IUT
CERGY-PONTOISE

Interruptions

J.Lorandel & O.Romain
MAJ F.Ghaffari & S.Zuckerman – 2020

IUT
CERGY-PONTOISE

Séance 1

IUT CERGY-PONTOISE

Sommaire

- Définition du problème
- Qu'est ce qu'une interruption ?
- Comment gérer les interruptions dans le STM32 ?
- Comment programmer les interruptions ?
- Cas pratiques

Définition du problème

- Les évènements touchant un microcontrôleur sont imprévisibles
  - Il sont dit : asynchrones
  - Pas de connaissance *a priori* sur leur instant d'apparition

- Le déroulement d'un programme est séquentiel et synchrone
  - Besoin de méthode pour capturer ces évènements externes ou internes
  - Comment faire ?

Définition du problème

- Solution : la boucle de scrutation
  - Attente active (polling)

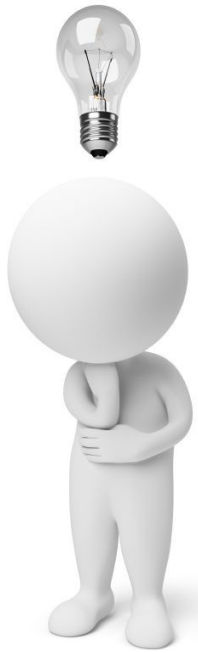
```
#include <stdio.h>
#include <stdlib.h>


int main(void)
{
    while(1)
    {
        avance();
        if (voiture == stop)
        while (!plus_voiture)
            if (voitures == true)
                plus_voiture=0;
       }
    return 0;
}
```

Définition du problème

- Problèmes : boucle de scrutation
  - On ne peut pas régler le temps d'attente de manière précise.
  - Le microprocesseur ne peut rien faire d'autre !
    - 100% du temps processeur est alloué à la tâche de scrutation
    - Le temps alloué à l'évènement est potentiellement long
    - Pas de multi-tâches possible
      - Pas de possibilité de faire d'autres opérations pendant l'attente

Définition du problème

- Pour mettre en œuvre efficacement le multitâche, il faut que le processeur ne soit pas bloqué en attente des opérations d'entrée-sorties, qui doivent se dérouler en parallèle avec les calculs.

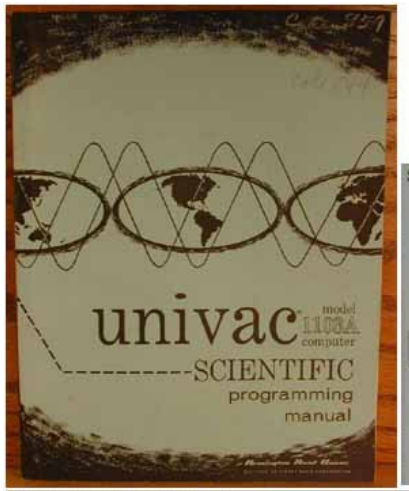


- La gestion des périphériques doit être confiée à des circuits spécialisés. Des commandes (requêtes) seront envoyées au processeur pour le prévenir par un signal d'interruption.

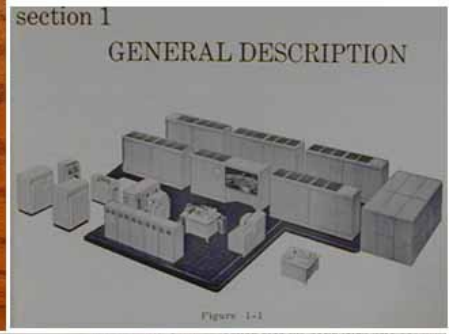
- Le fonctionnement par interruption décharge ainsi le processeur de la surveillance des périphériques. Le processeur peut du coup consacrer son temps à l'avancement des autres tâches.

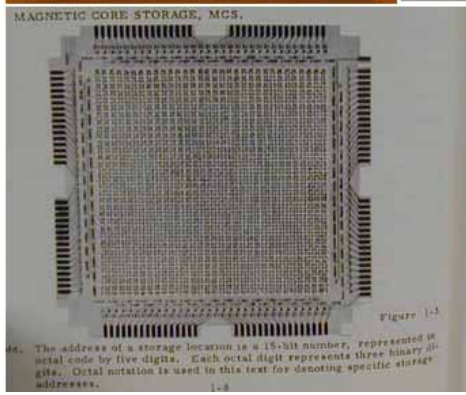
Interruptions

- L'idée des interruptions est apparue en 1955. La NASA possédait un Univac 1103 pour ses besoins de calculs scientifiques
  - Traitement de données d'essais en tunnel de soufflerie avec Boeing) et ses applications administratives. Le programme de collecte de données était chargé en mémoire et présent pendant que les applications de gestion tournaient. Quand les tests en soufflerie étaient prêts, un bouton poussoir située dans le hangar permettait d'activer le programme "instantanément" : le contenu des différents registres de l'ordinateur (compteur ordinal, conditions, …) était alors sauvegardé et le contrôle était transféré au programme de collecte des données.



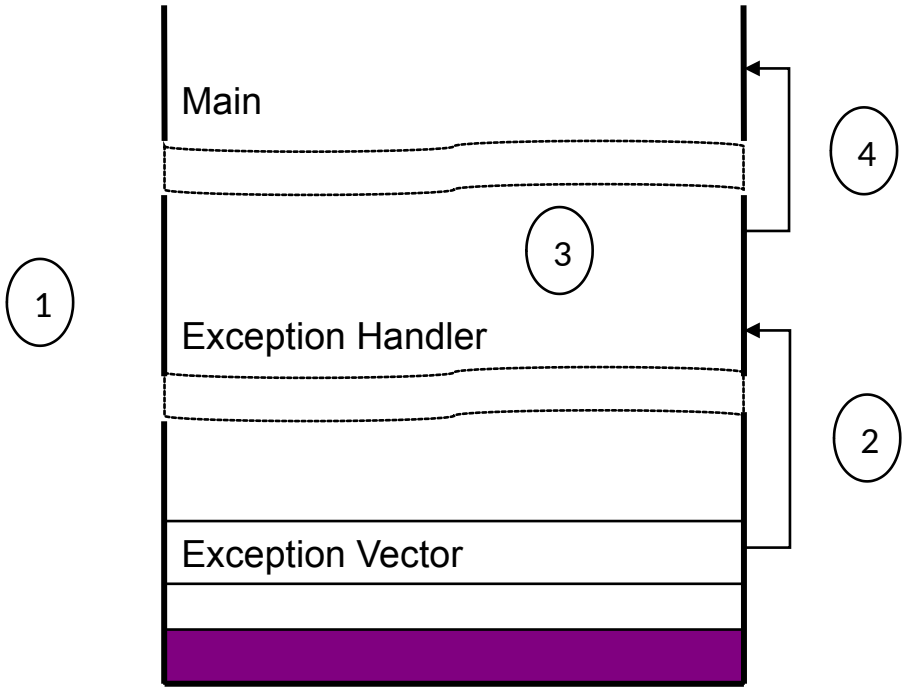
Univac 1103A Computer
Scientific Programming Manual
(1958 - 240 pages)
Very well illustrated with many
pictures, tables, and diagrams!





Interruption : c'est quoi ?

- Une interruption est un événement interne ou externe qui va de manière chronologique :
  1. Arrêter le fonctionnement en cours du programme.
  2. Sauvegarder le contexte d'exécution : état du programme en cours d'instruction (registres et compteur ordinal – program counter)
  3. Exécuter une suite d'instruction pour servir l'interruption : routine de service d'interruption – programme spécifique à l'interruption
  4. Restaurer le contexte précédent d'exécution
  5. Reprendre le fonctionnement du programme



Interruption : classification

- Une interruption est un événement interne ou externe : 2 types

- **Interruptions internes : exceptions**
  - Les interruptions internes sont appelées exceptions. Elles surviennent lors d'une erreur d'exécution du programme (overflow, watchdog, instruction réservée, adressage, etc.)

  - **Propres à l'exécution d'un processus**
  - **Concernent des appels systèmes**
  - **Erreurs d'exécution**

- **Interruptions externes : interruptions**
  - Les interruptions externes concernent essentiellement les demandes de service des périphériques. Elles arrivent de manière asynchrone avec les instructions en cours d'exécution.

  - **Commandée par les périphériques**
  - **Asynchrone / aux instructions en cours**

Architecture du Cortex M4

- Le STM32F446 possède plus de 100 lignes d'interruptions et exceptions systèmes
  - 16 lignes d'exceptions systèmes
    - Reset
    - NMI (Non-masquable Interrupts)
    - SysTick (Timer)
    - Fault, …
  - 91 interruptions liées aux périphériques
    - GPIO : broches d'entrée / sortie
    - Bus : SPI, I2C
    - Périphériques d'acquisition et restitution : ADC et DAC

Interruption : hiérarchisation

Table des exceptions et interruptions

exceptions

Position	Priority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000_0000
	-3	fixed	Reset	Reset	0x0000_0004
	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
	-1	fixed	HardFault	All class of fault	0x0000_000C
	0	settable	MemManage	Memory management	0x0000_0010
	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
	-	-	-	Reserved	0x0000_001C - 0x0000_002B
	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
	-	-	-	Reserved	0x0000_0034
	5	settable	PendSV	Pendable request for system service	0x0000_0038
	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048

exceptions

Interruptions

Position	Priority	Type of priority	Acronym	Description	Address
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
19	26	settable	USB_HP_CAN_TX	USB High Priority or CAN TX interrupts	0x0000_008C
20	27	settable	USB_LP_CAN_RX0	USB Low Priority or CAN RX0 interrupts	0x0000_0090
21	28	settable	CAN_RX1	CAN RX1 interrupt	0x0000_0094
22	29	settable	CAN_SCE	CAN SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4

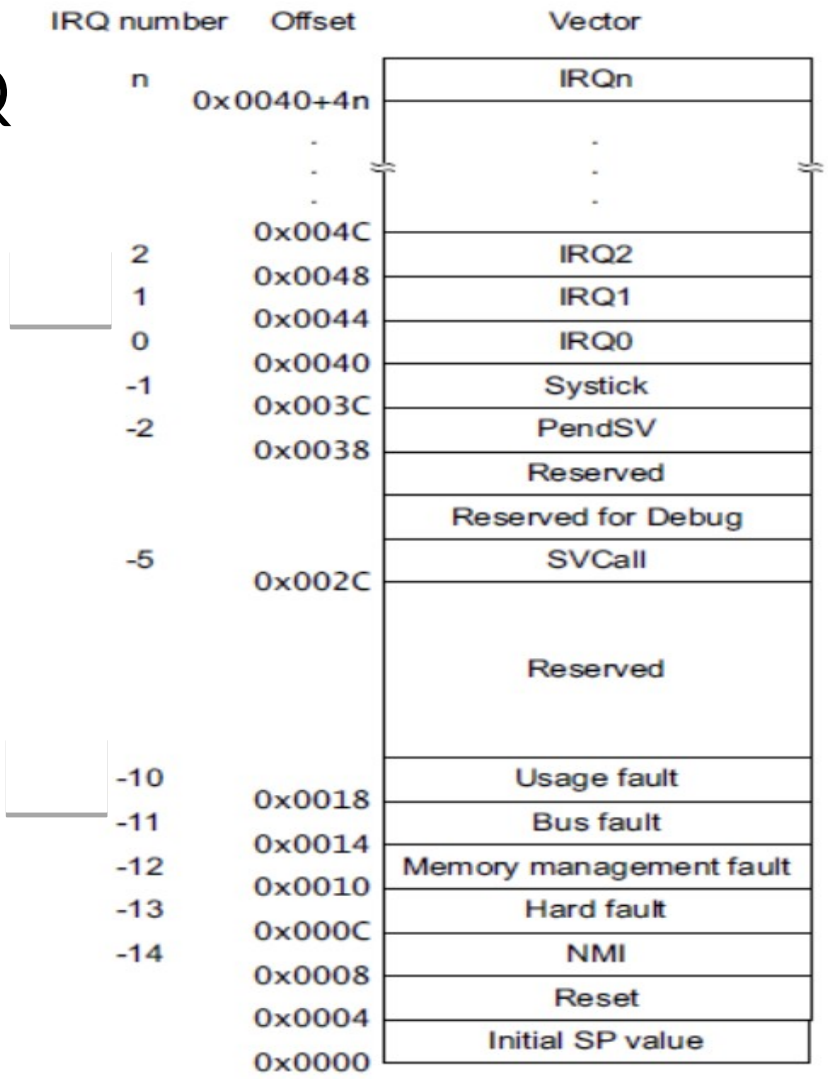
Interruption : hiérarchisation

- Si deux interruptions arrivent en même temps, comment fait on ?

