NESTED VECTOR INTERRUPT CONTROLLER - NVIC

1. Présentation

Les interruptions sont des événements généralement générés par le matériel (par exemple, des périphériques ou des broches d'entrée externes) qui provoquent des changements dans la séquence du programme (par exemple, pour fournir un service à un périphérique). Quand un périphérique matériel a besoin d'un service de la part du processeur, typiquement la séquence suivante doit se produise :

1. Le périphérique revendique une demande d'interruption au processeur (en positionnant son indicateur).
2. Le processeur suspend la tâche en cours.
3. Le processeur exécuté la routine d'interruption (ISR : Interrupt Service Routine) pour servir le périphérique, efface l'indicateur de demande d'interruption, s'il est nécessaire.
4. Le processeur reprend la tâche précédemment suspendue.

Dans la littérature, interruption désigne habituellement un signal provenant d'un périphérique et une exception désigne habituellement une faute survenant lors de l'exécution d'une instruction. Cependant, comme une interruption survient de manière exceptionnelle et comme les exceptions interrompent l'exécution en cours, les deux termes sont souvent mélangés. En terminologie ARM, une interruption est un type d'exception.

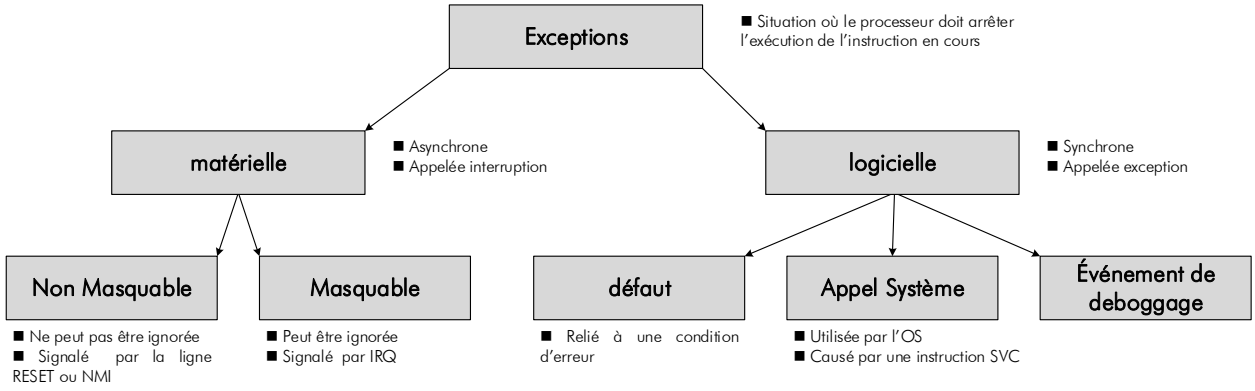


Figure 1

Nested Vector Interrupt Controller est une unité standard au sein du noyau de Cortex. Cela signifie que tous les microcontrôleurs basés Cortex-auront la même structure d'interruption, indépendamment du fabricant. Ainsi le code d'application et les systèmes d'exploitation peuvent être facilement portés d'un microcontrôleur. Le NVIC est également conçu pour avoir une très faible latence d'interruption. Cette latence d'interruption est également déterministe, avec plusieurs fonctions de gestion d'interruption avancées qui prennent en charge les applications en temps réel. Comme son nom l'indique, le NVIC est conçu pour supporter les interruptions imbriquées et sur le STM32 il y a 16 niveaux de priorité. La structure d'interruption NVIC est conçu pour être programmé entièrement en «C» et n'a pas besoin de macros Assembleur ou directives spéciales.

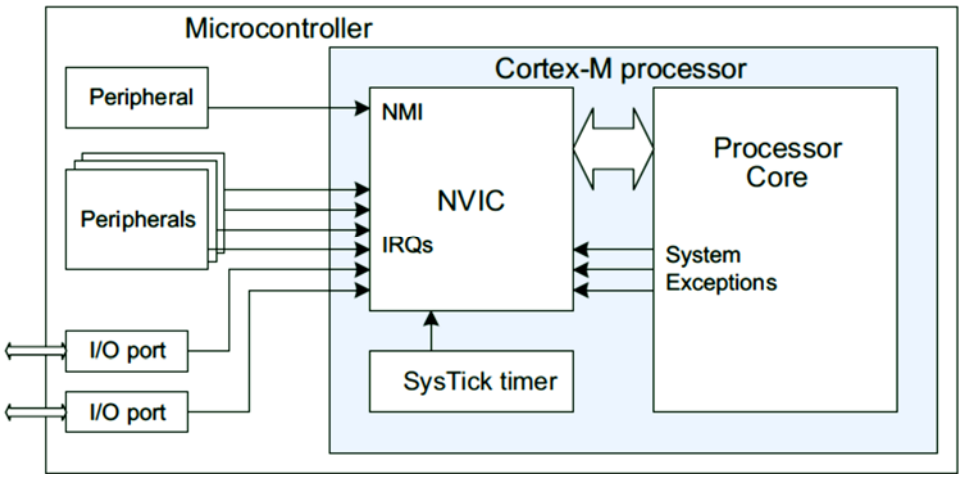


Figure 2

2. Vecteur d'interruptions et des exceptions

Les processeurs Cortex-M Fournissent une architecture d'exception riche en fonctionnalités qui prend en charge un certain nombre d'exceptions du système et des interruptions externes. Les exceptions numérotées 1 à 15 sont des exceptions système. La plupart des exceptions, y compris toutes les interruptions, ont des priorités programmables, et quelques exceptions du système ont des priorités fixes.

