
  - Le niveau logique de repos est le 1.

UART STM32

- Exemple
  - Configurer l'USART avec les paramètres de transmission: 9600 bauds, pas contrôle de flux, mode émission/réception, pas de parité, 1 bit de stop, taille des données transmises 8 bit

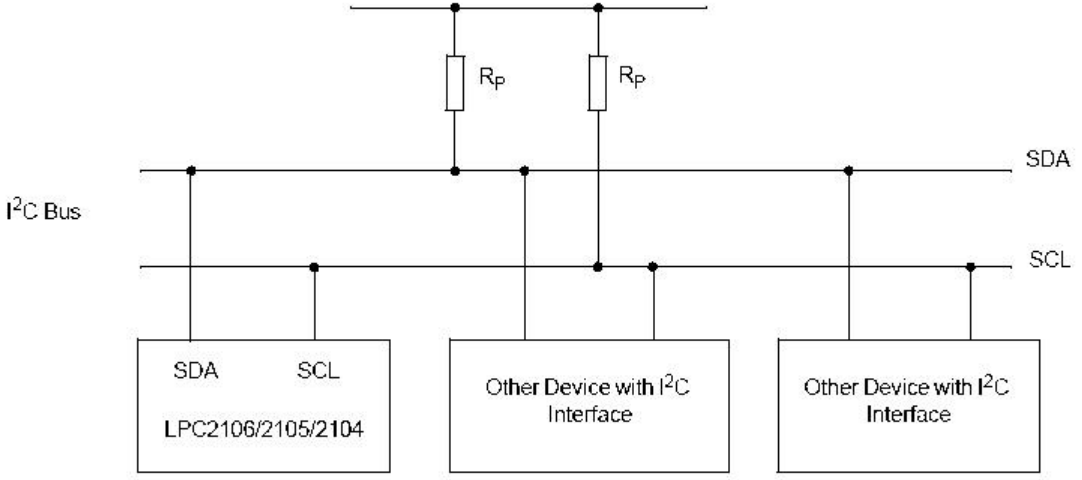
```
USART_InitTypeDef USART_InitStruct;


USART_InitStruct.USART_BaudRate = 9600;
USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Tx |
USART_Mode_Rx;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_WordLength =
USART_WordLength_8b;
USART_Init(USART1, &USART_InitStruct);
```

  Toutes les macros sont définies dans `stm32f4xx_usart.h`

I²C (1)

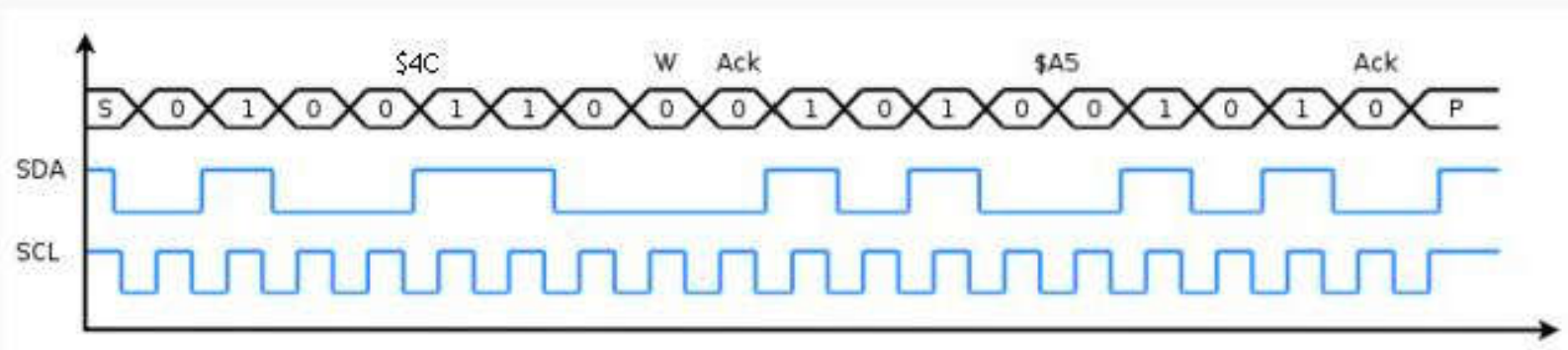
- Développé par Philips pour des applications de domotique
- bus série et synchrone composé de trois fils:
  - un signal de donnée (SDA)
  - un signal d'horloge (SCL)
  - un signal de référence (masse)

Plusieurs composants peuvent être branchés sur le même bus I2C. Pour que l'information aille au bon endroit chaque composant possède sa propre adresse. Elle est composée d'une partie fixe imposée par le constructeur, d'une partie configurable de façon matérielle par l'utilisateur, et du bit de read/write qui définit le sens de la transmission (0 pour écriture, 1 pour lecture).