- Hiérarchisation
  - Niveaux de priorité
    - IRQ : Interruption Request
  - Table des vecteurs d'interruption
  - Contrôleur d'interruption
    - Organisation des services

Interruption : hiérarchisation

- Exemple de hiérarchisation et d'IRQ
  - Fixée par un plan mémoire
  - IRQ1 plus prioritaire que l'IRQn

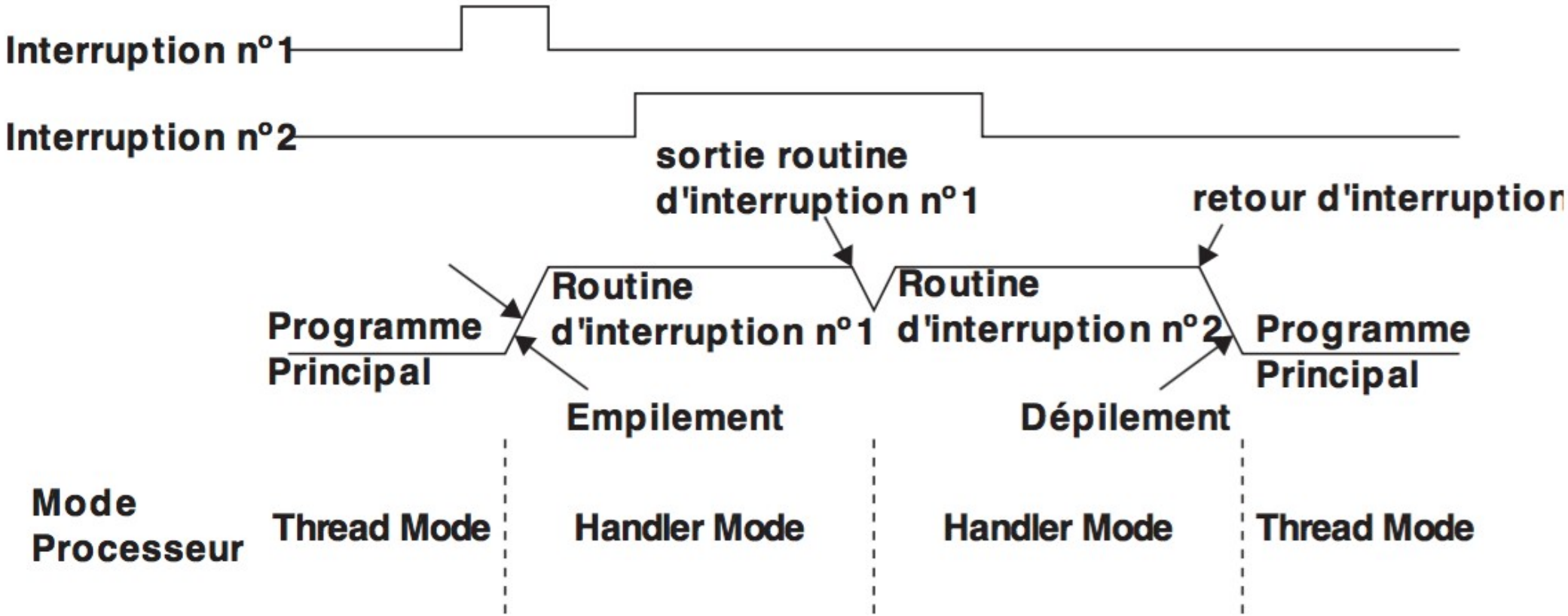


Interruption : hiérarchisation

- La hiérarchisation repose sur une politique de gestion des interruptions
- Elle doit permettre qu'une interruption plus prioritaire puisse interrompre une interruption moins prioritaire en cours d'exécution.
- Elle doit interdire qu'une interruption moins prioritaire ne puisse pas interrompre une interruption plus prioritaire en cours d'exécution.
- Elle peut fixer des priorités non modifiables
- Elle permet de fixer les niveaux de priorités de la plupart des sources d'interruption
  - Souplesse / à l'utilisateur