Différents microcontrôleurs Cortex-M3 ou M4 peuvent avoir des nombres différents de sources d'interruption (de 1 à 240) et des nombres différents de niveaux de priorité. Les microcontrôleurs STM32f2xx possèdent 81 sources d'interruptions sans compter les exceptions système. Le tableau suivant présente la table du vecteur d'interruptions.

Liste des 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

Liste des interruptions

Position	Priority	Type of priority	Acronym	Description	Address
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000_0044
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000_0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	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_Stream0	DMA1 Stream0 global interrupt	0x0000_006C
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000_0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000_0074
14	21	settable	DMA1_Stream3	DMA1 Stream3 global interrupt	0x0000_0078
15	22	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000_007C
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000_0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000_0084
18	25	settable	ADC	ADC1, ADC2 and ADC3 global interrupts	0x0000_0088
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000_008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000_0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000_0094
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000_00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000_00A4

26	33	settable	TIM1_TRG_COM_ TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTC_Alarm	RTC Alarms (A and B) through EXTI line interrupt	0x0000_00E4
42	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000_00E8
43	50	settable	TIM8_BRK_TIM12	TIM8 Break interrupt and TIM12 global interrupt	0x0000_00EC
44	51	settable	TIM8_UP_TIM13	TIM8 Update interrupt and TIM13 global interrupt	0x0000_00F0
45	52	settable	TIM8_TRG_COM_ TIM14	TIM8 Trigger and Commutation interrupts and TIM14 global interrupt	0x0000_00F4
46	53	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000_00F8
47	54	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000_00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000_0100
49	56	settable	SDIO	SDIO global interrupt	0x0000_0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C
52	59	settable	UART4	UART4 global interrupt	0x0000_0110
53	60	settable	UART5	UART5 global interrupt	0x0000_0114
54	61	settable	TIM6_DAC	TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C
56	63	settable	DMA2_Stream0	DMA2 Stream0 global interrupt	0x0000_0120
57	64	settable	DMA2_Stream1	DMA2 Stream1 global interrupt	0x0000_0124
58	65	settable	DMA2_Stream2	DMA2 Stream2 global interrupt	0x0000_0128
59	66	settable	DMA2_Stream3	DMA2 Stream3 global interrupt	0x0000_012C
60	67	settable	DMA2_Stream4	DMA2 Stream4 global interrupt	0x0000_0130

61	68	settable	ETH	Ethernet global interrupt	0x0000_0134
62	69	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	0x0000_0138
63	70	settable	CAN2_TX	CAN2 TX interrupts	0x0000_013C
64	71	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000_0140
65	72	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000_0144
66	73	settable	CAN2_SCE	CAN2 SCE interrupt	0x0000_0148
67	74	settable	OTG_FS	USB On The Go FS global interrupt	0x0000_014C
68	75	settable	DMA2_Stream5	DMA2 Stream5 global interrupt	0x0000_0150
69	76	settable	DMA2_Stream6	DMA2 Stream6 global interrupt	0x0000_0154
70	77	settable	DMA2_Stream7	DMA2 Stream7 global interrupt	0x0000_0158
71	78	settable	USART6	USART6 global interrupt	0x0000_015C
72	79	settable	I2C3_EV	I^2C3 event interrupt	0x0000_0160
73	80	settable	I2C3_ER	I^2C3 error interrupt	0x0000_0164
74	81	settable	OTG_HS_EP1_OU T	USB On The Go HS End Point 1 Out global interrupt	0x0000_0168
75	82	settable	OTG_HS_EP1_IN	USB On The Go HS End Point 1 In global interrupt	0x0000_016C
76	83	settable	OTG_HS_WKUP	USB On The Go HS Wakeup through EXTI interrupt	0x0000_0170
77	84	settable	OTG_HS	USB On The Go HS global interrupt	0x0000_0174
78	85	settable	DCMI	DCMI global interrupt	0x0000_0178
79	86	settable	CRYP	CRYP crypto global interrupt	0x0000_017C
80	87	settable	HASH_RNG	Hash and Rng global interrupt	0x0000_0180

3. Priorité des interruptions

Les processeurs Cortex-M3 et Cortex-M4 supporte trois niveaux de priorité fixe et jusqu'à 256 niveaux de priorité programmable (avec un maximum de 128 niveaux préemptible). Le nombre réel de niveaux de priorité programmables disponibles est décidée par les concepteurs du circuit.