I²C (2)

- Exemple

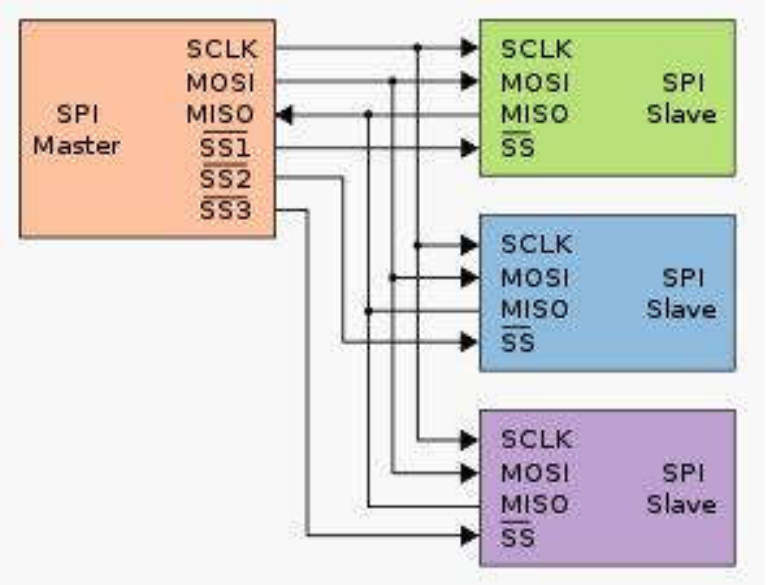


  - La communication commence par le StartBit
  - Puis on envoie
    - l'adresse (4C, sur 8 bits)
    - L'acknowledge (Ack)
    - Un octet de donnée (A5)
    - De nouveau un acknowledge (Ack)
    - Pour finir, le StopBit

SPI (1)

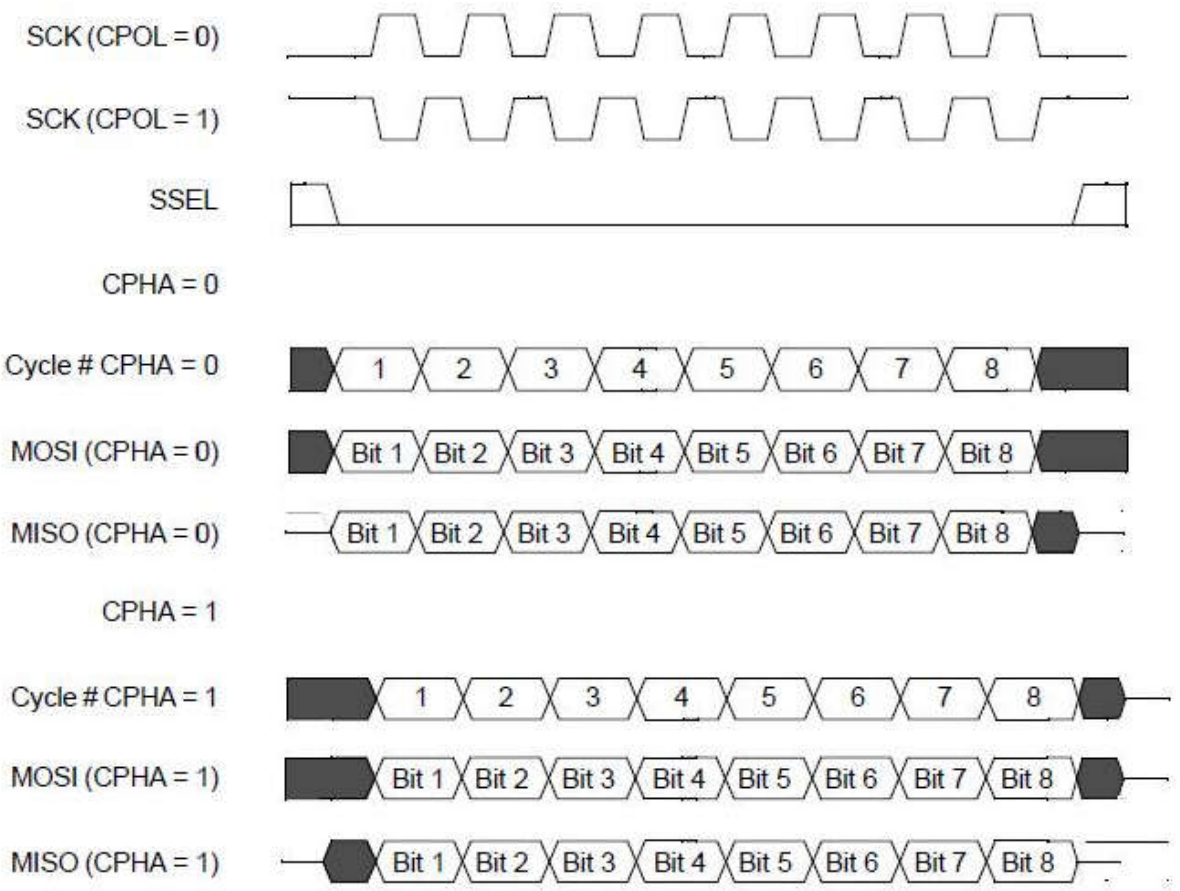
- Bus de donnée série synchrone développé par Motorola
- Communication type maître-esclave
  - Le maître gère la communication
  - Plusieurs esclaves possibles
  - La sélection de l'esclave se fait par une ligne dédiée appelée *chip select*