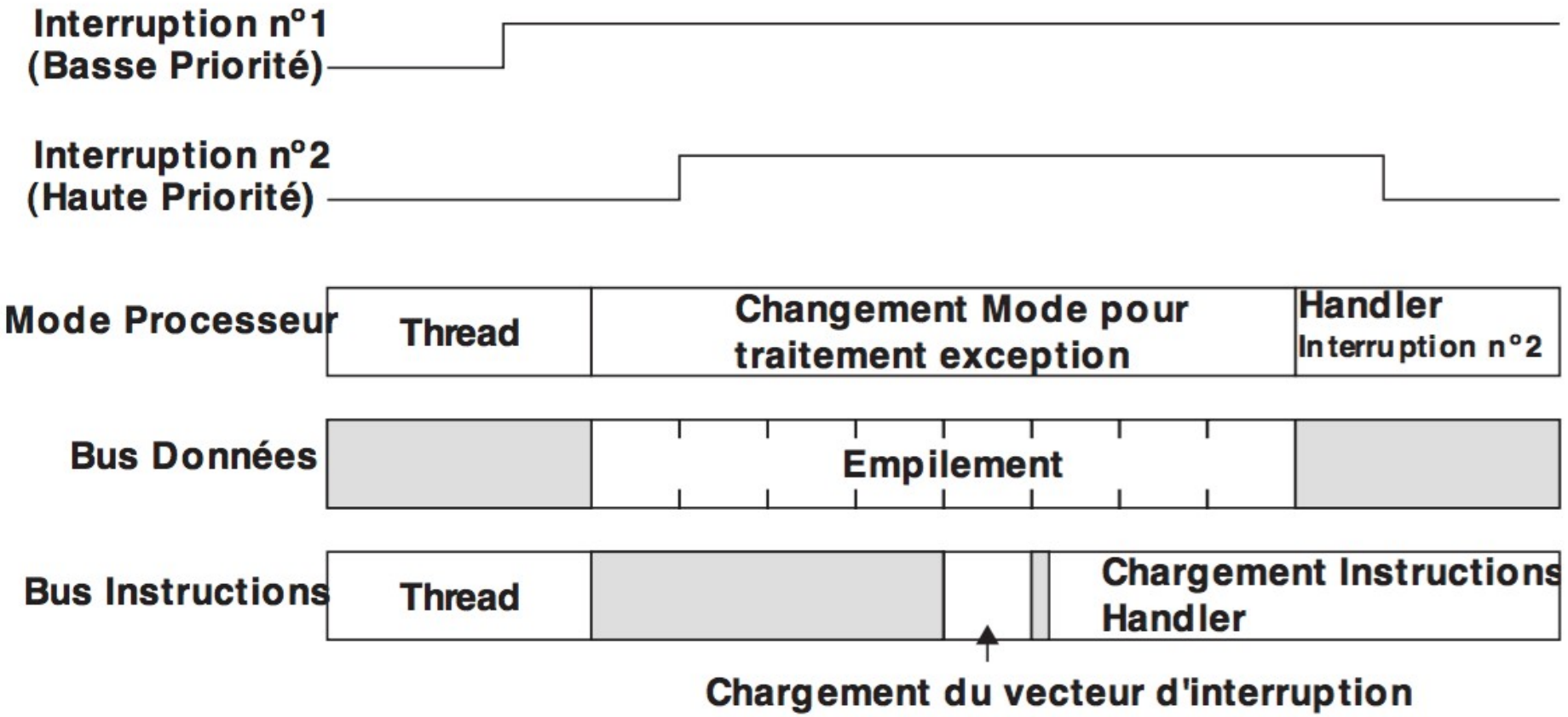
Interruption : typologie

- Niveau Interruption 1 > Niveau Interruption 2



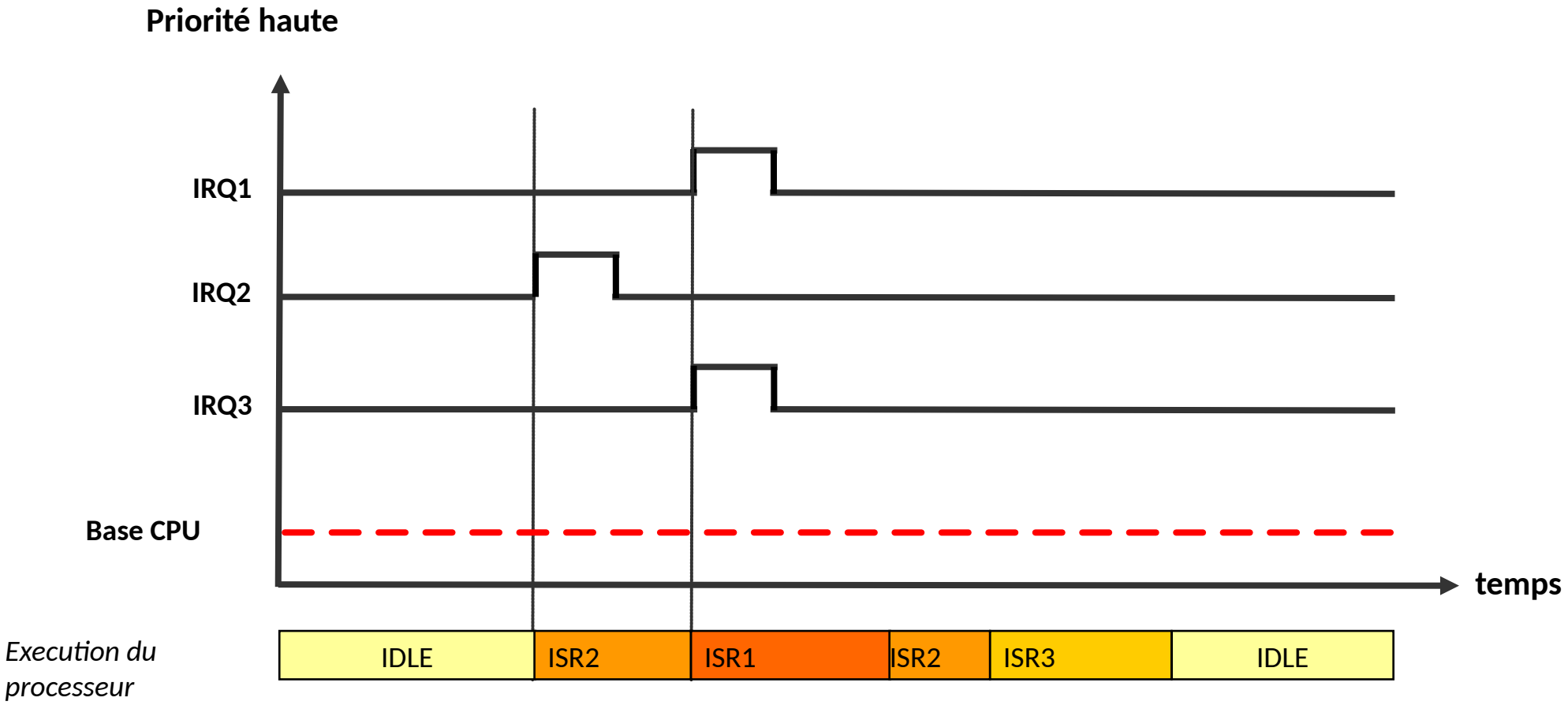
Interruption : typologie

- Niveau Interruption 1 > Niveau Interruption 2



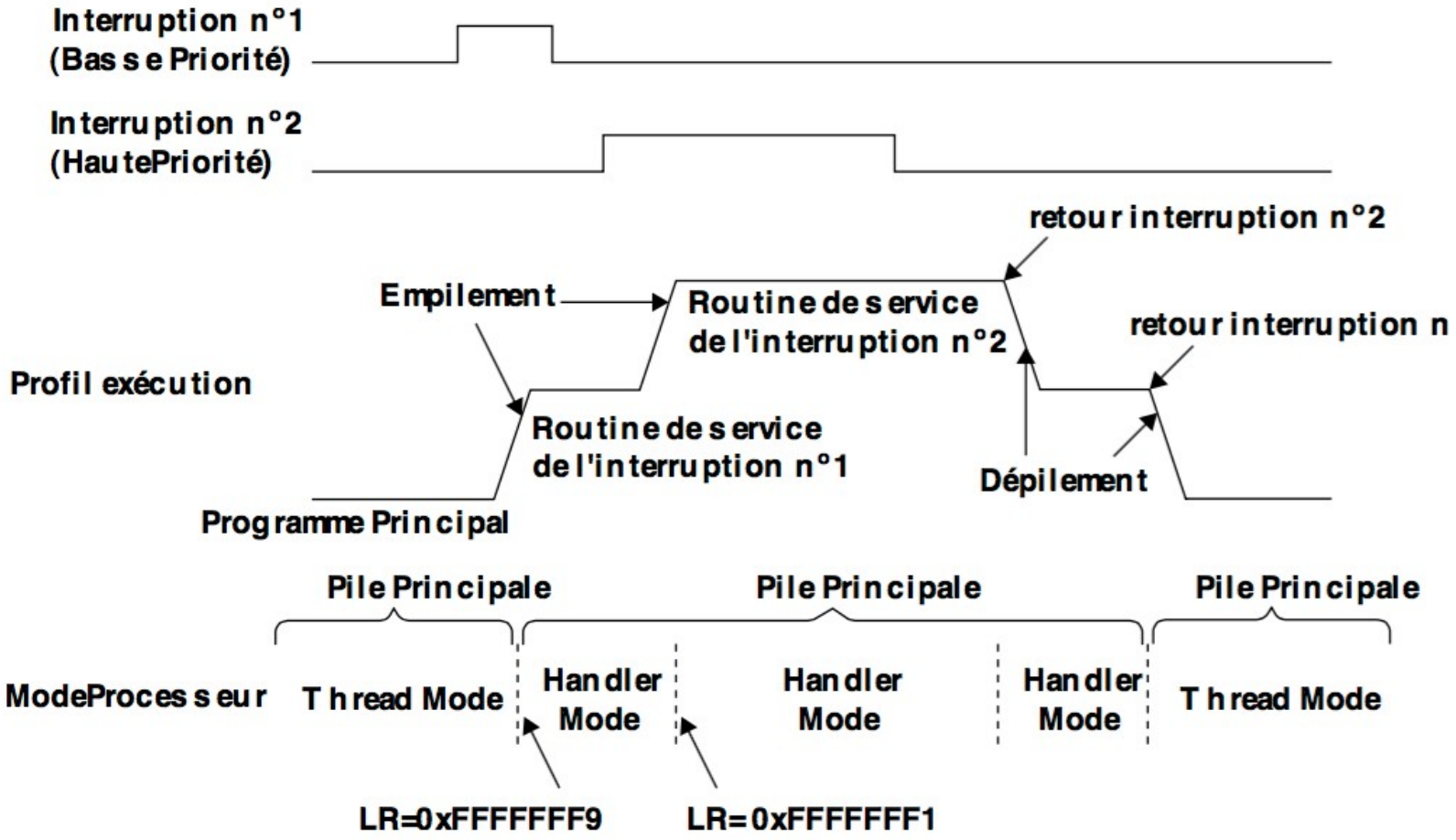
Interruption : typologie

- Cas de trois interruptions



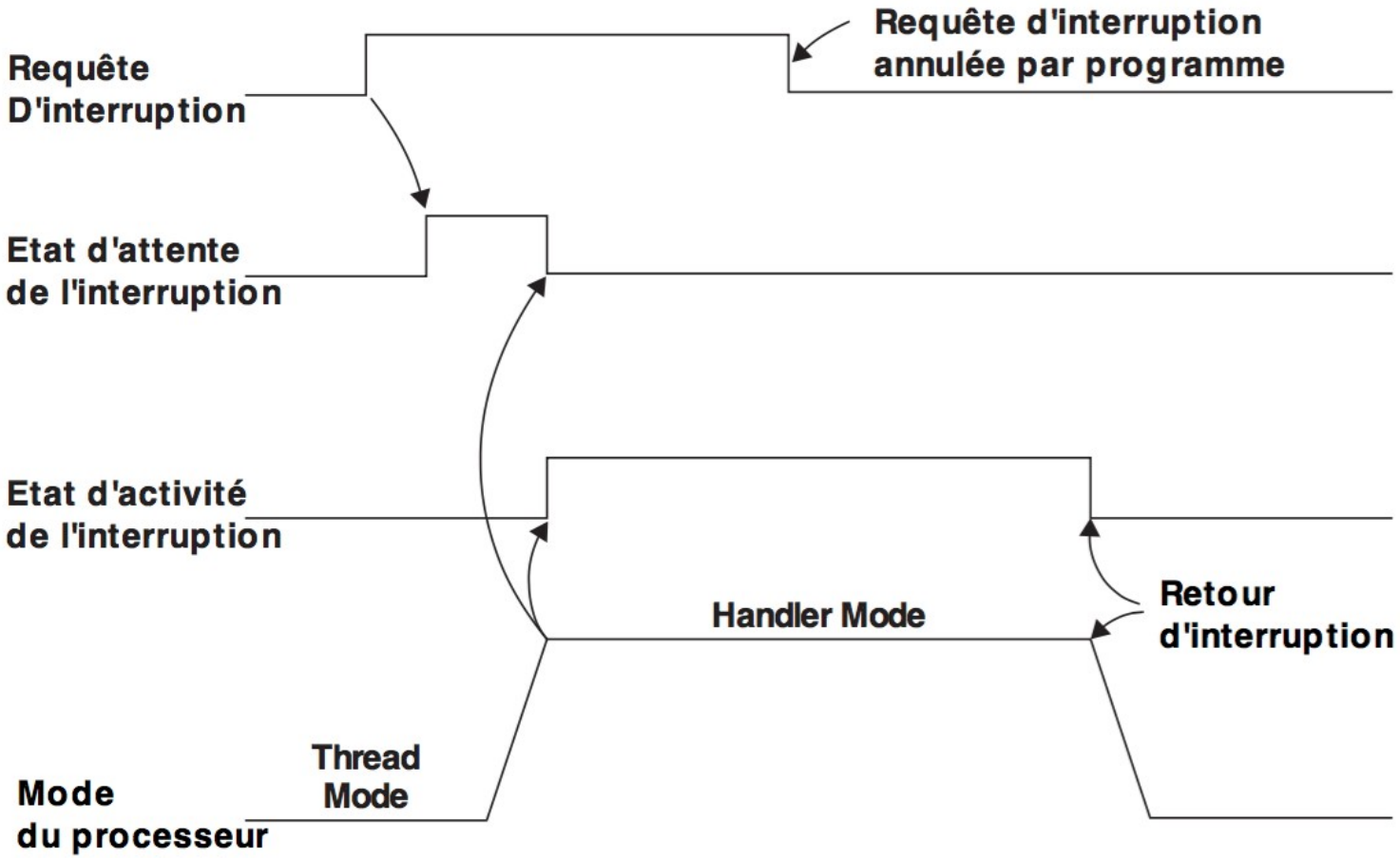
Interruption : typologie

- Niveau Interruption 2 > Niveau Interruption 1



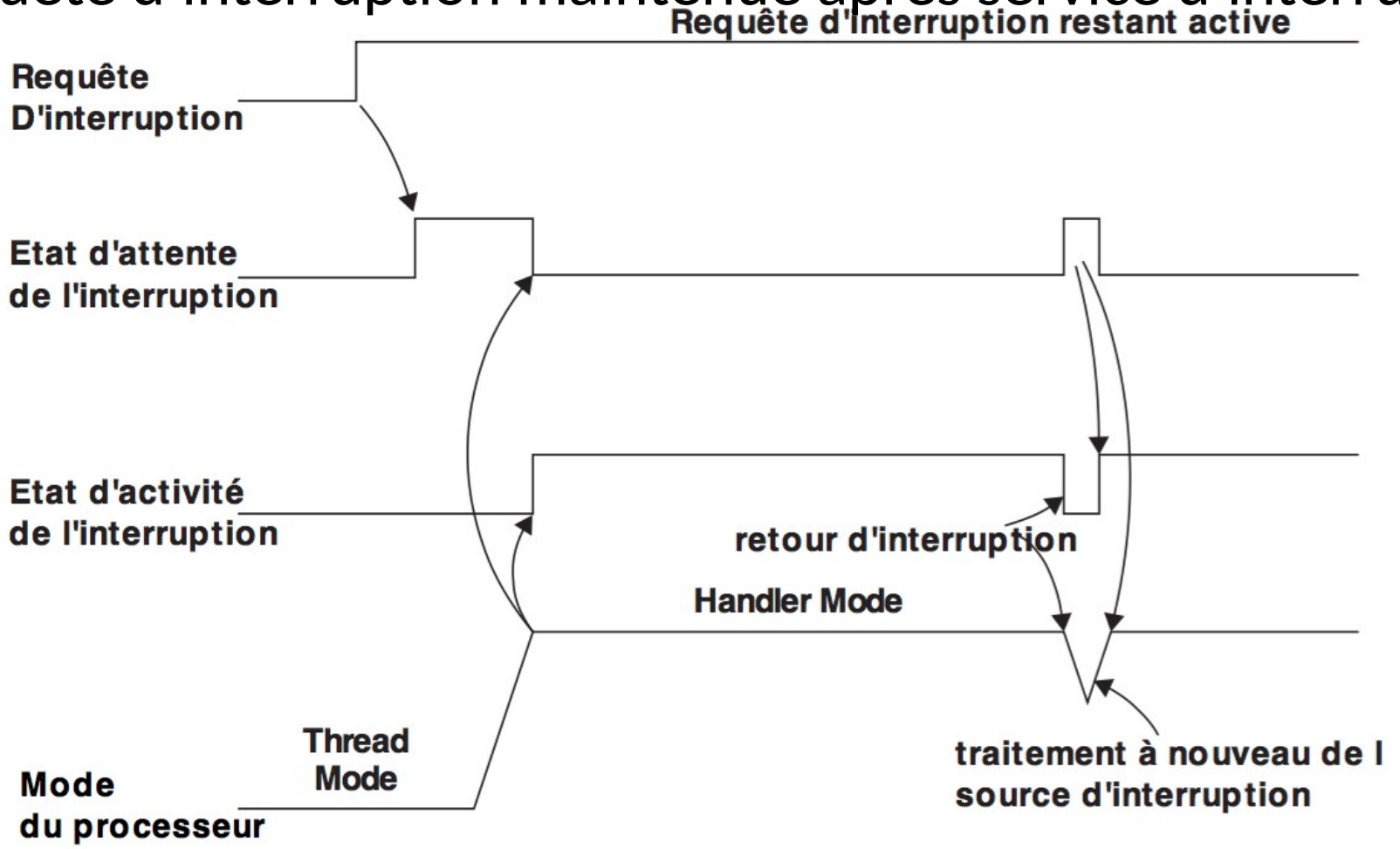
Interruption : typologie

- Requête d'interruption maintenue lors du début d'interruption



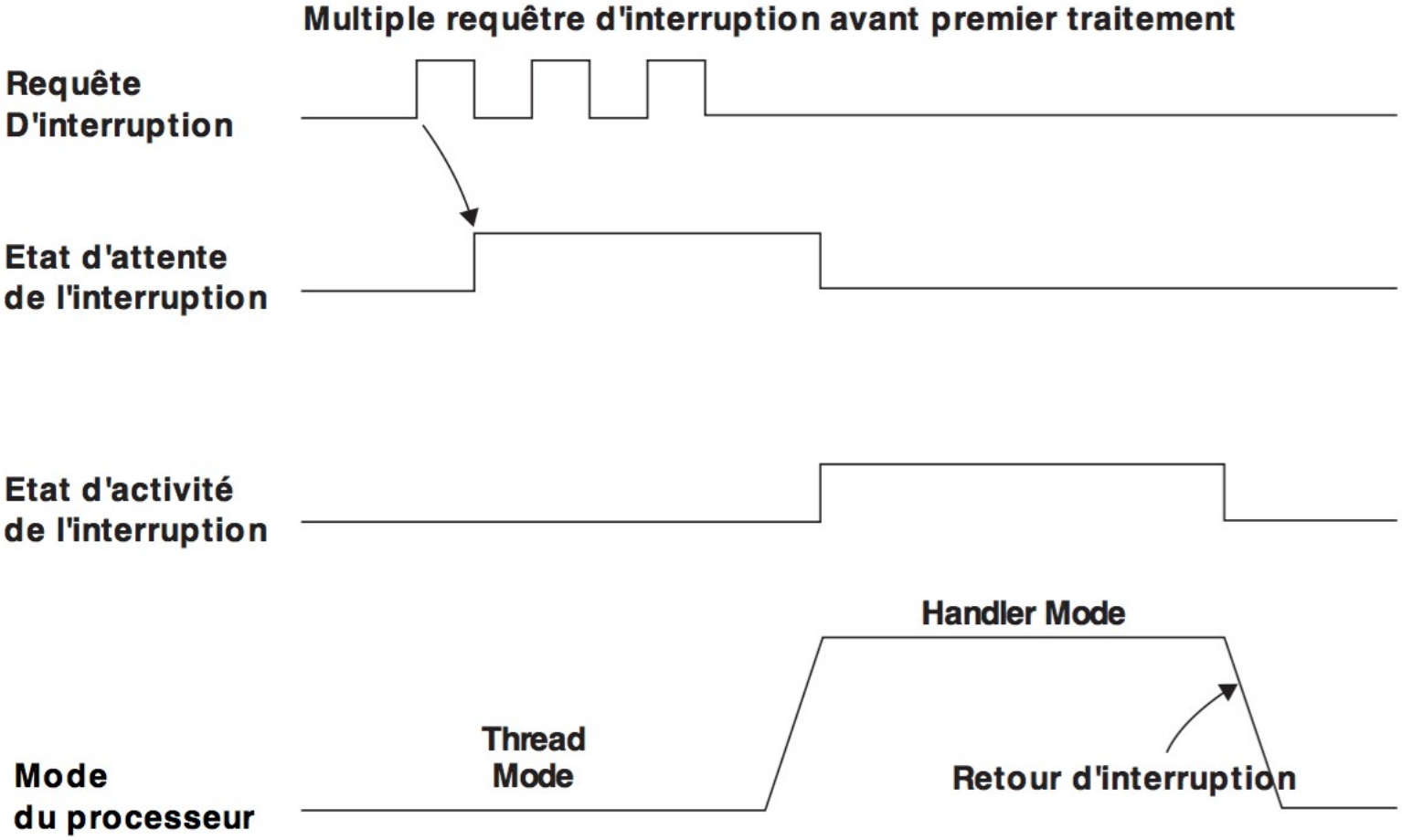
Interruption : typologie

- Requête d'interruption maintenue après service d'interruption



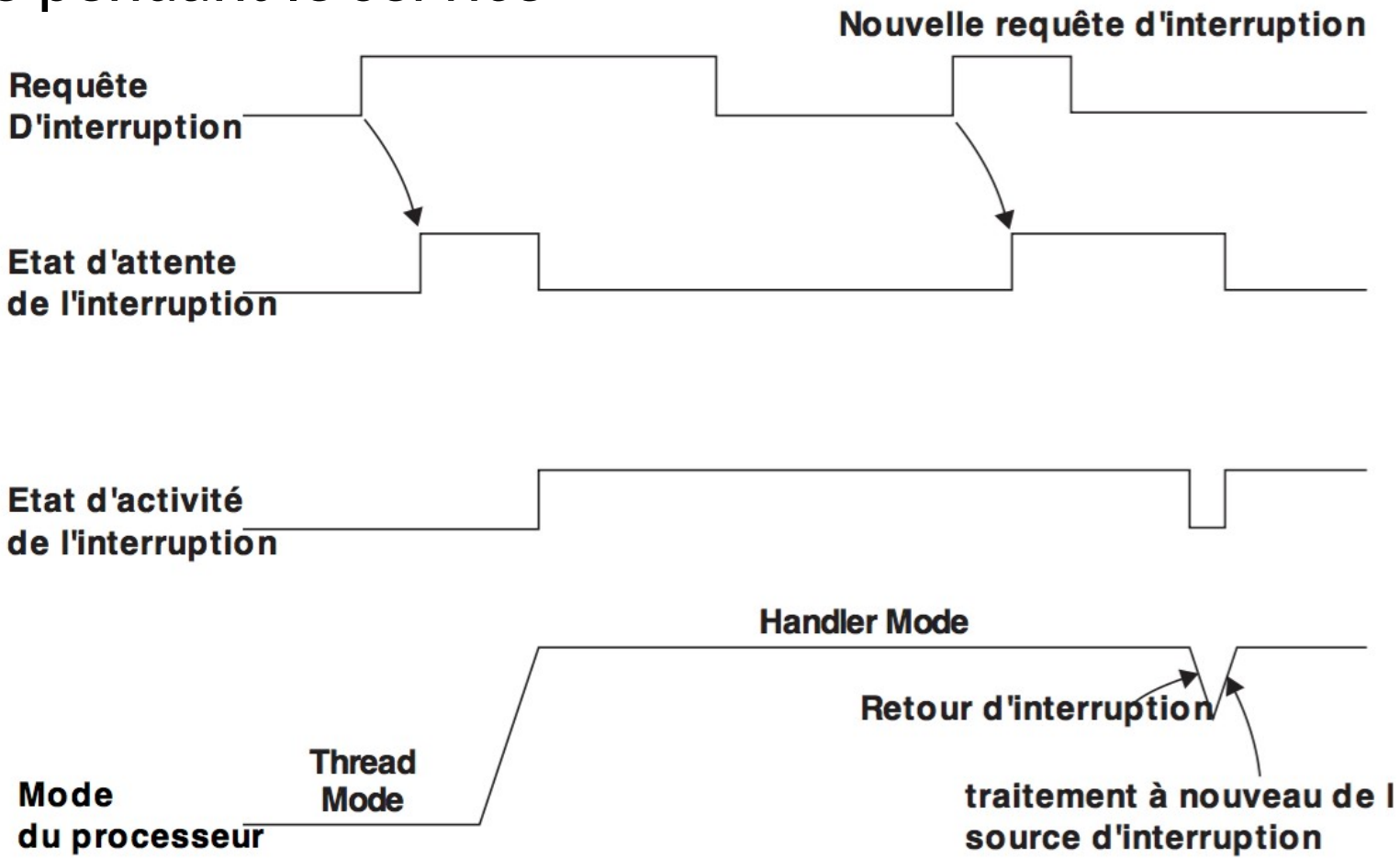
Interruption : typologie

- Requêtes multiples



Interruption : typologie

- Requête pendant le service



Interruption : masquage ou non traitement

- Masquage des interruptions
  - Retardement de la prise en compte
    - Nécessite une hiérarchisation des interruptions de priorité inférieure
  - Mémorisation des interruptions masquées
    - Notion de Pile
      - FIFO pour les interruptions