En effet, ayant un grand nombre de niveaux de priorité peut augmenter la complexité du contrôleur NVIC et augmenter sa consommation d'énergie. Dans la plupart des cas, les applications ne nécessitent pas un nombre important de niveaux de priorité programmables. Par conséquent, les concepteurs de puces doivent personnaliser leur conception du processeur en fonction du nombre de niveaux de priorité dans les applications ciblées.

Le niveau de priorité des interruptions est contrôlé par un registre de niveau de priorité, avec une largeur 8 bits. Généralement ce niveau de priorité est codé sur 3 ou 4 bits, les bits de faible poids sont toujours à zéro.

Niveau de priorité sur 4 bits							
b7	b6	b5	b4	b3	b1	b1	b0
Implémenté				Non implémenté			

Le niveau de priorité est divisé en deux parties : groupe de priorité et sous-priorité. Le groupe de priorité est appelé aussi priorité préemptible. Le groupe de priorité fixe la priorité des périphériques ; c-à-d un

périphérique peut être interrompu par un autre plus prioritaire. La sous priorité intervient lorsque deux périphériques ou plus de même priorité demandent un service, le périphérique qui a la petite valeur de sous priorité est servi le premier.

Le niveau de priorité de l'STM32F2xx est code sur 4 bits.

Number CMSIS	PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
		Binary point	Group priority bits	Subpriority bits	Group priorities	Sub priorities
–	0b000	0bxxxx.xxx.y	[7:1]	[0]	16	None
–	0b001	0bxxxx.xx.yy	[7:2]	[1:0]	16	None
–	0b010	0bxxxx.x.yyy	[7:3]	[2:0]	16	None
4	0b011	0bxxxx.yyyy	[7:4]	[3:0]	16	None
3	0b100	0bxxx.yyyyy	[7:5]	[4:0]	8	2
2	0b101	0bxx.yyyyyy	[7:6]	[5:0]	4	4
1	0b110	0bx.yyyyyyy	[7]	[6:0]	2	8
0	0b111	0b.yyyyyyyy	None	[7:0]	None	16

Le groupe de priorité est définie dans le champ « PRIGROUP » du registre SCB_AIRCR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)															
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
ENDIANESS	Reserved				PRIGROUP			Reserved					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
r					rw	rw	rw						w	w	w

Dans le fichier cmsis.h, quatre groupes sont définie de la manière suivante :

```
#define NVIC_PriorityGroup_0         ((uint32_t)0x700) /*!< 0 bits for pre-emption priority
                                                            4 bits for subpriority */
#define NVIC_PriorityGroup_1         ((uint32_t)0x600) /*!< 1 bits for pre-emption priority
                                                            3 bits for subpriority */
#define NVIC_PriorityGroup_2         ((uint32_t)0x500) /*!< 2 bits for pre-emption priority
                                                            2 bits for subpriority */
#define NVIC_PriorityGroup_3         ((uint32_t)0x400) /*!< 3 bits for pre-emption priority
                                                            1 bits for subpriority */
#define NVIC_PriorityGroup_4         ((uint32_t)0x300) /*!< 4 bits for pre-emption priority
                                                            0 bits for subpriority */
```

La fonction suivante initialise le champ PRIGROUP

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
{ /* Set the PRIGROUP[10:8] bits according to NVIC_PriorityGroup value */
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
}
```

Cette fonction est appelée avec le paramètre *NVIC_PriorityGroup_x (0 .. 4)*

Exemple : pour définir 8 niveaux de priorité (préemption) et 2 niveaux de sous priorité, il suffit d'appeler la fonction suivante :

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
```

4. Registre de l'NVIC

4.1. Etats des exceptions

Une interruption peut être dans l'un des états suivants :

- Inactive : elle n'est pas active (n'est pas en cours d'exécution) et n'est en attente d'exécution.
- En attente (Pending) : l'exception est en attente pour être servi par le processeur.
- Active : l'exception est servie par le processeur, mais elle n'a terminé.
- Active et en Attente (Active and Pending) : l'exception est active et une exception de même source est en attente.

Donc une interruption peut être :

- Désactivé (Disable) ou activé (Enable)
- En attente d'exécution ou non.
- Active, c-à-d servie par le processeur ou non.