SCLK — Horloge (généré par le maître)
MOSI — Master Output, Slave Input (généré par le maître)
MISO — Master Input, Slave Output (généré par l'esclave)
SS — Slave Select, Actif à l'état bas, (généré par le maître)



SPI (2)

- Une transmission SPI typique est une communication simultanée entre un maître et un esclave.
  - Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer
  - L'esclave répond aux requêtes du maître
- A chaque coup d'horloge le maître et l'esclave s'échangent un bit.



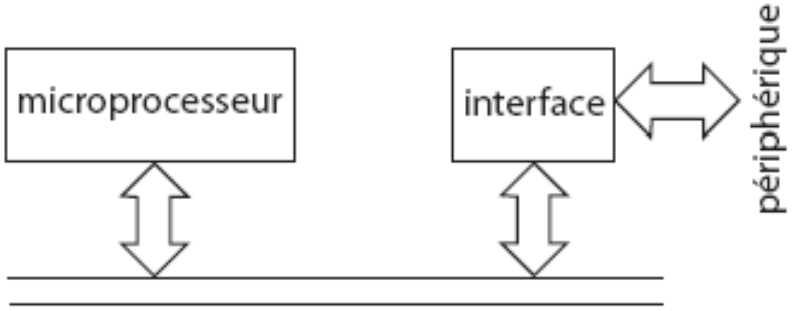
POLYTECH
NICE-SOPHIA

Microcontrôleur ARM Cortex-M

- Microcontrôleur
- Les périphériques
- Interruptions
- Les interruptions du processeur ARM

Les interruptions

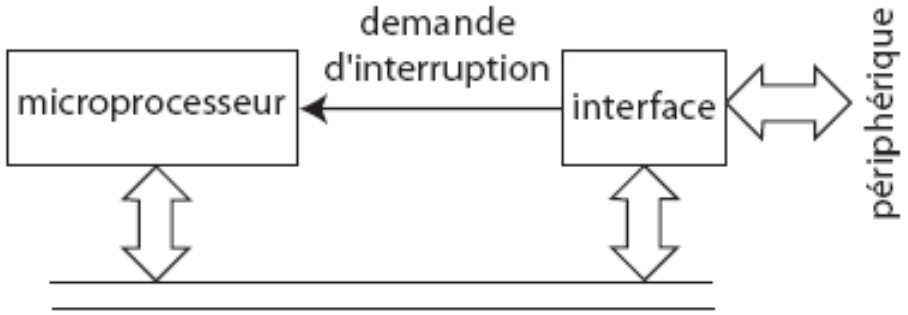
- Un microprocesseur doit échanger des données avec un périphérique
- Première méthode
  - Scrutation périodique (polling): le programme principal contient des instructions qui lisent cycliquement l'état des ports d'E/S