- Désarmer les interruptions
  - Essentiellement les interruptions non systèmes

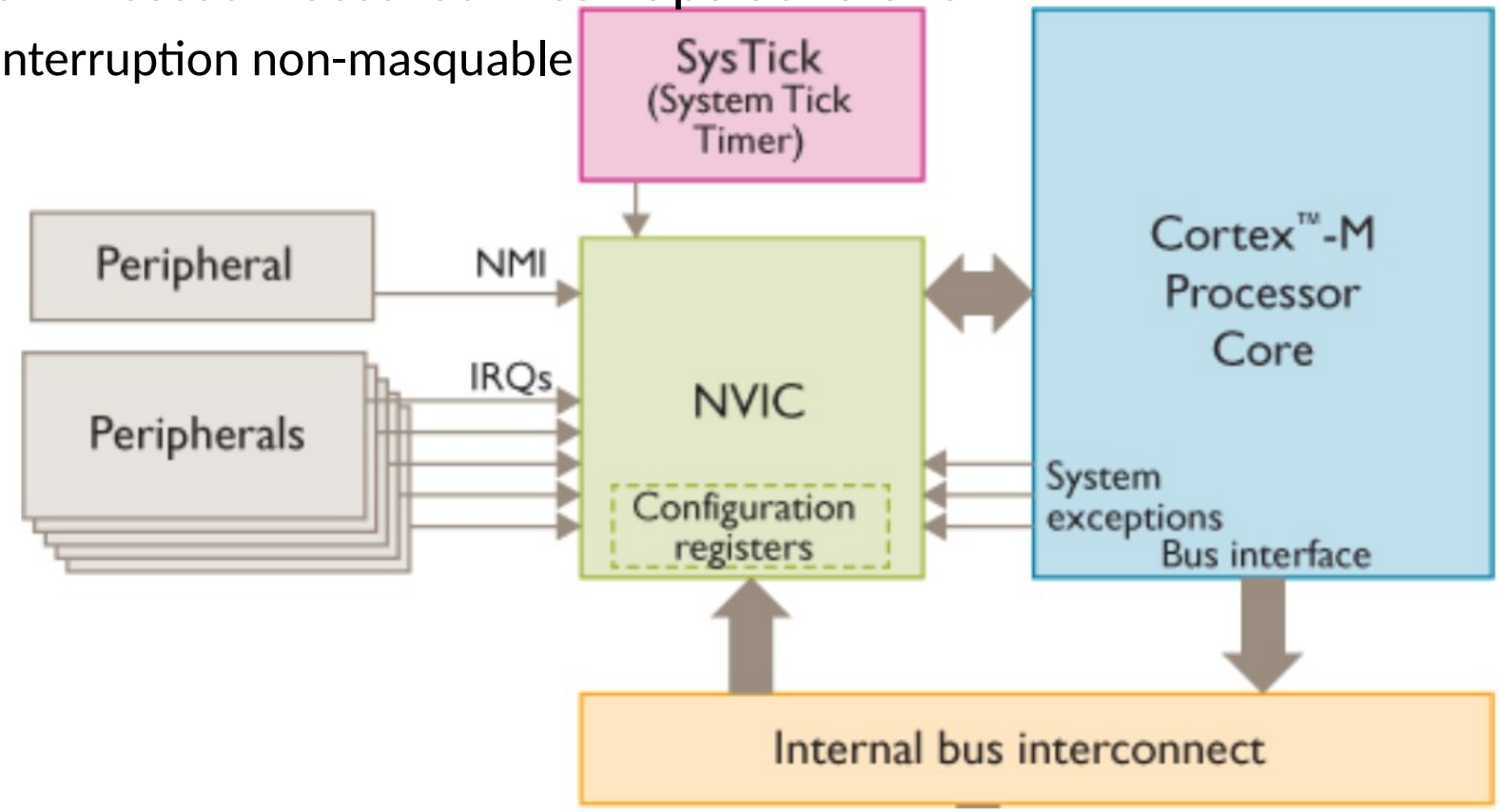
IUT
CERGY-PONTOISE

Gestion des interruptions dans le STM32F

J.Lorandel & O.Romain
MAJ F.Ghaffari & S.Zuckerman – 2020

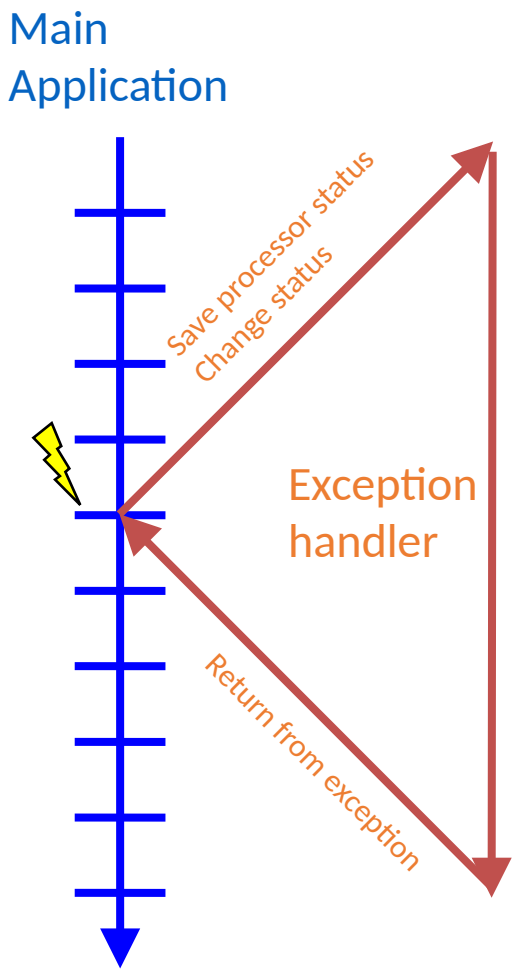
Architecture du Cortex M4

- Gestion des interruptions au niveau architecture
  - Contrôleur NVIC – Nested Vectored Interrupt Controller
    - Supporte une interruption non-masquable



Architecture du Cortex M4

- Les interruptions du STM32 sont dites « vectorisées »
  - Quand une interruption intervient, la « bonne » routine doit être fournie : **Interrupt Handler**
  - Pour cela il faut **déterminer l'adresse du programme** correspondant à cette routine
  - Cette information est stockée dans une **table des vecteurs d'interruption**
  - Par défaut cette table est stockée à l'adresse 0
  - Cette table est relogeable via le registre VTOR (Vector Table Offset Register), c'est à dire qui peut être mis en mémoire à un autre endroit)
  - La table est organisée en mots de 4 octets (32 bits)
  - L'adresse du vecteur d'une interruption est calculée en multipliant par 4 son numéro.



Architecture du Cortex M4

- Table des vecteurs d'interruption

Adresse	Numéro de l'exception	Valeur
0x00000000	–	Valeur initiale de MSP
0x00000004	1	Valeur initiale du Compteur de Programme (vecteur de Reset)
0x00000008	2	Adresse de la fonction de gestion de l'exception NMI
0x0000000C	3	Adresse de la fonction de gestion de l'exception HardFault
...	...	Autres adresse des fonctions de gestion des exceptions

Architecture du Cortex M4

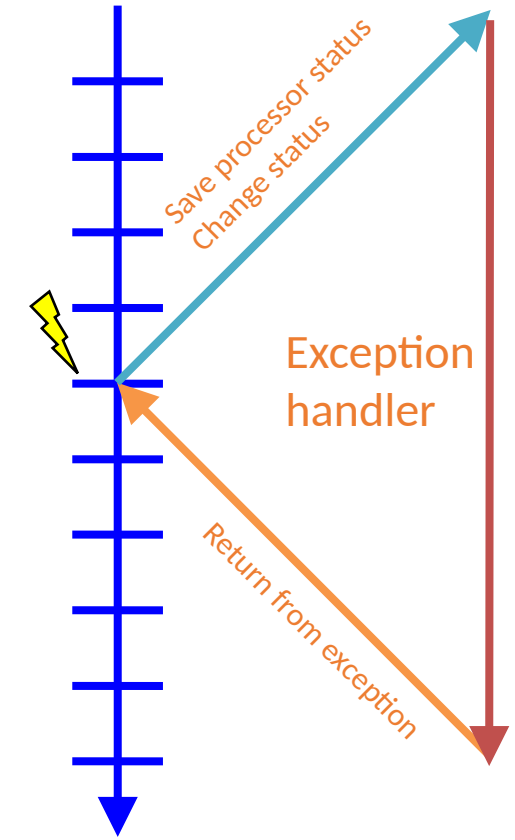
- Registre Vector Table Offset – VTO

Bits	Nom	Type	Valeur initiale	Description
29	TBLBASE	R/W	0	Base de la table : 0 pour ROM, 1 pour RAM
28:7	TBLOFF	R/W	0	Valeur du décalage de la table des vecteurs d'exceptions

Architecture du Cortex M4

- Traitement d'une interruption par le Cortex M4 comporte trois phases :
  - Phase 1 : Sauvegarde du contexte du programme
  - Phase 2 : Chargement de la routine d'interruption – Interrupt Handler
  - Phase 3 : Mise à jour du pointeur de pile SP (Stack Pointer), du registre de lien LR (Link Register) et du compteur ordinal PC (compteur de programme)

Architecture du Cortex M4

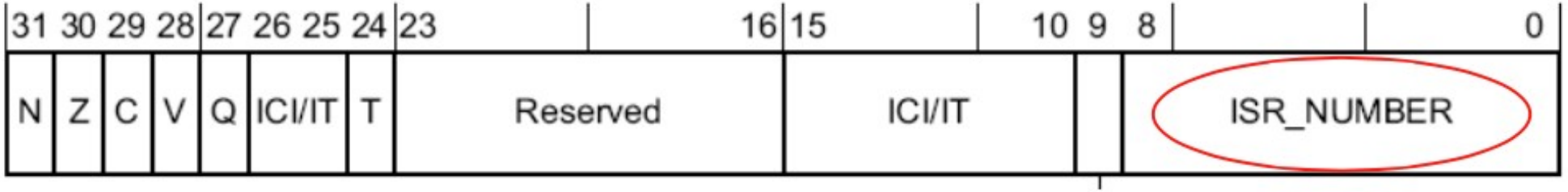


- **Phase 1 : Sauvegarde du contexte du programme**
  - Registrement des registres xPSR, PC, LR, R12, R3-R0
  - Sauvegarde de CPSR dans SPSR
  - Sauvegarde de PC dans LR
- **Phase 2 : Chargement de la routine d'interruption – Interrupt Handler**
  - Changement du mode du processeur
  - PC = adresse du service d'interruption
  - Exécution du code utilisateur
- **Phase 3 : Mise à jour** du pointeur de pile SP (Stack Pointeur), du registre de lien LR (Link Register) et du compteur ordinal PC (compteur de programme)
  - Dépilement des registres après le service de l'interruption
  - Restauration du contexte du programme avant l'interruption (xPSR, PC, LR, R12, R3-R0)
  - Restauration de SPSR dans CPSR
  - Restauration de PC à partir de LR

Séance 2

Architecture du Cortex M4

- Le numéro de l'exception en cours de service peut être connu par :
  - Le registre PSR – Processor Status Register
  - 9 premiers bits correspondent à l'ISR en cours d'exécution



  - Le registre ICSR (Interrupt Control State Register) du NVIC dans le champ VECTACTIVE

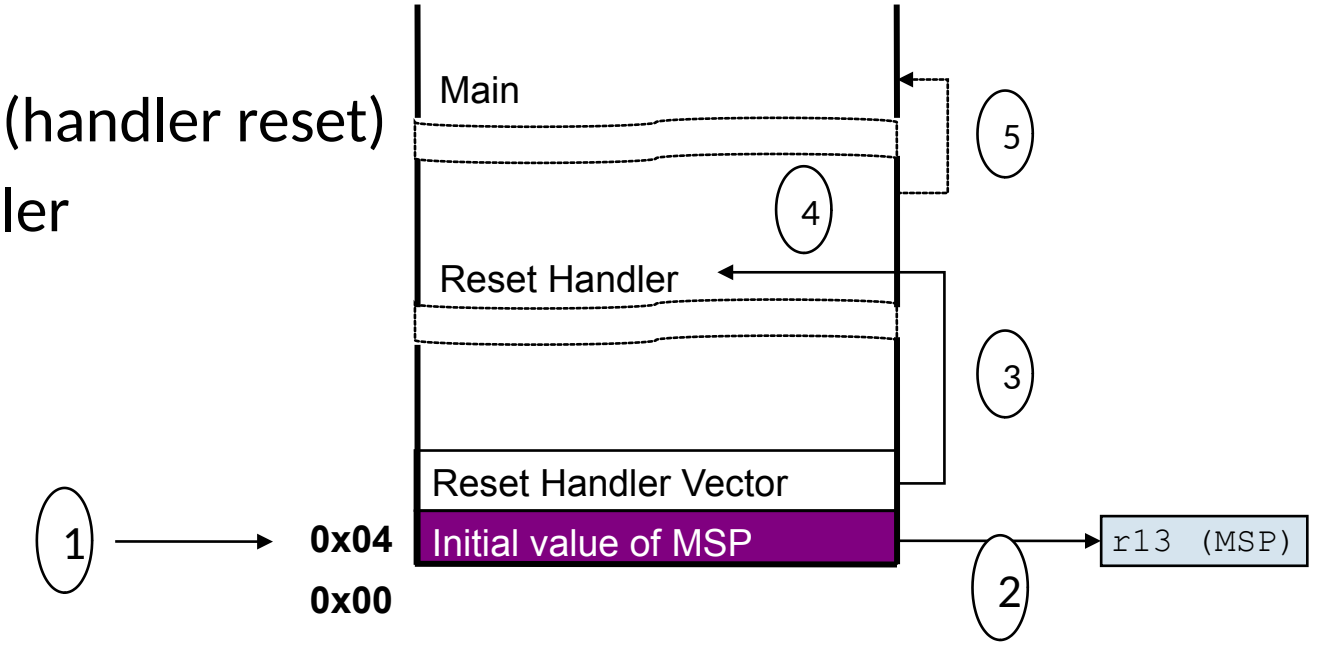
Architecture du Cortex M4

- Les vecteurs des exceptions sont rangés en mémoire :
  - Les uns à la suite des autres
  - Dans une structure de type tableau nommée table des vecteurs
  - Ils sont repérés individuellement et peuvent être calculés à partir de leur numéro d'exception après les 16 places réservées au début de la table (offset)

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Architecture du Cortex M4

- Exemple : cas du RESET
  1. Première entrée 0x00 contient la valeur initiale du MSP (Main Stack Pointer)
  2. Lance la routine depuis l'adresse 0x04 (handler reset)
  3. Execution de la routine en mode Handler
  4. Execution du programme principale



Architecture du Cortex M4

- Exemple : EXT1
  - 16 × 4 + numéro × 4
  - 16 × 4 + 7 × 4 = 0x5C

Position	Priority	Type of priority	Acronym	Description	Address
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
19	26	settable	USB_HP_CAN_TX	USB High Priority or CAN TX interrupts	0x0000_008C
20	27	settable	USB_LP_CAN_	USB Low Priority or CAN RX0	0x0000_0090

Architecture du Cortex M4

- Vecteur d'interruption
  - On appelle vecteur d'une interruption l'adresse à laquelle démarre le code (ou la routine) de cette interruption.
    - Elle est définie sur 32 bits

- Exemple
  - Si l'adresse associée à l'interruption TIMER6 est de 0x80000240
  - La première instruction du traitement associée à cette interruption se situera donc à l'adresse 0x80000240