La figure suivante illustre la gestion d'une demande d'interruption

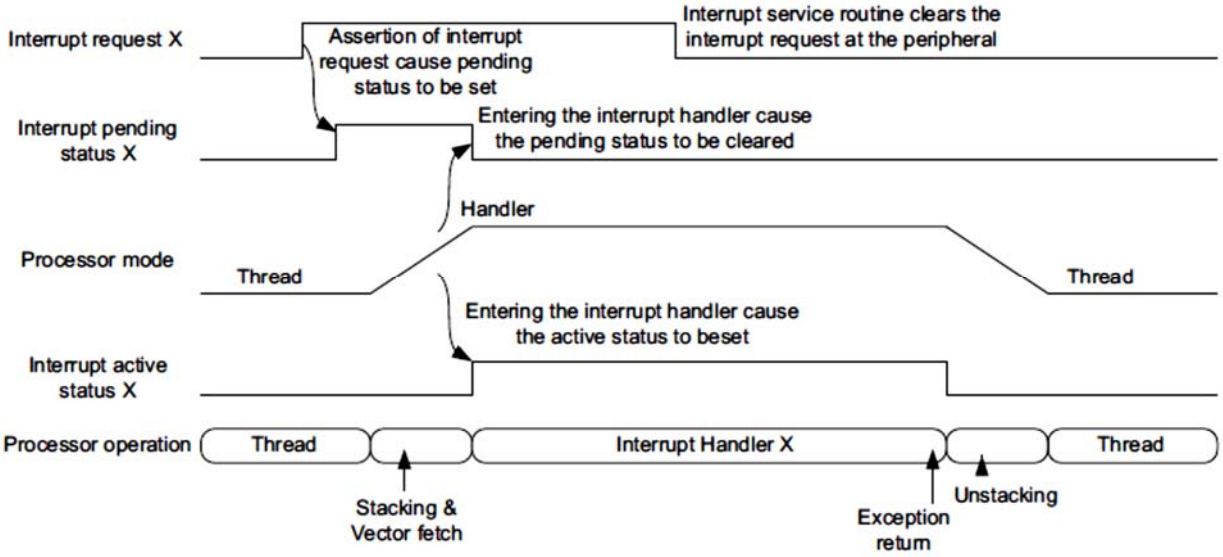


Figure 3

4.2. Gestion des interruptions

Les NVIC est conçu pour supporter des périphériques qui génèrent des demandes d'interruptions pulsés ou à niveau. Dans tous les cas, le NVIC mémorise les demandes dans des indicateurs nommés « Pending flag ». Pour les demandes à niveau, cette demande reste maintenu jusqu'à où elle sera désactivé. Lorsque le processeur entre dans l'ISR l'indicateur « pending bit » est automatiquement remis à zéro en positionnant le bit « Active bit ». Si l'indicateur de demande d'interruption n'est pas désactivé avant le retour de l'ISR, le processeur ré-entre à nouveau dans l'ISR.

En mode impulsion, le NVIC continu à surveiller le signal, si une impulsion arrive pendant l'exécution de l'ISR, l'interruption passe à l'état d'attente et active (le deux bits Pending bit et Active bit à 1). A la sortie de l'ISR, l'interruption devient à l'état d'attente et le processeur ré-entre immédiatement dans l'ISR.

Le deux cas sont illustrés par les figures suivantes :

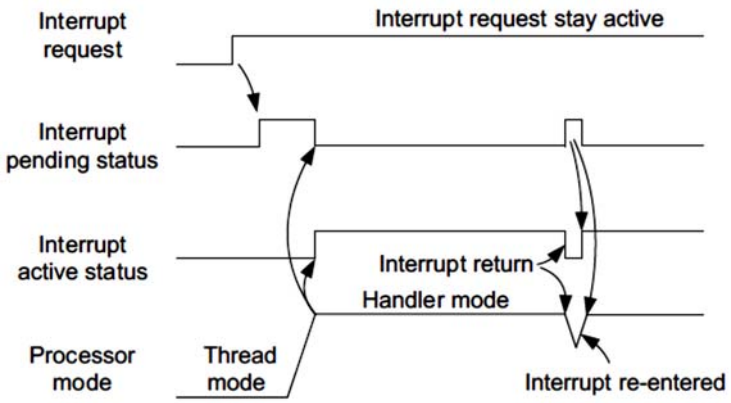


Figure 4

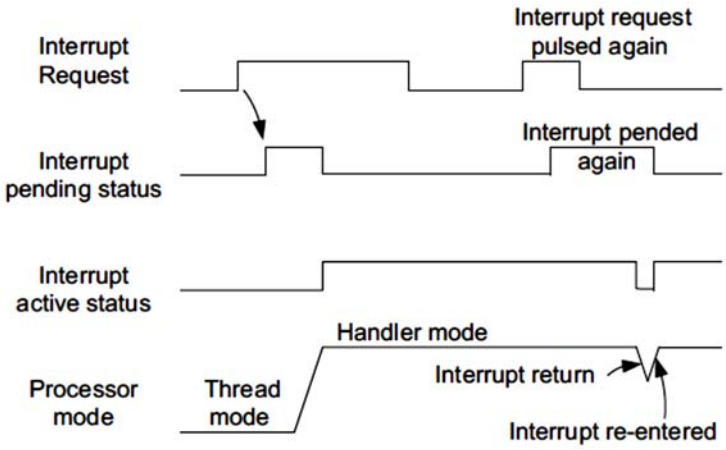


Figure 5

4.3. Registres de l'NVIC

Pour chaque entrée d'interruption, le contrôleur NVIC fournit :