  - Avantage:
    - Facile à programmer
  - Inconvénient:
    - De nouvelles données ne sont pas toujours présentes
    - Des données peuvent être perdues si elles changent rapidement
    - Perte de temps sil y a de nombreux périphériques à interroger

Les interruptions

- Deuxième méthode
  - Interruption: lorsqu'une donnée apparaît sur un périphérique, le circuit d'E/S le signale au microprocesseur par une demande d'interruption (IRQ Interrupt Request)



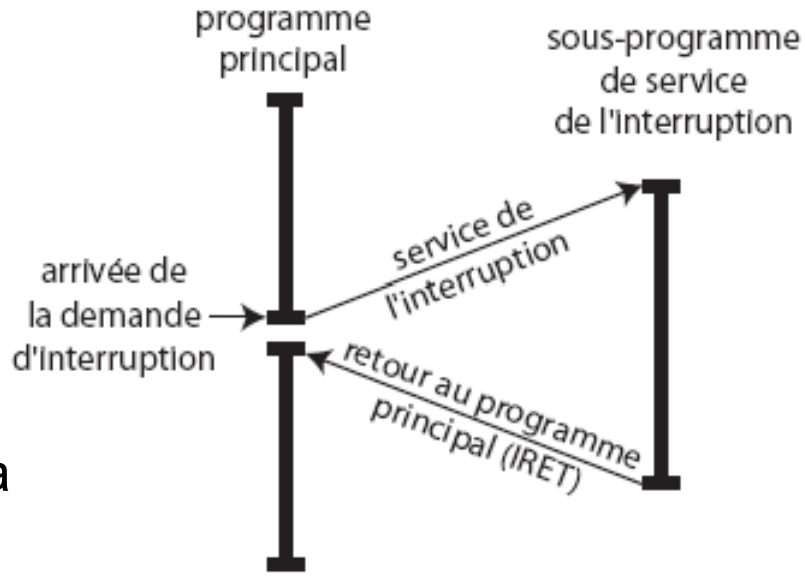
  - Avantage:
    - Le microprocesseur effectue une lecture des ports d'E/S seulement lorsqu'une donnée est disponible
    - Permet de gagner du temps
    - Évite la perte de données

Les interruptions

- Exemples d'interruptions
  - Clavier: demande d'interruption lorsqu'une touche est enfoncée
  - Port série: demande d'interruption lors de l'arrivée d'un caractère sur la ligne de transmission
- Remarque:
  - les interruptions peuvent être générées par le microprocesseur lui-même
    - en cas de problème tels qu'une erreur d'alimentation, une division par zéro ou un circuit mémoire défectueux (erreurs fatales). Dans ce cas, l'interruption conduit à l'arrêt du µP

Les interruptions

- Fonctionnement d'une interruption
  - Le processeur termine l'exécution de l'instruction en cours
  - Il range le contenu des principaux registres dans la pile
  - Il peut émettre un accusé de réception (Interrupt Acknowledge) indiquant au circuit d'E/S que l'interruption est acceptée
    - Il peut refuser l'interruption (masquée)
  - Il abandonne l'exécution en cours et va exécuter un sous-programme de service de l'interruption (ISR)
  - Après exécution de l'ISR, les registres sont restaurés et le processeur reprend l'exécution du programme interrompu