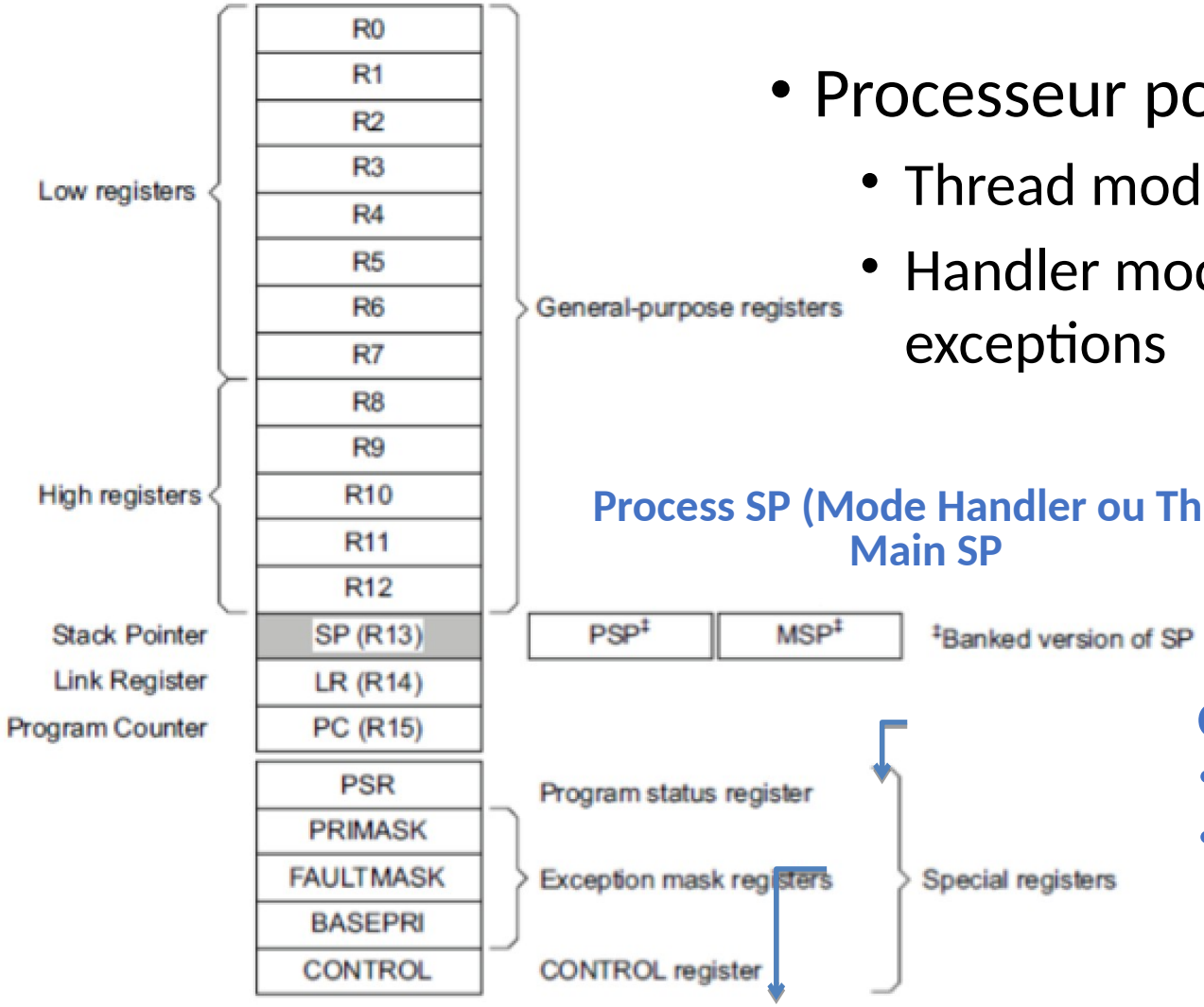
Architecture du Cortex M4

- Table des vecteurs
  - Elle commence par défaut à l'adresse 0x00000000
  - Cette adresse est en ROM (mémoire flash)
  - Les vecteurs peuvent être modifiés lors de l'exécution d'un programme – table relogeable
  - Le registre SCB_VTOR contient l'adresse à laquelle débute la table des vecteurs
    - Initialement SCB_VTOR = 0x00000000
    - Les vecteurs se repère / à la valeur de l'interruption

Adresse	***Mémoire Donnée sur 32 bits***
VTOR	Réservé
VTOR + 1*4	Vecteur exception 1
VTOR + 2*4	Vecteur exception 2
VTOR + 3*4	Vecteur exception 3
VTOR + 4*4	Vecteur exception 4
VTOR + 5*4	Vecteur exception 5
VTOR + 255*4	Vecteur exception 255

38

Architecture du Cortex M4



- Processeur possède deux modes
  - Thread mode pour les tâches utilisateurs
  - Handler mode pour les tâches systèmes et les exceptions

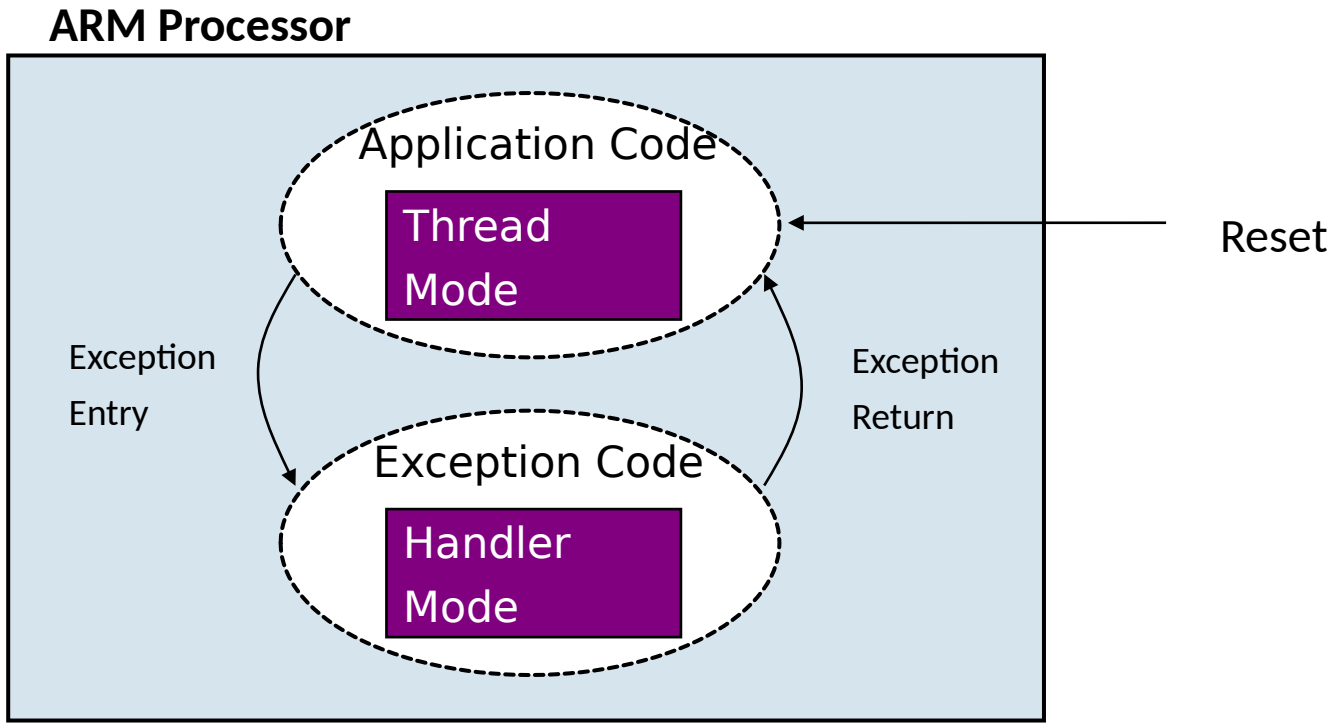
Process SP (Mode Handler ou Thread)
Main SP

Convention :
- **Process SP en mode thread**
- **MSP en mode handler**

39

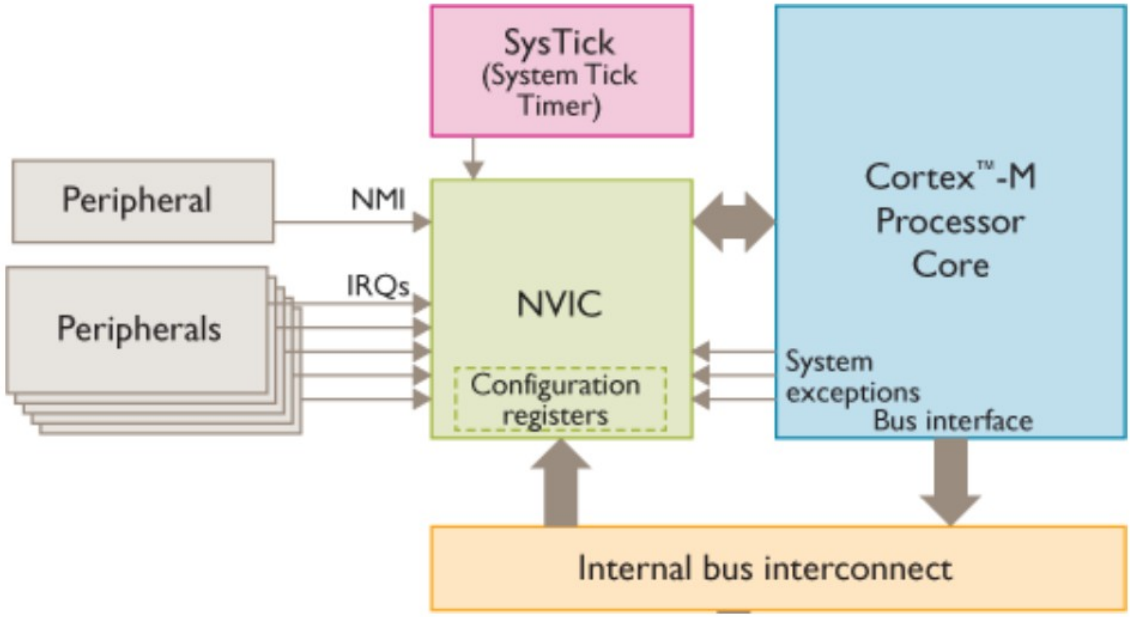
Architecture du Cortex M4

- Les modes opératoires
- Thread mode – traitement normal
- Handler mode – traitement des exceptions / interruptions



Contrôleur NVIC

- Contrôleur NVIC – Nested Vectored Interrupt Controller
- Le NVIC permet de contrôler 82 sources d'interruptions provenant des périphériques
- Le NVIC interrompt le CPU avec l'IRQ de plus haute priorité
  - Le CPU utilise le numéro d'IRQ pour accéder à la routine d'interruption via son adresse en mémoire (table d'interruption)



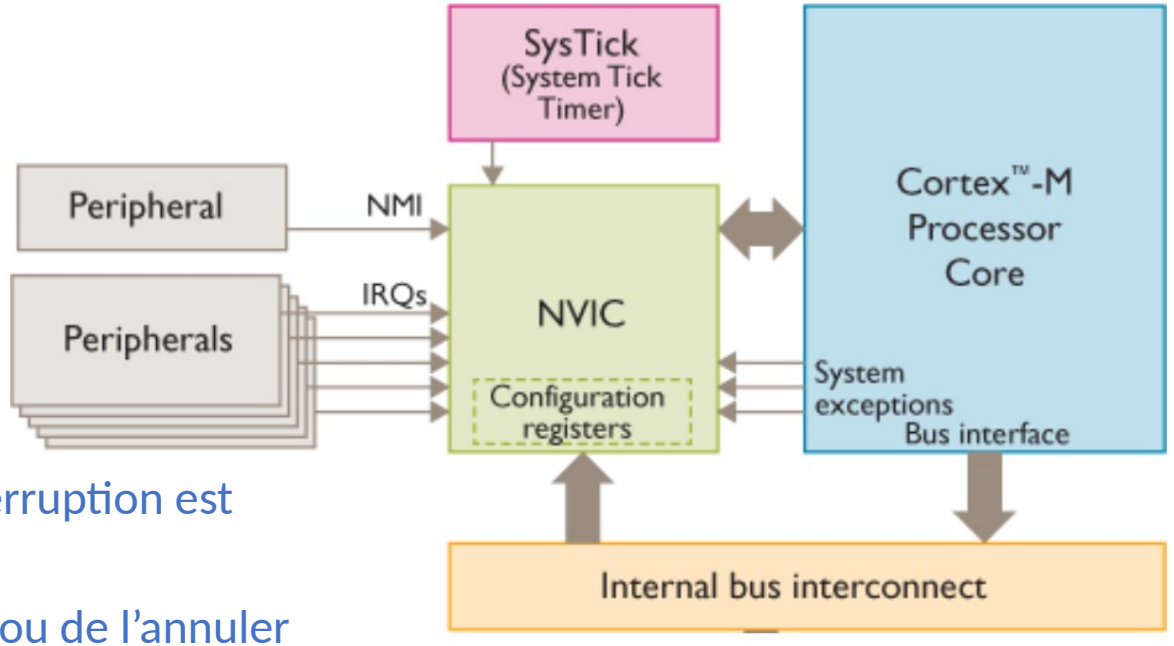
Contrôleur NVIC

- Les registres du NVIC

Address	Name	Type	Required privilege	Reset value	Description
0xE000E100-0xE000E11C	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000	*Interrupt Set-enable Registers*
0XE000E180-0xE000E19C	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000	*Interrupt Clear-enable Registers*
0XE000E200-0xE000E21C	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000	*Interrupt Set-pending Registers*
0XE000E280-0xE000E29C	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000	*Interrupt Clear-pending Registers*
0xE000E300-0xE000E31C	NVIC_IABR0-NVIC_IABR7	RW	Privileged	0x00000000	*Interrupt Active Bit Registers*
0xE000E400-0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000	*Interrupt Priority Registers*
0xE000EF00	STIR	WO	Configurable[a]	0x00000000	*Software Trigger Interrupt Register*