- Un bit de validation d'interruption dans le registre ISERx : Interrupt Set Enable Register
- Un bit de désactivation d'interruption dans le registre ICERx : Interrupt Clear Enable Register
- Un bit de mise à 1 de l'inducteur « Pending flag » de demande d'interruption dans le registre ISPRx : Interrupt Set Pending Regisrer. Ce bit est normalement mis à 1 par le matériel.
- Un bit de remise à zéro de l'indicateur « Pending flag » dans le registre ICPRx : Interrupt Clear Pending Regisrer. Ce bit est remis automatiquement à zéro à l'entrée de l'ISR.
- Un bit « Active bit » dans le registre IABRx : Interrupt Active Bit Register. Ce bit à lecture seule est mis à 1 à l'entrée de l'ISR et reste à cet état jusqu'à la sortie de l'ISR.
- Une autre manière de mette à 1 l'indicateur « Pending flag », est d'écrire le numéro d'interruption dans le champ INTID du registre STIR : Software trigger interrupt register.
- Un champ IP de 8 bits dans le registre IPRx : Interrupt Priority Register pour fixer le numéro de priorité du périphérique.

Le tableau suivant présente les différents registres de l'NVIC

Nom des registres	Nombre de registres	Type d'accès	Description
NVIC_ISER0 - NVIC_ISER2	03	RW	Interrupt Set-Enable Registers
NVIC_ICER0 - NVIC_ICER2	03	RW	Interrupt Clear-Enable Registers
NVIC_ISPR0 - NVIC_ISPR2	03	RW	Interrupt Set-Pending Registers
NVIC_ICPR0 - NVIC_ICPR2	03	RW	Interrupt Clear-Pending Registers
NVIC_IABR0 - NVIC_IABR2	03	R	Interrupt Active Bit Register
NVIC_IPR0 - NVIC_IPR20	21	RW	Interrupt Priority Register
NVIC_STIR	01	W	Software trigger interrupt register

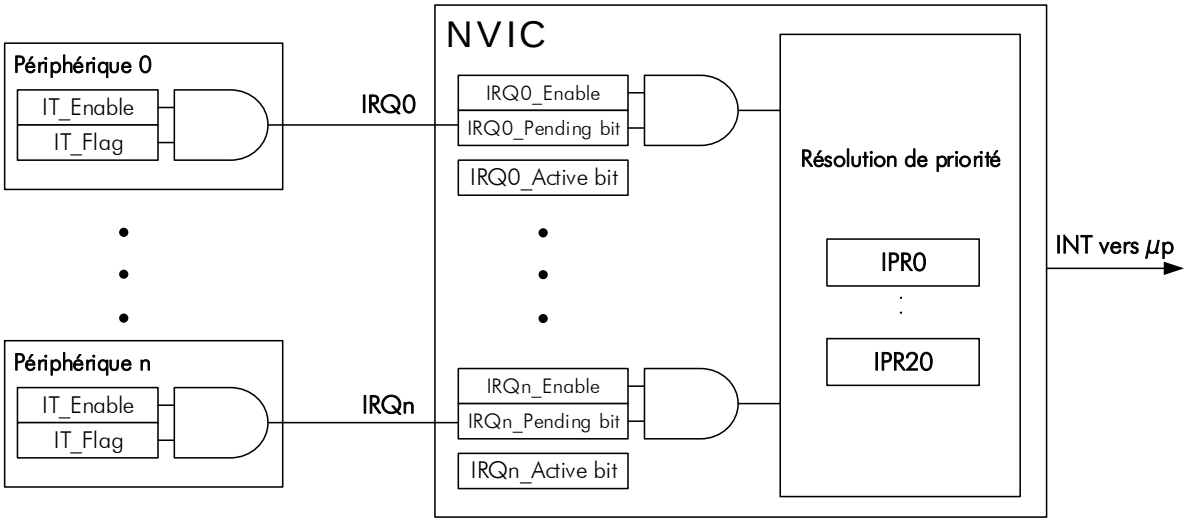


Figure 6

4.3.1. Interrupt set-enable registers : NVIC_ISERx (x : 0 .. 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Bits 31:0 **SETENA[31:0]**: Interrupt set-enable bits.

Write:
0: No effect
1: Enable interrupt
Read:
0: Interrupt disabled
1: Interrupt enabled.

4.3.2. Interrupt clear-enable registers : NVIC_ICERx (x : 0 .. 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Bits 31:0 **CLRENA[31:0]**: Interrupt clear-enable bits.