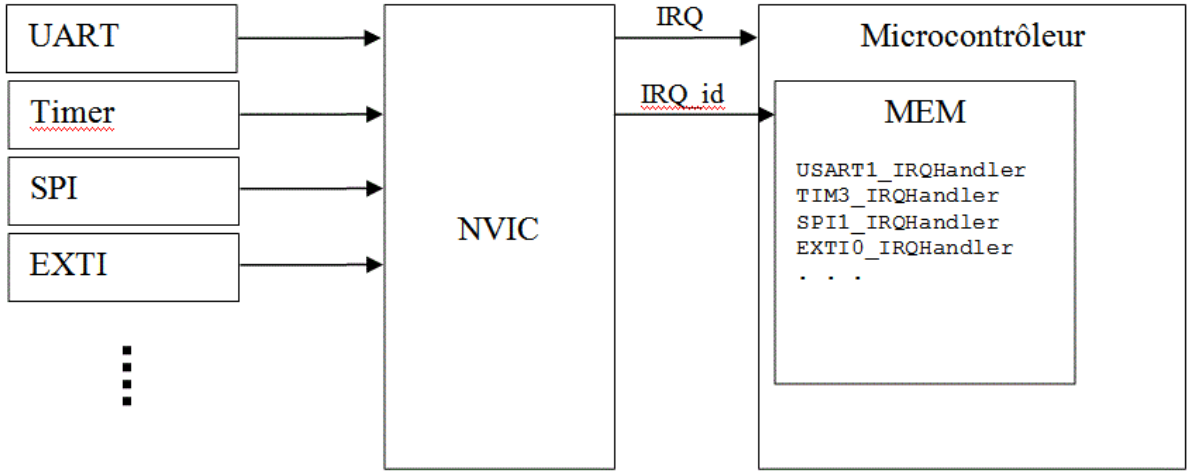
Les interruptions

- Remarques
  - Si plusieurs interruptions peuvent se produire en même temps, on leur affecte une priorité pour que le processeur sache dans quel ordre les exécuter
- Adresse des ISR
  - Lorsqu'une interruption se produit, le processeur a besoin de connaître l'adresse de l'ISR à exécuter
  - L'adresse de l'ISR est contenue dans une zone réservée en mémoire qui s'appelle la table d'interruption.
  - Chaque élément de cette table s'appelle un vecteur d'interruption.
  - AU moment ou survient une interruption, le processeur analyse l'état courant et les priorités afin de savoir quelle ISR doit être exécutée.

Le contrôleur d'interruptions

- S'il est présent, il assiste le processeur dans sa tâche de traitement des interruptions
- Gestion des interruptions périphériques:
  - Tous les périphériques sont connectés au contrôleur d'interruption (chaque périphérique a un numéro d'interruption qui l'identifie).
  - A chaque interruption est associée une priorité (définie par le programmeur en fonction de ses besoins).
  - Lorsqu'une (et/ou plusieurs) interruption(s) se produi(sen)t, le CI analyse les priorités et renvoie l'adresse de la routine d'interruption à exécuter au processeur.

Contrôleur d'interruption STM32




```
NVIC_InitTypeDef NVIC_InitStruct;                    // see misc.h
NVIC_EnableIRQ(TIM4_IRQn);                           //core_cm4.h

NVIC_InitStruct.NVIC_IRQChannel = EXTI15_10_IRQn;    // defined in stm32f4xx.h
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_Init(&NVIC_InitStruct);
NVIC_InitStruct.NVIC_IRQChannel = TIM4_IRQn;         // defined in stm32f4xx.h
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
NVIC_Init(&NVIC_InitStruct);
```

POLYTECH
NICE-SOPHIA

Microcontrôleur ARM Cortex-M

- Microcontrôleur
- Les périphériques
- Interruptions
- Les interruptions du processeur ARM

Les interruptions du processeur ARM

- Exceptions et interruptions du ARM
  - Le processeur ARM possède plusieurs modes de fonctionnement (operating modes).
  - Le mode de fonctionnement courant est le « user mode ».
  - Le processeur peut aussi fonctionner dans des modes « privileged » qui sont utilisés pour la gestion des exceptions et de « supervisor calls » (ex: SWI).
  - Les bits [4..0] du CPSR indiquent le mode de fonctionnement courant du processeur.