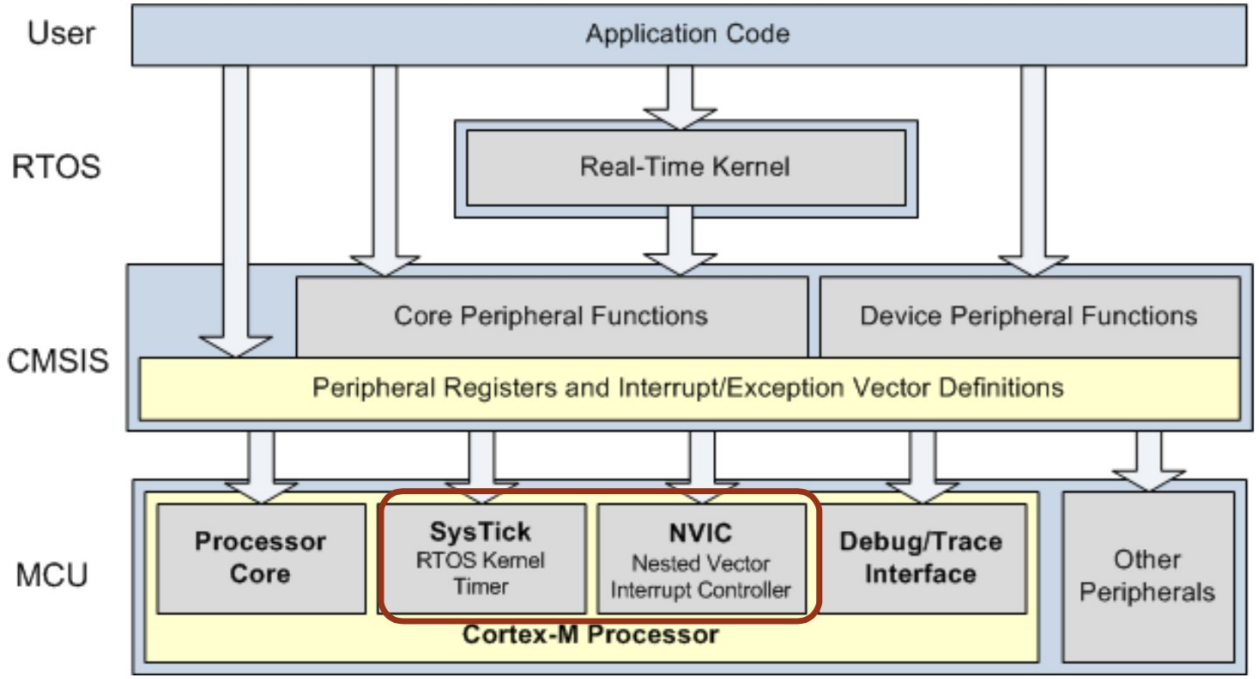
Contrôleur NVIC

- Les registres du NVIC
- NVIC_ISERx/NVIC_ICERx
  - Interrupt Set/Clear Enable Register
  - 1 = Enable interruption / Clear
  - Chaque IRQ a son propre bit d'activation
- NVIC_ISPRx/NVIC_ICPRx
  - Interrupt Set/Clear Pending Register
  - Lecture d'un 1 dans le registre ISPR permet de savoir si l'interruption est suspendue (pending state)
  - En mode écriture = permet de suspendre une interruption ou de l'annuler
- NVIC_IABRx – Interrupt Active Bit Register
  - Permet de savoir si l'interruption est active



Fonctions CMSIS

- Fonctions CMSIS : Contex Microcontroller Software Interface Standard
  - HAL commun au Cortex-M
    - Réutilisation et portabilité du code sur des supports cortex différents
  - Comprend :
    - Core Peripheral Access Layer (fonctions pour accéder aux registres des périphériques)
    - Device Peripheral Access Layer (adresse et driver des périphériques)
    - Access functions for Peripheral (fonctions d'aide à l'accès aux périphériques)



Fonctions CMSIS

Fonctions CMSIS

CMSIS interrupt control function	Description
`void NVIC_SetPriorityGrouping(uint32_t priority_grouping)`	Set the priority grouping
`void NVIC_EnableIRQ(IRQn_t IRQn)`	Enable IRQn
`void NVIC_DisableIRQ(IRQn_t IRQn)`	Disable IRQn
`uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)`	Return true (IRQ-Number) if IRQn is pending
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	Clear IRQn pending status
`uint32_t NVIC_GetActive (IRQn_t IRQn)`	Return the IRQ number of the active interrupt
`void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn
`uint32_t NVIC_GetPriority (IRQn_t IRQn)`	Read priority of IRQn
`void NVIC_SystemReset (void)`	Reset the system

Fonctions CMSIS

- Fonctions CMSIS : exemple

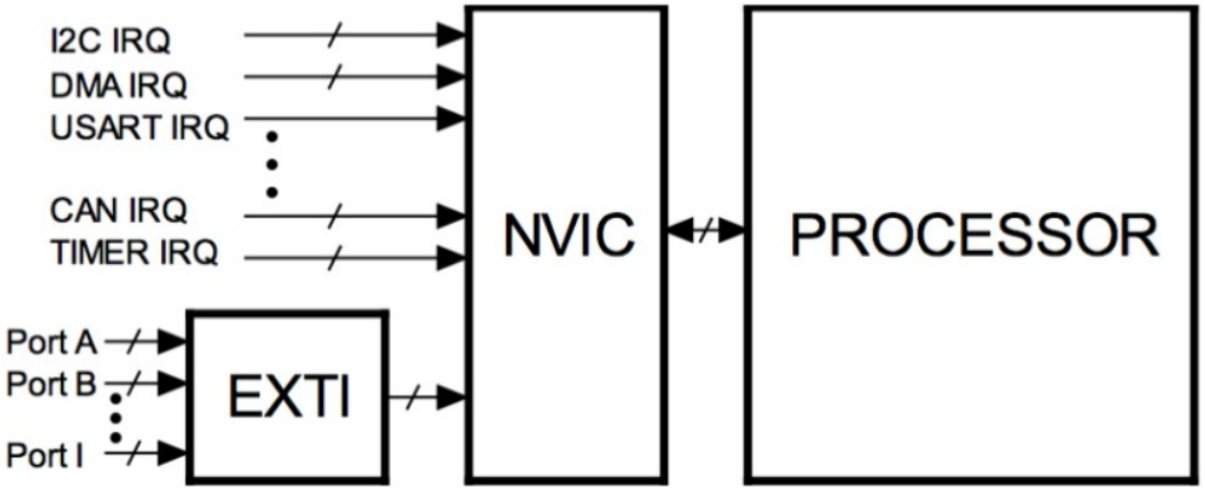
- Bibliothèque stm32f4xx.h définie toutes les variables globales

```
EXTIO_IRQn=6;              // Interruption externe EXTIO
                  // (IRQ #6 voir table des vecteurs)
TIM3_IRQn = 29;            // Interruption TIM3 du timer (IRQ#29)

NVIC_EnableIRQ(EXTIO_IRQn);    //autorise l'interruption EXTIO
NVIC_DisableIRQ(TIM3_IRQn);    //interdit l'interruption du timer TIM3
```

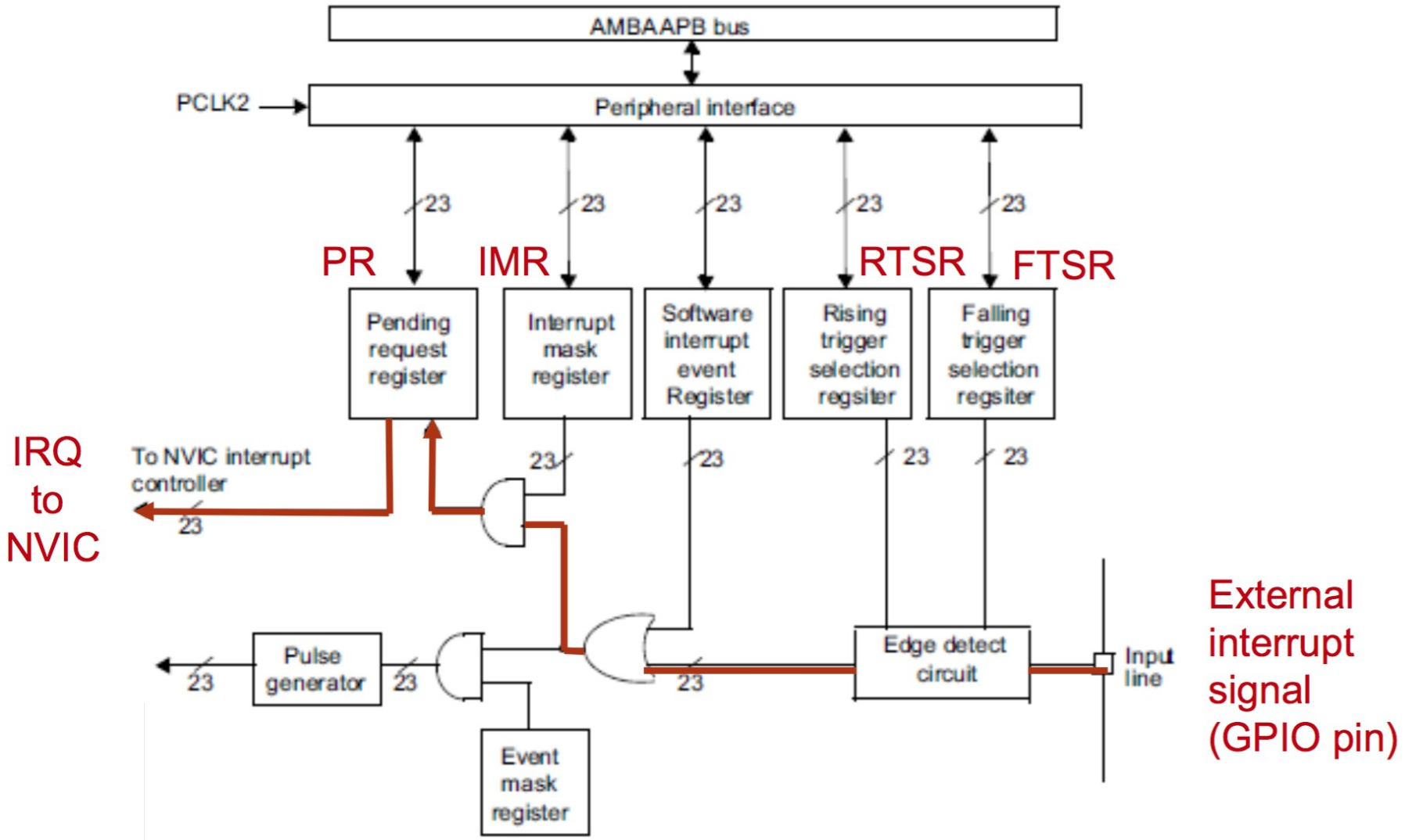

46

Interruptions externes

- Gestion des interruptions externe au niveau architecture
  - EXTI proviennent généralement de lignes externes – connectées à un GPIO
  - 23 détecteurs de front pour synchroniser les évènements et les interruptions délivrés par 240 broches de GPIO et 7 évènements internes

