

MINF
Programmation des PIC32MX

Chapitre 5

Les interruptions



Théorie PIC32MX

Christian HUBER (CHR)
Serge CASTOLDI (SCA)
Version 1.9 novembre 2017

CONTENU

5. Les interruptions du PIC32MX	5-1
5.1. Notion d'interruption	5-1
5.1.1. Aspect temporel du traitement d'une interruption	5-2
5.1.1. Gestion du signal d'interruption	5-3
5.1.2. Fréquence limite d'une interruption	5-4
5.2. Mécanisme des interruptions du PIC32MX	5-5
5.2.1. Quelques caractéristiques	5-5
5.2.2. Le contrôleur d'interruption	5-5
5.2.3. Les registres SFR pour la gestion des interruptions	5-6
5.2.3.1. Le registre INTCON	5-7
5.2.3.2. Le registre INTSTAT	5-8
5.2.3.3. Les registres IFSx	5-8
5.2.3.4. Les registres IECx	5-8
5.2.3.5. Les registres IPCx	5-8
5.2.4. Relation entre sources d'interruption et registres	5-9
5.2.5. Mécanisme de traitement des interruptions	5-11
5.2.5.1. A propos des shadow registers	5-13
5.2.6. Calcul de l'adresse du vecteur d'interruption	5-13
5.2.6.1. Valeurs utilisées par le compilateur	5-13
5.2.6.2. Adresse des vecteurs	5-13
5.2.6.3. Calcul en mode simple vecteur	5-13
5.2.6.4. Calcul en mode multiple vecteur	5-14
5.2.7. Contenu des vecteurs d'interruption	5-14
5.2.8. Contenu système de la réponse à l'interruption	5-14
5.2.9. Priorités des Interruptions	5-14
5.2.9.1. Cas des sous-priorités	5-14
5.3. Gestion des interruptions avec Harmony & XC32	5-15
5.3.1. Fichier à inclure	5-15
5.3.2. Sélection du mode	5-15
5.3.2.1. PLIB_INT_MultiVectorSelect	5-15
5.3.2.2. PLIB_INT_SingleVectorSelect	5-15
5.3.3. Configuration de l'interruption	5-16
5.3.3.1. PLIB_INT_SourceEnable	5-16
5.3.3.2. PLIB_INT_VectorPrioritySet	5-17
5.3.3.3. PLIB_INT_VectorSubPrioritySet	5-18
5.3.3.4. Exemple de configuration	5-18
5.3.4. Relation routine de réponse et vecteur	5-19
5.3.4.1. Indication du niveau de priorité	5-19
5.3.4.2. Situation avec Harmony	5-19
5.3.4.3. Définition des numéros de vecteurs	5-20
5.3.4.4. Détails macro __ISR	5-20
5.3.4.5. Détails macro __ISR_AT_VECTOR	5-21
5.3.4.6. Vector attribute	5-21
5.4. Exemple pratique avec timer	5-22
5.4.1. Principe de l'exemple	5-22
5.4.2. Configuration timer 1 et interruption	5-22