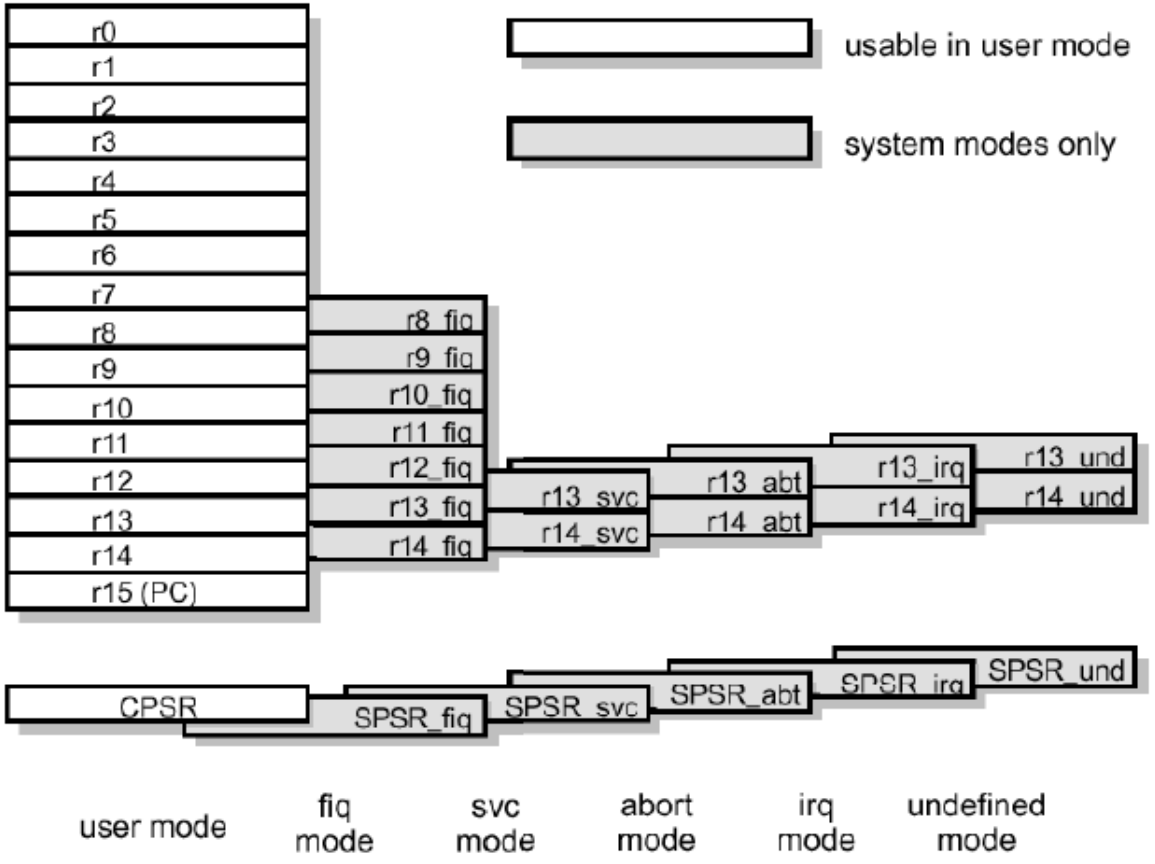
CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Les interruptions du processeur ARM

- Comment sont générées les exceptions?
  - Par défaut le processeur est en mode « user »
  - Il entre dans un mode d'exception quand une exception se produit.
  - Il existe trois type d'exceptions différentes (certaines sont des interruptions):
    - Celles résultant *directement* de l'exécution d'une instruction telle que:
      - Software Interrupt Instruction (SWI)
      - Instruction non-définie ou illégale
      - Erreur mémoire au cours du fetch d'une instruction
    - Celles résultant *indirectement* de l'exécution d'une instruction
      - Memory fault lors d'un accès mémoire
      - Erreur arithmétique (ex: division par zéro)
    - Celles provenant d'un signal de périphérique externe, tel que
      - Reset
      - Fast Interrupt (FIQ)
      - Normal Interrupt (IRQ)

Les interruptions du processeur ARM

- Shadow registers
  - Lorsque le processeur entre dans un mode d'exception, de nouveaux registres sont utilisables:



Les interruptions du processeur ARM

- Shadow registers
  - Par exemple, un événement externe génère une Fast Interrupt (sur la broche FIQ). Le processeur passe en mode de fonctionnement FIQ.
  - Il continue de voir les registres R0-R7 comme précédemment, mais aussi un nouveau groupe de registres R8-R14 ainsi qu'un registre appelé SPSR (Saved Processor Status Register).
  - Ce basculement vers de nouveaux registres permet de préserver l'état du processeur plus facilement. Par exemple, pendant un mode FIQ, les registres R8-R14 peuvent être utilisé librement. Au retour en mode user, les valeurs originales de ces registres sont restaurées automatiquement.