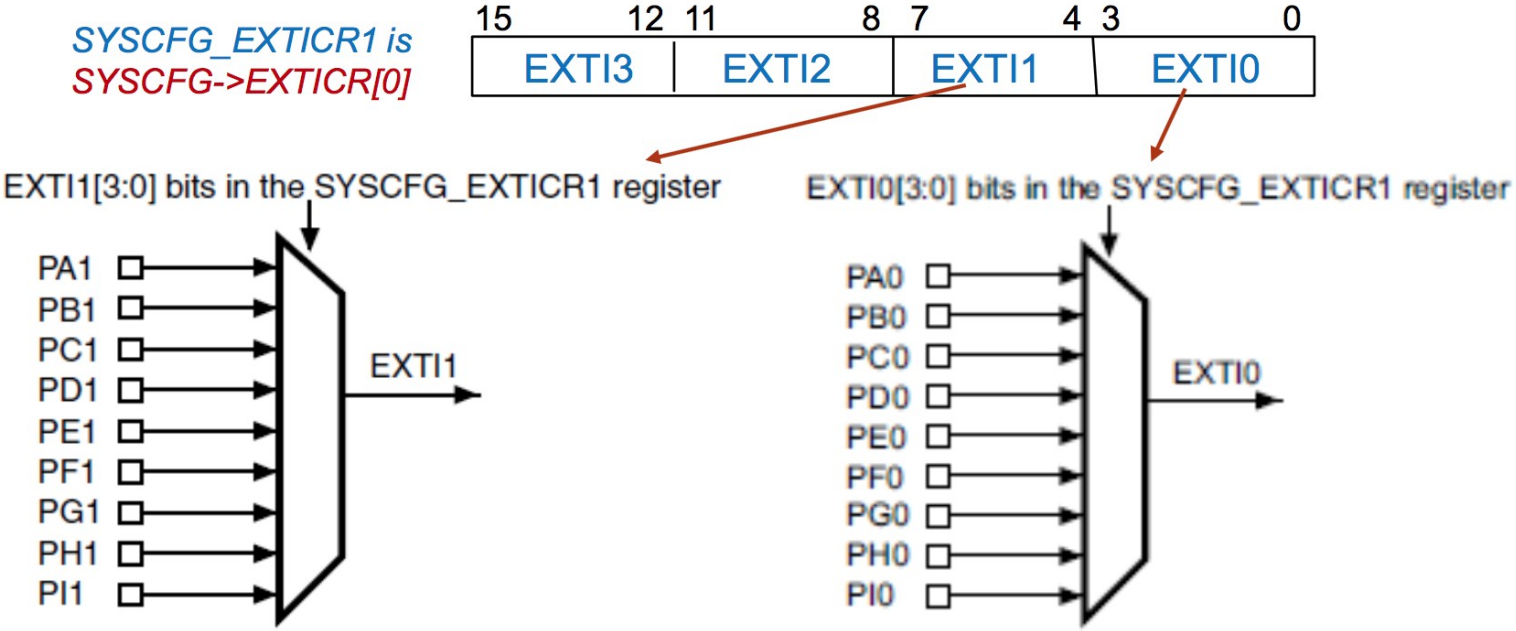


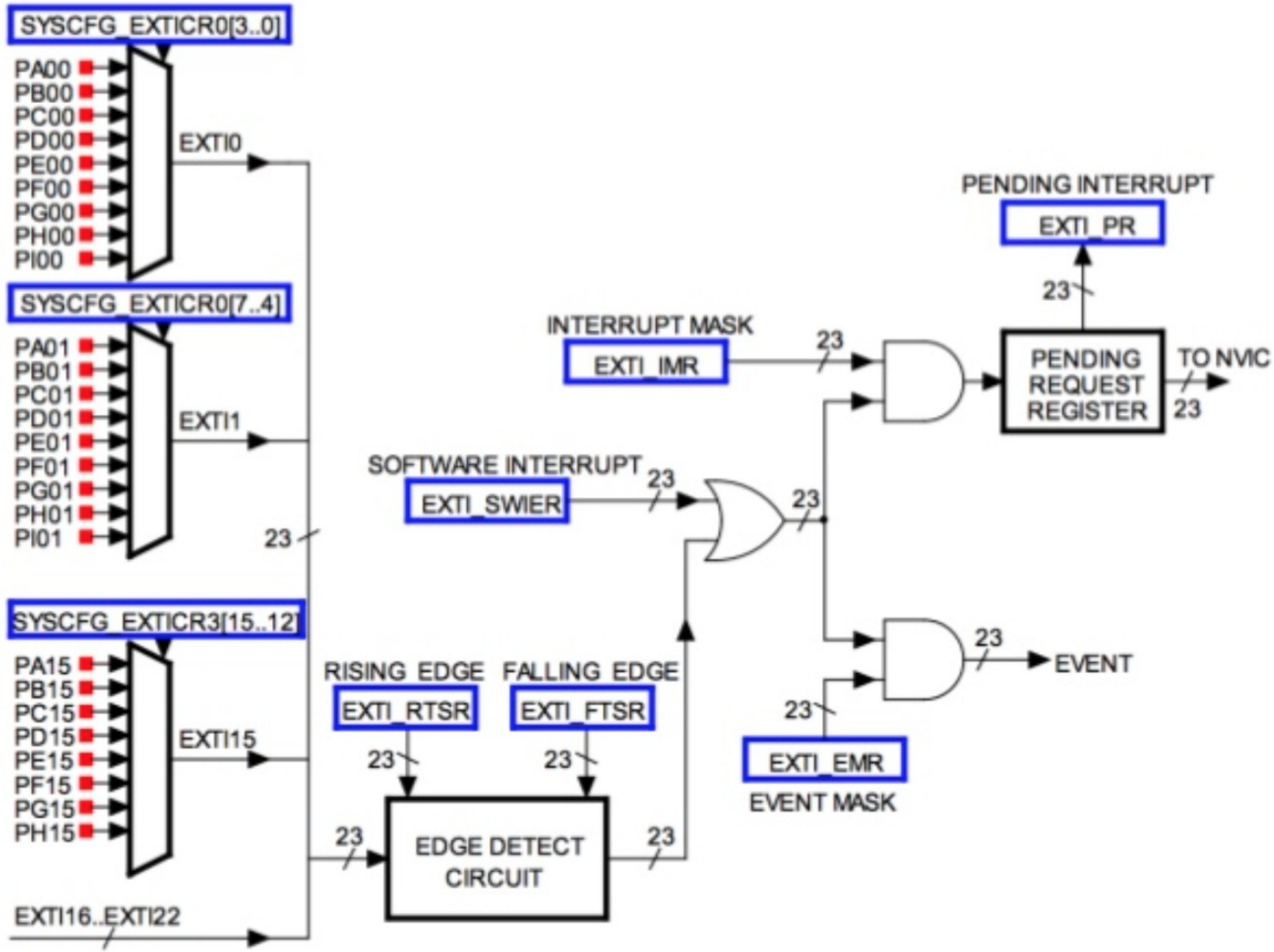
Interruption externe



Interruptions externes

- Multiplexeurs sélectionnent les GPIO comme des lignes d'interruptions externes EXTI0 à EXTI15
- Les entrées des multiplexeurs sont sélectionnées via 4 bits dans le registre EXTICR[k]
  - `EXTIx = 0` pour Pax, `1` pour PBx, `2` pour PCx, etc…
  - `EXTICR[0]` pour EXTI3-EXTI0; `EXTICR[1]` pour EXTI7-EXTI4, etc…




SYSCFG_EXTICR0[3..0]
PA00
PB00
PC00
PD00
PE00
PF00
PG00
PH00
PI00
EXTI0
SYSCFG_EXTICR0[7..4]
PA01
PB01
PC01
PD01
PE01
PF01
PG01
PH01
PI01
EXTI1
23
SYSCFG_EXTICR3[15..12]
PA15
PB15
PC15
PD15
PE15
PF15
PG15
PH15
EXTI15
EXTI16..EXTI22
PENDING INTERRUPT
EXTI_PR
23
INTERRUPT MASK
EXTI_IMR
23
PENDING REQUEST REGISTER
TO NVIC
23
SOFTWARE INTERRUPT
EXTI_SWIER
23
23
RISING_EDGE
EXTI_RTSR
FALLING_EDGE
EXTI_FTSR
23
23
23
EDGE DETECT CIRCUIT
23
23
EVENT
23
EXTI_EMR
EVENT MASK

Interruptions externes

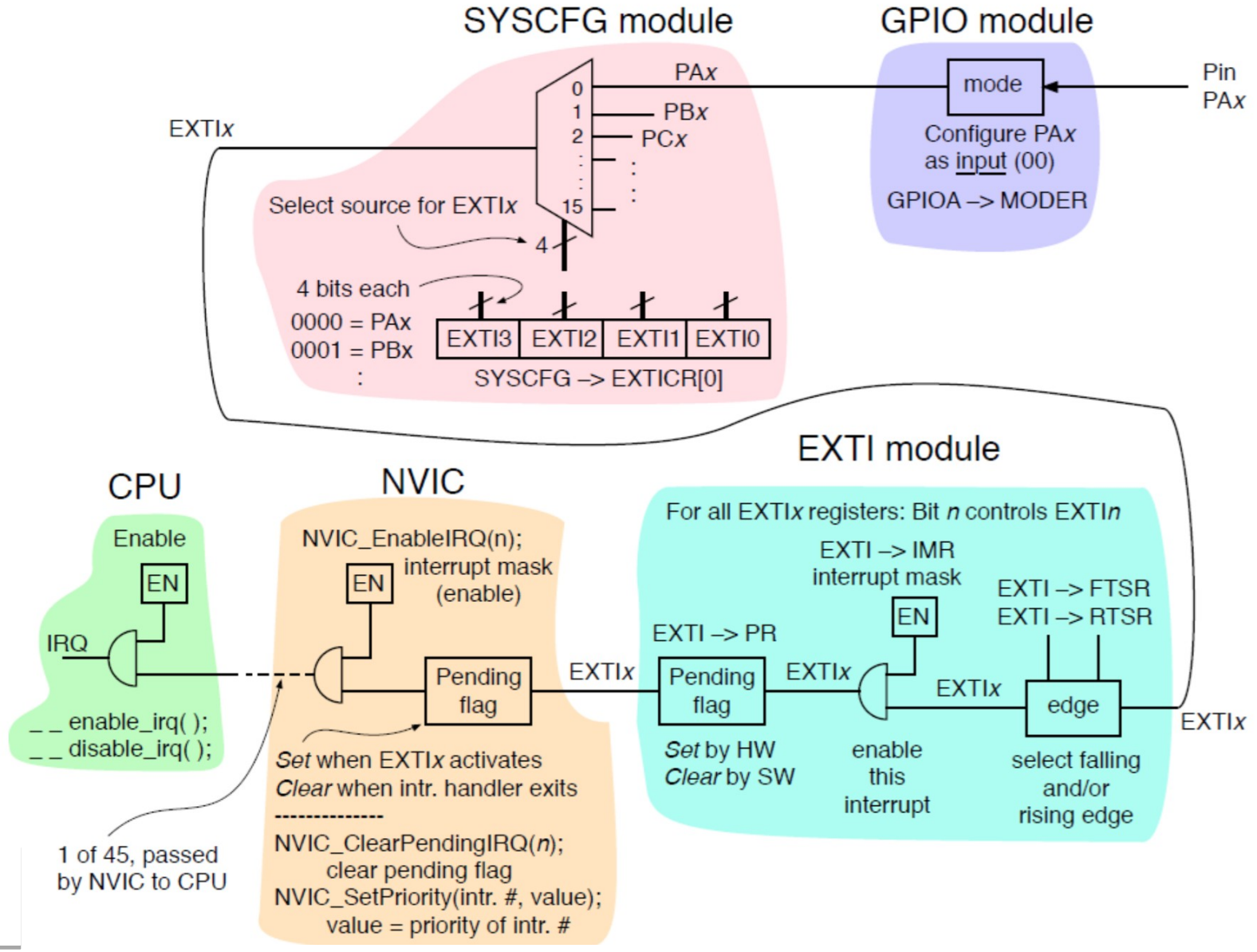
- Les sources sont :
  - 16 interruptions externes
    - EXTI0 à EXTI15
  - 7 événements
    - EXTI16 = PVD output
    - EXTI17 = RTC Alarm Event
    - EXTI18 = USB OTG FS Wakeup event
    - EXTI19 = Ethernet Wakeup event
    - EXTI20 USB OTG HS Wakeup event
    - EXTI21 = RTC Tamper and TimeStamp events
    - EXTI22 = RTC Wakeup event

Interruptions externes

- Les principaux registres EXTI
  - `EXTI_IMR` – interrupt mask register
    - 0 disable les interruptions
    - 1 autorise les interruptions
  - `EXTI_RTSR/FTSR` – rising / falling register
    - 1 enable la détection de front
    - 0 disable
  - `EXTI_PR` – interrupt / event pending register
    - en lecture : un 1 indique que l'interruption a lieu
    - en écriture : un 1 arrête la suspension de l'interruption

IUT CERGY-PONTOISE

Chemin d'une interruptions externes



IUT
CERGY-PONTOISE

Exemples d'application des interruptions

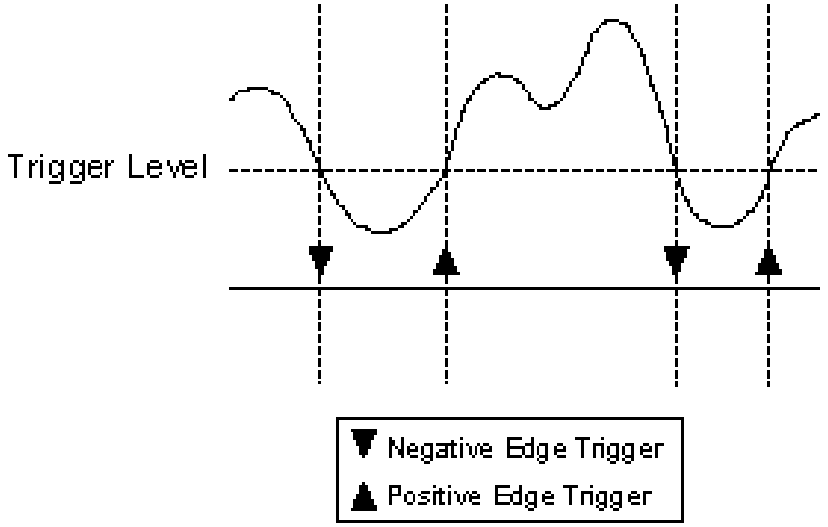
IUT
CERGY-PONTOISE

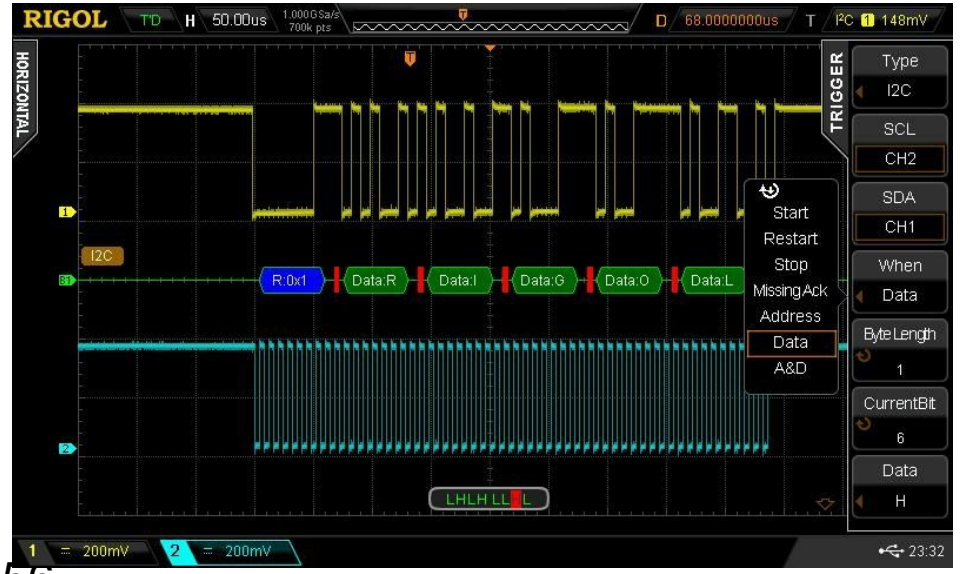
Traitement des interruptions externes sur GPIO

J.Lorandel & O.Romain
MAJ F.Ghaffari & S.Zuckerman – 2020

Interruptions sur GPIO

- Interruption sur GPIO
- Exemple : signal de trigger
  - Synchronisation d'une action sur un signal extérieur
  - Synchronisation sur un front ou un niveau
  - Appui sur un bouton, clavier, etc.
  - Analyse d'un signal, d'un bus, etc.