5.4.2.1.	Autorisation de l'interruption lors du Start	5-23
5.4.3.	Routine réponse interruption du timer 1	5-23
5.4.3.1.	Mise à zéro du flag d'interruption	5-23
5.4.4.	Listing assembleur	5-24
5.4.4.1.	Contenu Vecteur timer 1	5-24
5.4.4.2.	Réponse interruption timer1	5-24
5.4.5.	Configuration timer 4 et interruption	5-26
5.4.5.1.	Autorisation de l'interruption lors du Start	5-27
5.4.6.	Routine réponse interruption du timer 4	5-27
5.4.6.1.	Réponse interruption timer4 en assembleur	5-28
5.4.7.	Action pour faire fonctionner le Test	5-29
5.5.	Les interruptions externes	5-30
5.5.1.	Liste des interruptions externes	5-30
5.5.1.1.	Définitions des sources	5-30
5.5.1.2.	Définitions des Vecteurs (plib_int)	5-30
5.5.1.3.	Définitions des Vecteurs (macro __ISR)	5-30
5.5.2.	Configuration de la polarité du flanc	5-31
5.5.2.1.	Type énuméré INT_EXTERNAL_SOURCES	5-31
5.5.2.2.	PLIB_INT_ExternalRisingEdgeSelect, exemple	5-31
5.6.	Temps de réaction d'une interruption externe	5-32
5.6.1.	Principe de l'exemple	5-32
5.6.1.1.	Contrainte Kit pour la réalisation	5-32
5.6.2.	Configuration des Int. Ext. avec le MHC	5-32
5.6.2.1.	Configuration Int3 (Instance 0)	5-33
5.6.2.2.	Configuration Int4 (Instance 1)	5-33
5.6.2.3.	Modification priorité interruption timer 4	5-33
5.6.3.	Code généré pour les interruptions ext. par le MHC	5-33
5.6.3.1.	Configuration Int3	5-34
5.6.3.2.	Configuration Int4	5-34
5.6.4.	Les fonctions SYS_INT	5-34
5.6.4.1.	La macro SYS_INT_VectorPrioritySet	5-34
5.6.4.2.	La macro SYS_INT_VectorSubprioritySet	5-34
5.6.4.3.	La macro SYS_INT_SourceEnable	5-34
5.6.4.4.	La macro SYS_INT_ExternalInterruptTriggerSet	5-35
5.6.5.	Réponses aux interruptions externes	5-35
5.6.5.1.	Réponse interruption Int3	5-35
5.6.5.2.	Réponse interruption Int4	5-35
5.6.6.	Action pour faire fonctionner le Test	5-36
5.6.6.1.	Modification pour le Timer2	5-36
5.6.7.	Mesure des temps de réaction	5-37
5.6.7.1.	Vue d'ensemble	5-37
5.6.7.2.	Temps réaction Int3	5-37
5.6.7.3.	Temps réaction Int4	5-38
5.6.8.	Différence dans les routines de réponse	5-39
5.6.8.1.	ISR Int3 en assembleur	5-39
5.6.8.2.	ISR Int4 en assembleur	5-40
5.6.9.	Conclusion sur les interruptions externes	5-42
5.6.9.1.	Fréquence limite	5-42
5.6.9.2.	Situation favorable	5-42
5.7.	Conclusion	5-42

5.8.	Historique des versions	5-43
5.8.1.	Version 1.0 février 2014	5-43
5.8.2.	Version 1.5 décembre 2014	5-43
5.8.3.	Version 1.7 décembre 2015	5-43
5.8.4.	Version 1.8 novembre 2016	5-43
5.8.5.	Version 1.9 novembre 2017	5-43

5. LES INTERRUPTIONS DU PIC32MX

Ce chapitre traite du mécanisme et de la gestion des interruptions du PIC32MX.