Write:
0: No effect
1: Disable interrupt
Read:
0: Interrupt disabled
1: Interrupt enabled

4.3.3. Interrupt set-pending registers : NVIC_ISPRx (x : 0 .. 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPEND[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETPEND[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Bits 31:0 **SETPEND[31:0]**: Interrupt set-pending bits
Write:
0: No effect
1: Changes interrupt state to pending
Read:
0: Interrupt is not pending
1: Interrupt is pending

4.3.4. Interrupt clear-pending registers : NVIC_ICPRx (x : 0 .. 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRPEND[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRPEND[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:0 **CLRPEND[31:0]**: Interrupt clear-pending bits
Write:
0: No effect
1: Removes the pending state of an interrupt
Read:
0: Interrupt is not pending
1: Interrupt is pending

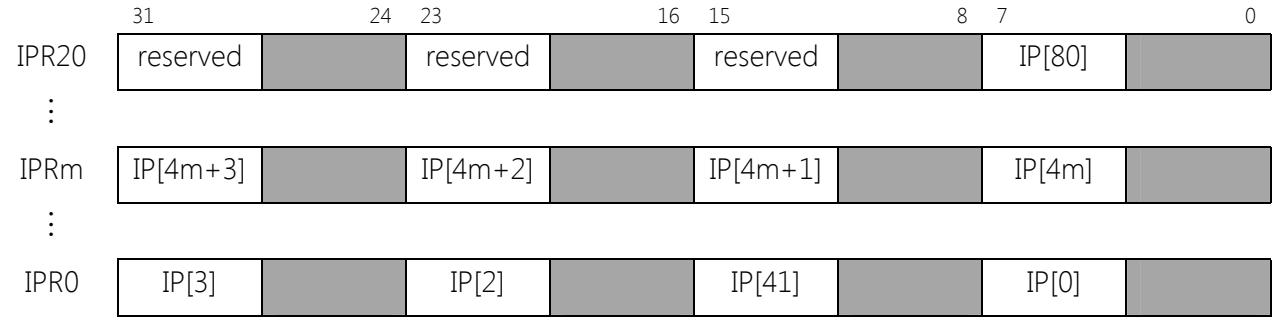
4.3.5. Interrupt active bit registers : NVIC_IABRx (x : 0 .. 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ACTIVE[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACTIVE[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **ACTIVE[31:0]**: Interrupt active flags
0: Interrupt not active
1: Interrupt active

4.3.6. Interrupt priority registers : NVIC_IPRx (x : 0 .. 20)

Dans le microcontrôleur STM32, il y a 21 registres priorités. Chaque registre de priorité est divisé en quatre champs de huit bits, chaque champ étant affecté à un vecteur d'interruption. Le STM32 utilise seulement la moitié de ce champ pour mettre en œuvre 16 niveaux de priorité, de 0 à 15 avec le niveau zéro le plus prioritaire. Il est également possible de formater le champ de priorité en groupes et sous-groupes de priorité, en programmant le champ PRIGROUP dans le registre SCB-> AIRCR.



4.3.7. Software trigger interrupt register : NVIC_STIR

Ecrire le numéro de l'interruption dans le champ INTID, génère une interruption logicielle (SGI). Le numéro doit être compris entre 0 et 239, soit de 0 à 80 dans le cas de l'

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								INTID[7:0]							
reserved								w	w	w	w	w	w	w	w

5. Programmation de l'NVIC

Dans le fichier core_cm3.h est définie les registres de l'NVIC dans la structure NVIC_Type

```
typedef struct
{
  __IOM uint32_t ISER[8U];            /* Interrupt Set Enable Register */
        uint32_t RESERVED0[24U];
  __IOM uint32_t ICER[8U];            /* Interrupt Clear Enable Register */
        uint32_t RSERVED1[24U];
  __IOM uint32_t ISPR[8U];            /* Interrupt Set Pending Register */
        uint32_t RESERVED2[24U];
  __IOM uint32_t ICPR[8U];            /* Interrupt Clear Pending Register */
        uint32_t RESERVED3[24U];
  __IOM uint32_t IABR[8U];            /* Interrupt Active bit Register */
        uint32_t RESERVED4[56U];
  __IOM uint8_t  IP[240U];     /* Interrupt Priority Register (8Bit wide) */
        uint32_t RESERVED5[644U];
  __OM  uint32_t STIR;           /* Software Trigger Interrupt Register */
}  NVIC_Type;
```