Les interruptions du processeur ARM

- Qu'arrive t'il quand une exception se produit ?
  - Le processeur achève l'instruction en cours.
  - Il abandonne la séquence d'instruction courante en effectuant les étapes suivantes:
    - Il *change de mode de fonctionnement* et passe dans celui correspondant à l'exception.
    - Il *sauvegarde PC dans le registre R14* correspondant au nouveau mode (ex: R14_FIQ).
    - Il *sauvegarde la valeur de CPSR* (avant exception) dans le registre SPSR (Saved Program Status Register).
    - Il *désactive les interruptions moins prioritaires*.
    - Il *force PC avec une nouvelle valeur correspondant à l'exception*. Cela revient à un saut forcé vers le gestionnaire d'exception (Exception Handler) / routine d'interruption (Interrupt Service Routine)

Les interruptions du processeur ARM

- Où se trouve le gestionnaire d'exception ?
  - Les exceptions peuvent être vues comme des appels de sous-fonctions « forcés ».
  - Une adresse unique est prédéfinie pour chaque exception (IRQ, FIQ, etc). C'est cette adresse est chargée dans le PC au déclenchement de l'exception.
  - Cette adresse le vecteur d'interruption/exception. Elle doit contenir un branchement vers la fonction qui décrit le traitement de l'exception (gestionnaire d'exception / routine d'interruption).

Les interruptions du processeur ARM

- Adresses des vecteurs d'interruption
  - Chaque vecteur (sauf FIQ) occupe 4 octets (une instruction)
  - A cette adresse, on place une instruction de branchement vers le gestionnaire d'exception / routine d'interruption:

```
B       exception_handler
```

  - Le traitement des FIQ est particulier pour deux raisons:
    - La routine d'interruption FIQ peut être placée directement à l'adresse du vecteur FIQ (0x0000001C), car celui-ci est à la fin de la table d'interruption.
    - Il y a plus de shadow registers utilisables en mode FIQ. Il y a donc moins de registres à sauvegarder dans la pile par rapport aux autres exceptions d'où un traitement plus rapide.

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Les interruptions du processeur ARM

- Retour d'une exception
  - Lorsqu'une exception a été traitée (par le gestionnaire d'exception), la tâche utilisateur est reprise.
  - Le gestionnaire d'exception doit restaurer l'état du processeur exactement à l'identique de l'état précédent l'exception.
    1. Tout registre modifié doit être restauré depuis la pile utilisée par le gestionnaire d'exception.
    2. Le CPSR doit être restauré depuis le SPSR approprié.
    3. PC doit être restauré avec l'adresse instruction correcte dans le flux d'instruction utilisateur.
  - Les étapes 1 et 3 sont effectuées par l'utilisateur, l'étape 2 est à la charge du processeur.
  - La restauration des registres depuis la pile est la même que pour des appels à sous-fonctions.
  - La restauration du PC est plus complexe. La façon exacte de procéder dépend du type d'exception dont on revient.

Les interruptions du processeur ARM

- Retour d'une exception
  - On suppose que l'adresse de retour a été sauvegardée dans R14 avant d'entrer dans le gestionnaire d'interruption.
  - Pour revenir d'une software interrupt (SWI) ou d'une undefined instruction (UND):

```
MOVS   PC, R14
```

  - Pour revenir d'une IRQ, FIQ, prefetch abort (erreur instruction fetch):

```
SUBS   PC, R14, #4
```

  - Pour revenir d'une data abort (erreur d'accès donnée en mémoire) et ré-essayer l'accès:

```
MOVS   PC, R14, #8
```

  - /!\ Le suffixe S ne sert pas à modifier CPSR, mais à *indiquer la restauration du CPSR quand le registre de destination est PC*.
  - La différence entre ces trois modes est due au pipeline du processeur. La valeur du PC stockée dans R14 peut être en avance d'une ou deux instructions à cause du pipeline.

Les interruptions du processeur ARM

- Priorité des exceptions
  - Plusieurs exceptions peuvent se produire simultanément, un ordre de priorité doit être défini clairement:
    - Reset (priorité la plus forte)
    - Data abort (erreur mémoire lors d'une lecture/écriture de donnée)
    - Fast Interrupt Request (FIQ)
    - Normal Interrupt Request (IRQ)
    - Prefetch abort (erreur lors d'un fetch instruction)
    - Software interrupt (SWI), Undefined instruction
  - Si une IRQ et une FIQ se produisent simultanément, le processeur exécute d'abord le gestionnaire FIQ, et doit se « souvenir » qu'il y a une IRQ en suspens.
  - Au retour de l'interruption FIQ, le processeur procèdera immédiatement à l'exécution du gestionnaire d'IRQ.