Exemple de programmation en C

```
#include "STM32F4xx.h"
/*---------------------------------------------------------------------------
      Intialize the GPIO and the external interrupt
*---------------------------------------------------------------------------*/
void Init_Switch(void){

      //Enable the clock for GPIO
      RCC->AHB1ENR| = RCC_AHB1ENR_GPIOAEN;

      //Pull-up pin 0
      GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;

      //Connect the portA pin0 to external interrupt line0
      SYSCFG->EXTICR[0] &= SYSCFG_EXTICR1_EXTI0_PA;

      //Interrupt Mask
      EXTI->IMR |= (1<<0);

      //Falling trigger selection
      EXTI->FTSR |= (1<<0);

      //Enable interrupt
      __enable_irq();

      //Set the priority
      NVIC_SetPriority(EXTI0_IRQn,0);

      //Clear the pending bit
      NVIC_ClearPendingIRQ(EXTI0_IRQn);

      //Enable EXTI0
      NVIC_EnableIRQ(EXTI0_IRQn);}
```

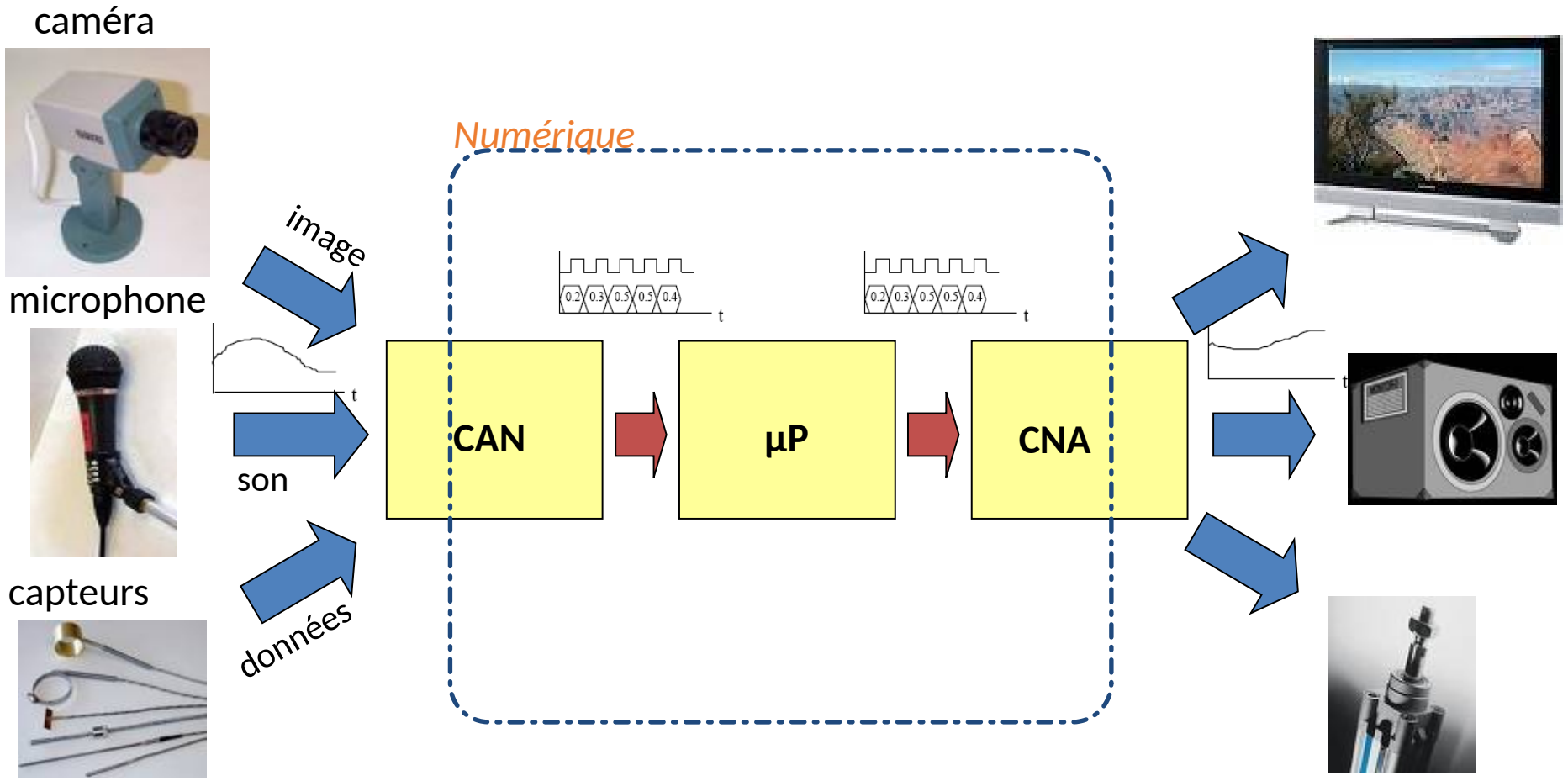
```
/*---------------------------------------------------------------------------
      Interrupt Handler – count button presses
*---------------------------------------------------------------------------*/
void EXTI0_IRQHandler(void) {

      //Make sure the Button is really pressed
      if (!(GPIOA->IDR & (1<<0)) )
      {
            count++;
      }

      //Clear the EXTI pending bits
      NVIC_ClearPendingIRQ(EXTI3_IRQn);
      EXTI->PR|=(1<<0);
}
```

IUT
CERGY-PONTOISE

Traitement du signal

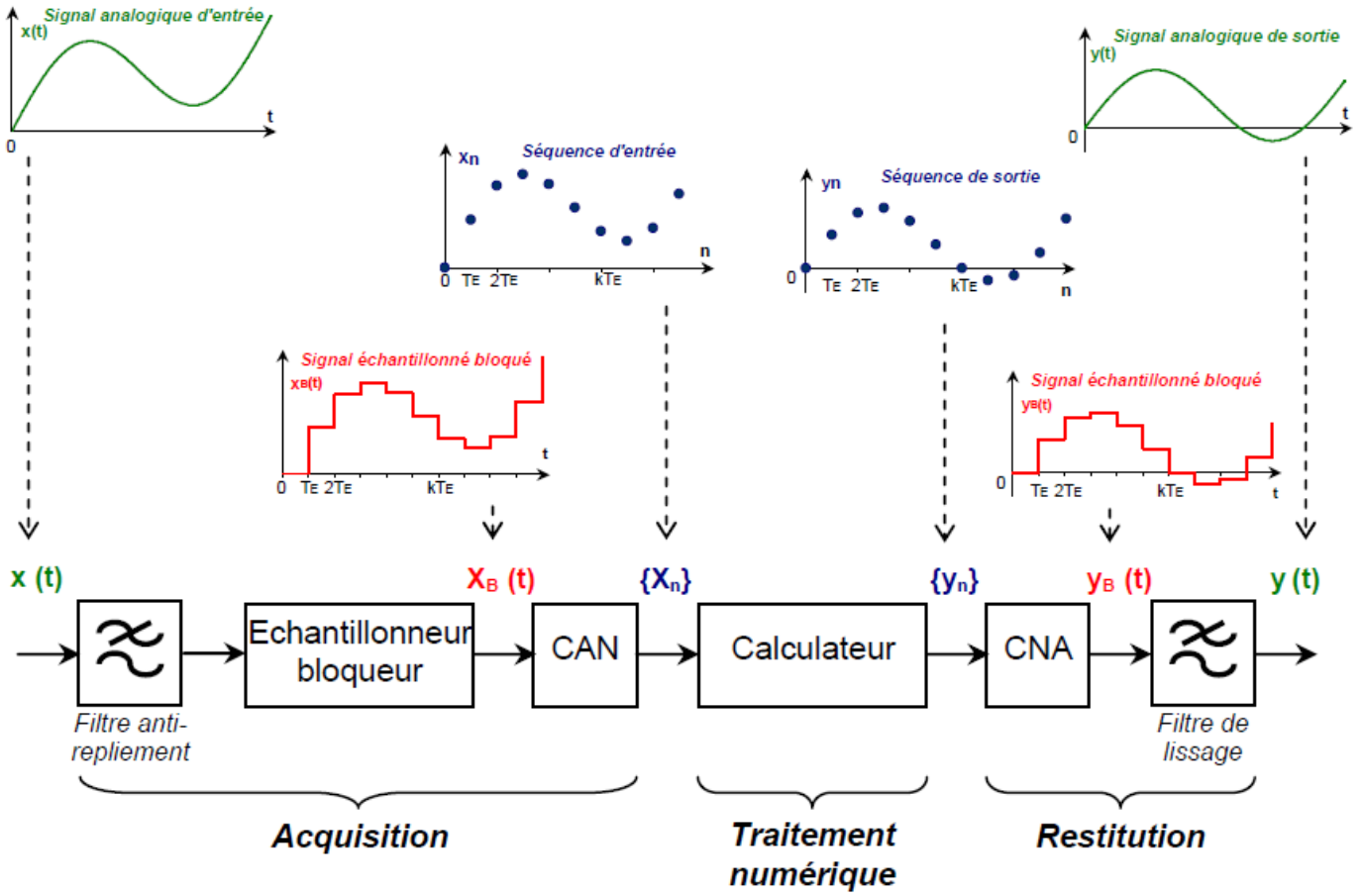
Traitement du signal

- Synoptique d'une chaîne de traitement du signal



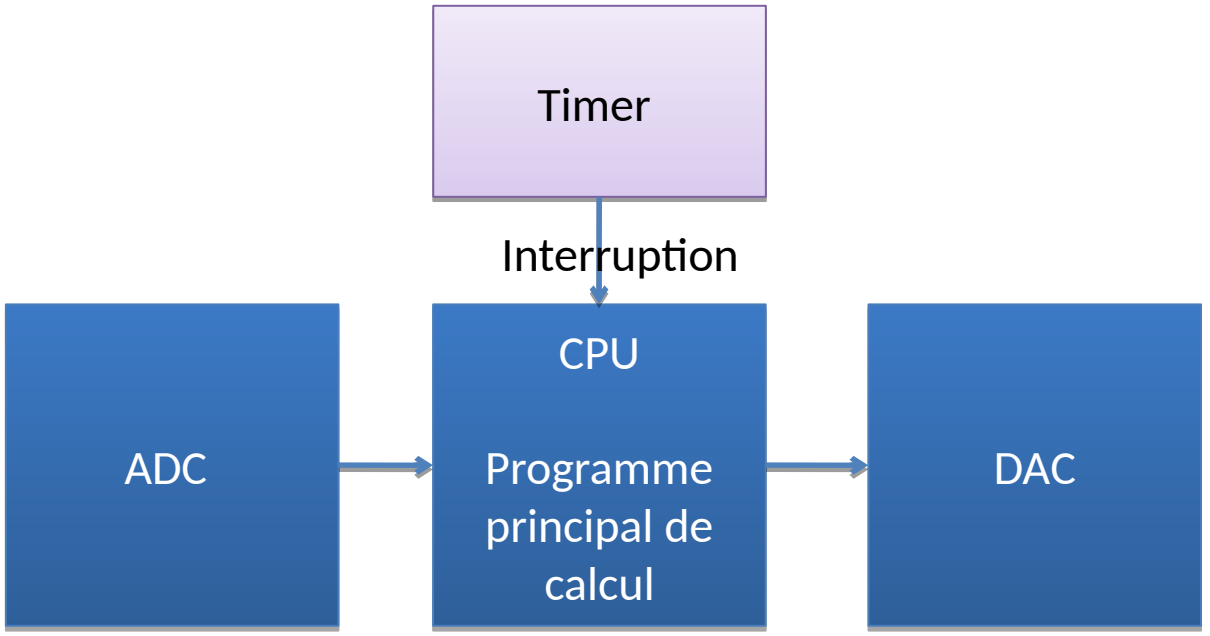
Traitement du signal

- Traitement à intervales temporels réguliers des données issues d'un ADC
  - Echantillonnage temporel des données
  - Nécessite une gestion précise temporelle de l'acquisition



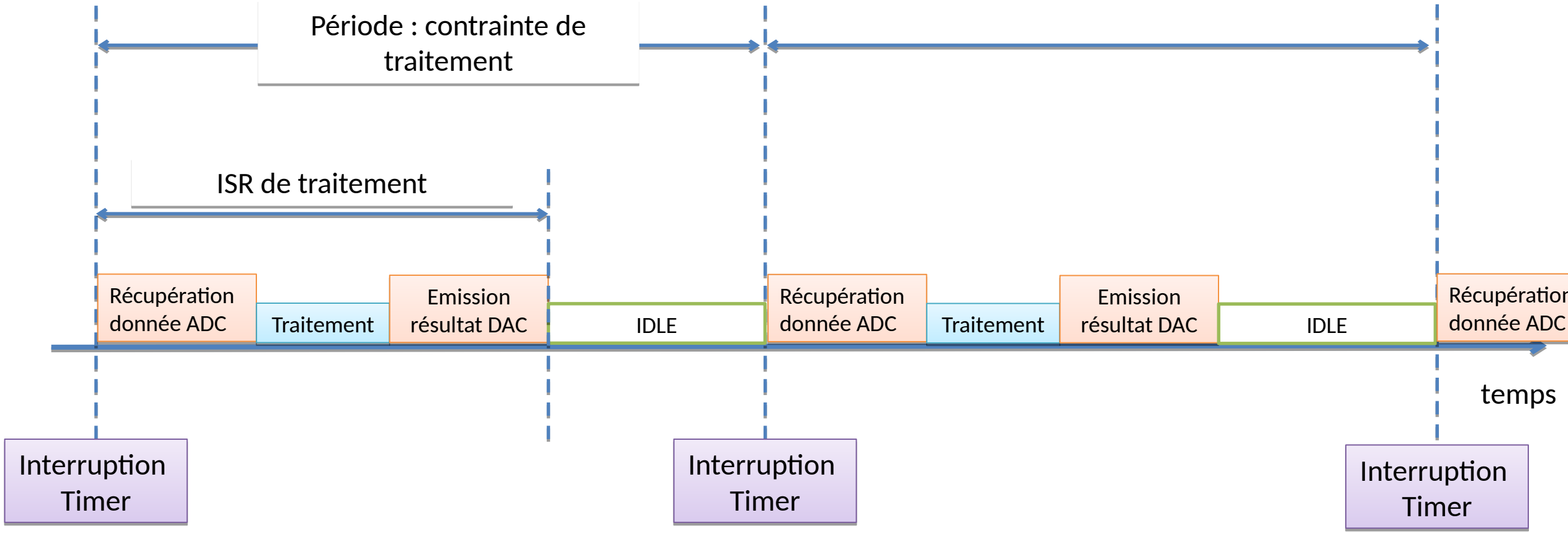
Traitement du signal

- Utilisation d'un périphérique dédié à la gestion du temps : TIMER
  - Le timer va réveiller le CPU à des intervalles de temps réguliers et précis
  - Le CPU récupère le nouvel échantillon et le traite avant de le restituer à un DAC



Traitement du signal

Analyse temporelle



Traitement du signal

- Remarques
  - Le programme principal et le programme et la routine d'interruption partage le « temps CPU »
  - La programmation par interruption nécessite de repenser l'algorithmie. Ce sont des évènements qui déclenchent l'exécution d'un code
  - L'événement est ici périodique : fin de comptage d'un compteur
  - La périodicité des interruptions est programmable via les registres du TIMER

Traitement du signal

- SysTick Timer est un timer sur 24 bits
  - Interruption lorsque le compteur passe à 0 en décomptant
  - Routine d'interruption : SysTick_Handler(void)

- Les bits du registre de contrôle SYST_CSR
  - 0 : autorise
  - 1 : autorise les interruptions
  - 2 : source de l'horloge
    - FCLK = horloge interne
    - STCLK = horloge externe

Bits	Name	Function
[2]	CLKSOURCE	Indicates the clock source: 0 = external clock 1 = processor clock.
[1]	TICKINT	Enables SysTick exception request: 0 = counting down to zero does not assert the SysTick exception request 1 = counting down to zero asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0 = counter disabled 1 = counter enabled.

IUT
CERGY-PONTOISE

STM Cube

Exemple de configuration des GPIOs

- Démonstration de la configuration des GPIOs sous STMCubeMx

1. Configuration des périphériques
2. Génération du code
3. Importation sous keil
4. Execution du code

Interruptions sur GPIO

- Interruption sur GPIO avec l'environnement mbed

- Exemple : déclenchement d'une action à partir de l'appui d'un bouton

```
#include "mbed.h"

InterruptIn mybutton(USER_BUTTON);
DigitalOut myled(LED1);

float delay = 1.0; // 1 sec

void pressed()
{
    if (delay == 1.0)
        delay = 0.2; // 200 ms
    else
        delay = 1.0; // 1 sec
}

int main()
{
    mybutton.fall(&pressed);
    while (1) {
        myled = !myled;
        wait(delay);
    }
}
```