Les informations nécessaires à la configuration :

```
typedef struct {
 uint8_t NVIC_IRQChannel;
 uint8_t NVIC_IRQChannelPreemptionPriority;
 uint8_t NVIC_IRQChannelSubPriority;
 FunctionalState NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

NVIC_IRQChannel : Spécifie l'entrée d'interruption, ses paramètres sont définis dans l'énumération IRQn du fichier STM32F2xx.h.

NVIC_IRQChannelPreemptionPriority et NVIC_IRQChannelSubPriority : fixent la priorité de la ligne d'interruption. Le tableau suivant présente les valeurs possibles en fonction du paramètre *NVIC_PriorityGroup*.

NVIC_IRQChannelCmd : ce paramètre spécifie si l'entrée d'interruption définie dans le paramètre *NVIC_IRQChannel* est activée ou désactive. Il prend les paramètres ENABLE ou DISABLE.

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority
NVIC_PriorityGroup_0	0	0-15
NVIC_PriorityGroup_1	0-1	0-7
NVIC_PriorityGroup_2	0-3	0-3
NVIC_PriorityGroup_3	0-7	0-1
NVIC_PriorityGroup_4	0-15	0

La fonction NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct); permet de faire les initialisations nécessaires pour configurer la ligne IRQn.

Exemple : configuer le timer 10 en mode interruption.

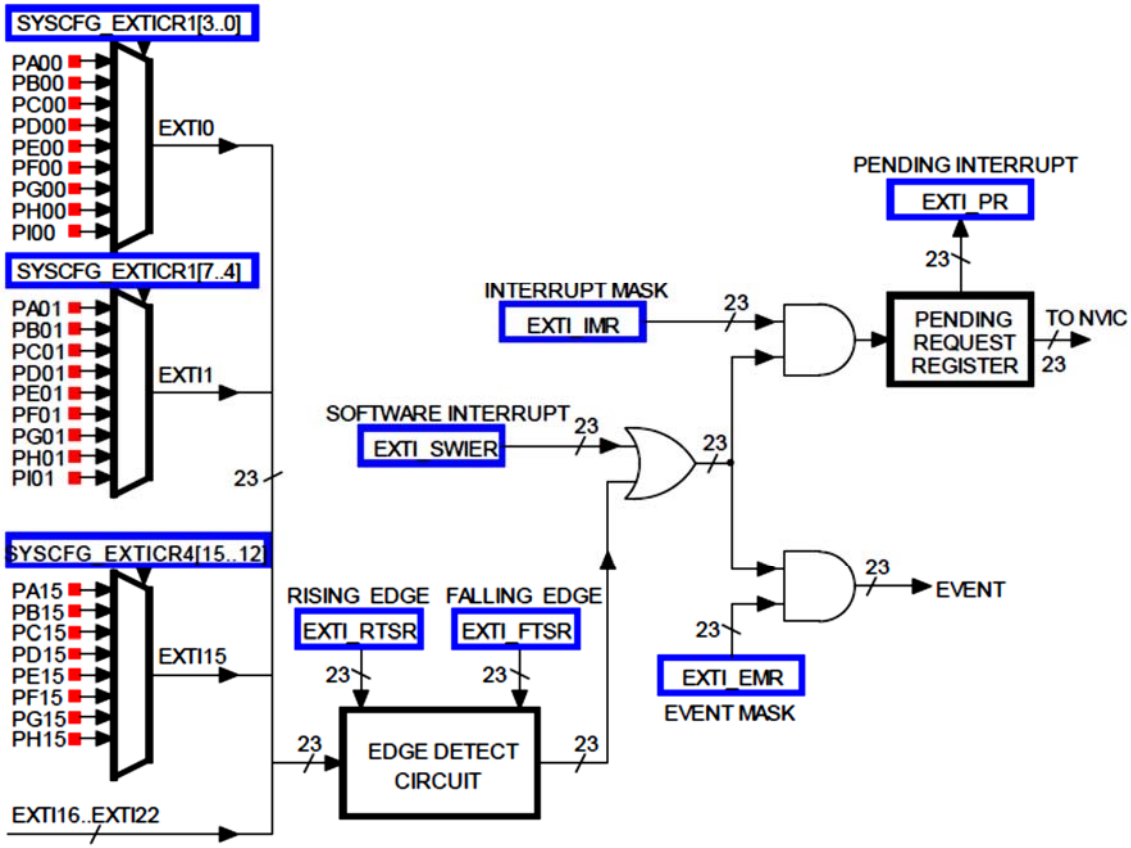
```
NVIC_InitTypeDef NVIC_InitStructure;

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
NVIC_InitStructure.NVIC_IRQChannel = TIM10_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

6. Interruption externes (EXTI)

Le module de gestion des interruptions externes, présente à l'NVIC 23 entrées d'interruptions (EXTI_0 à EXTI_22), 16 d'entre eux (EXTI_0 à EXTI_15) proviennent des entrées-sorties GPIO à travers des multiplexeurs. Les 7 lignes EXTI16 à EXTI22 proviennent des périphériques tels que le PVD, RTC, USB … . La figure suivante présente le schéma bloc simplifié du module EXTI. Il est à noter aussi que les lignes EXTI_5 à EXTI_9 et EXTI_10 à EXTI_15 sont multiplexées pour présenter à l'NVIC deux lignes EXTI_5-9 et EXTI_10-15. Toutes les lignes EXTIx peuvent aussi générer des événements pour réveiller le processeur (Wakeup).

Les registres de configuration pour la génération des interruptions externes :

- SYSCFG_EXTICRx : ces registres contiennent les bits de sélections des multiplieurs du module EXTI.
- EXTI_RTSR et EXTI_FTSR : permettent de sélectionner le front à lequel l'interruption est sensible (font montant ou descendant ou le deux).
- EXTI_SWIER : permet de déclencher une interruption externe de manière logicielle.
- EXTI_IMR : masquer ou activer les interruptions externes EXTI_0 à EXTI_23
- EXTI_EMR : masquer ou activer les événements pour le réveil du processeur.
- EXTI_PR : inducteurs (Pending bits) permettent de maintenir l'état de la demande d'interruption. Ce bit doit être remis à zéro par programme.



Les sept lignes EXTI 16 à 22 sont connectées comme suit :

- EXTI 16 est connectée à PVD output
- EXTI 17 est connectée à RTC Alarm event
- EXTI 18 est connectée à USB OTG FS Wakeup event
- EXTI 19 est connectée à Ethernet Wakeup event
- EXTI 20 est connectée à USB OTG HS (configured in FS) Wakeup event
- EXTI 21 est connectée à RTC Tamper and TimeStamp events
- EXTI 22 est connectée à RTC Wakeup event

6.1. Registres de configuration de l'EXTI

6.1.1. SYSCFG external interrupt configuration register x : SYSCFG_EXTICRx (x : 1 .. 4)

SYSCFG_EXTICR1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

...

SYSCFG_EXTICR4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

6.1.2. Interrupt mask register EXTI_IMR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

MRx = 0 : interruption de la ligne x est masquée

MRx = 1 : interruption de la ligne x non masquée

6.1.3. Event mask register EXTI_EMR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

EMRx = 0 : événement masquée

EMRx = 1 : événement non masquée

6.1.4. Rising trigger selection register (EXTI_RTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

TRx = 0 : front montant de la ligne x est désactivé

TRx = 1 : front montant de la ligne x est activé

6.1.5. Falling trigger selection register (EXTI_FTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

TRx = 0 : front descendant de la ligne x est désactivé

TRx = 1 : front descendant de la ligne x est activé

6.1.6. Software interrupt event register (EXTI_SWIER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Ecrire 1 dans SWIERx positionner le bit correspondant PRx. Ce bit est effacé en ecrivant 1 dans PRx.

6.1.7. Pending register (EXTI_PR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	PR20	PR19	PR18	PR17	PR16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

PRx est positionné lorsqu'un front est détecté sur la ligne correspondante. PRx peut être remis à 0 en écrivant 1 ou lors du changement du front dans l'un des registres EXTI_RTSR ou EXTI_FTSR.

6.2. Configuration du module EXTI

Pour faciliter la compréhension, mous allons prendre comme exemple, la génération d'une interruption à partir de la ligne 2 du port GPIOB.

On commence par activer l'horloge du module SYSCFG :

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
```

Activer aussi l'horloge du port (GPIOx) à lequel appartient l'entrée d'interruption.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

Configurer la ligne du GPIO en entrée

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Sélectionner la ligne 2 du GPIOB comme entrée d'interruption externe EXTI_2

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource2);
```

Configurer la ligne EXTI_2 comme entrée d'interruption sensible au front descendant en remplissant une structure de type EXTI_InitTypeDef.

```
typedef struct{
 uint32_t EXTI_Line;
 EXTIMode_TypeDef EXTI_Mode;
 EXTITrigger_TypeDef EXTI_Trigger;
```

```
  FunctionalState EXTI_LineCmd;
}EXTI_InitTypeDef;
```

EXTI_InitTypeDef EXTI_InitStructure ;

```
EXTI_InitStructure.EXTI_Line = EXTI_Line2;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
```

Le module EXTI est configuré, il reste à configurer le module NVIC (voir le module NVIC).

Lorsque le microprocesseur détecte une demande d'interruption, il arrête le traitement en cours et se branche sur la routine d'interruption (ISR). Le corps de la routine d'interruption doit être écrit dans le fichier STM32F2xx_it.c; le nom de la routine doit être identique à celui déclaré dans le fichier Startup_STM32F2xx.s. pour l'interruption externe EXTI_2, la routine d'interruption est la suivante :

```
void EXTI2_IRQHandler(void) {
  if(EXTI_GetITStatus(EXTI_Line2) != RESET)
  {
      /* Ecrire ici le traitement de l'IT */
    /* Clear the EXTI line 2 pending bit */
    EXTI_ClearITPendingBit(EXTI_Line2);
  }
}
```

7. SysTick

Le module Systick est un timer de 24 bits qui fait partie des exceptions. Ce timer peut être utilisé par un OS (Système d'exploitation) ou par votre application afin de gérer des actions périodiques.

Le fichier core_cm3.h fournie une fonction de configuration du timer SysTick en mode interruption.

`uint32_t SysTick_Config(uint32_t ticks)` ; : le paramètre ticks est la valeur de division de la fréquence du système.

Pour programmer ce timer pour générer des interruptions à une fréquence F en Hz, le paramètres ticks prend la valeur SystemCoreClock / F.

Exemple : Pour F = 200 Hz, on écrit `SysTick_Config(SystemCoreClock / 200)`

La routine d'interruption ISR du timer SysTick.

```
void SysTick_Handler(void)
{
      /* traitement de l'IT */
}
```