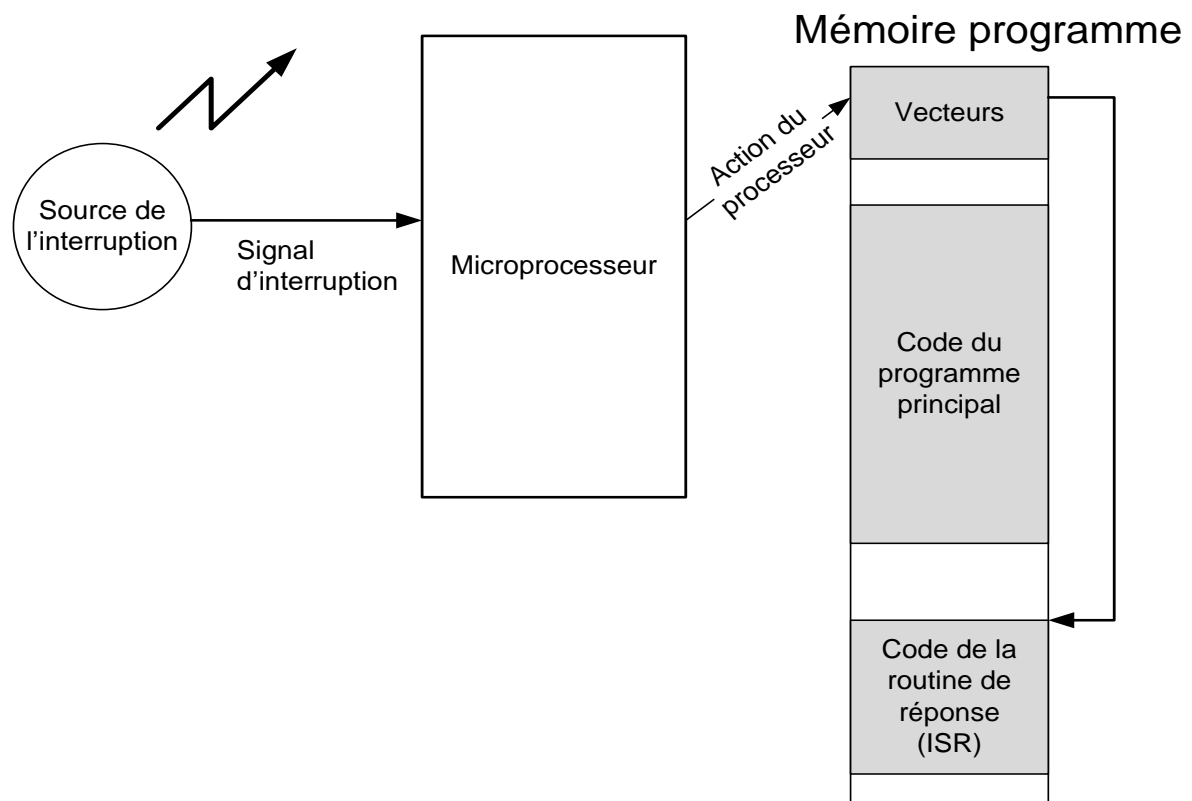
Les documents de référence sont :

- Document MPLAB-XC32-Users-Guide.pdf (documentation du compilateur xc32) :
Chapter 14 : Interrupts
Ce chapitre se base sur le compilateur xc32 v1.42.
- La documentation "PIC32 Family Reference Manual" :
Section 8 : Interrupts
- Pour les détails spécifiques au PIC32MX795F512L, il faut se référer au document
"PIC32MX5XX/6XX/7XX Family Data Sheet" :
Section 7 : Interrupt controller

5.1. NOTION D'INTERRUPTION

Avant de découvrir les détails du mécanisme, voici quelques notions de base concernant les interruptions.

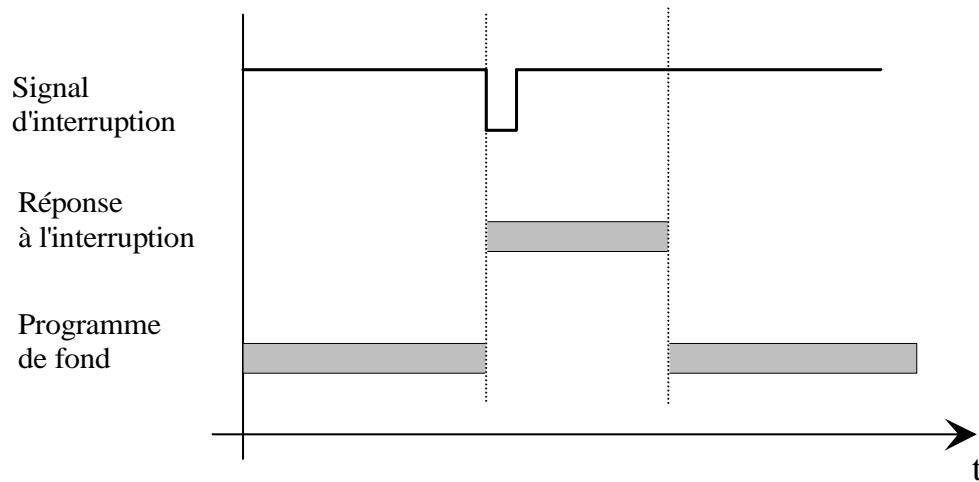
Voici les différents éléments à considérer :



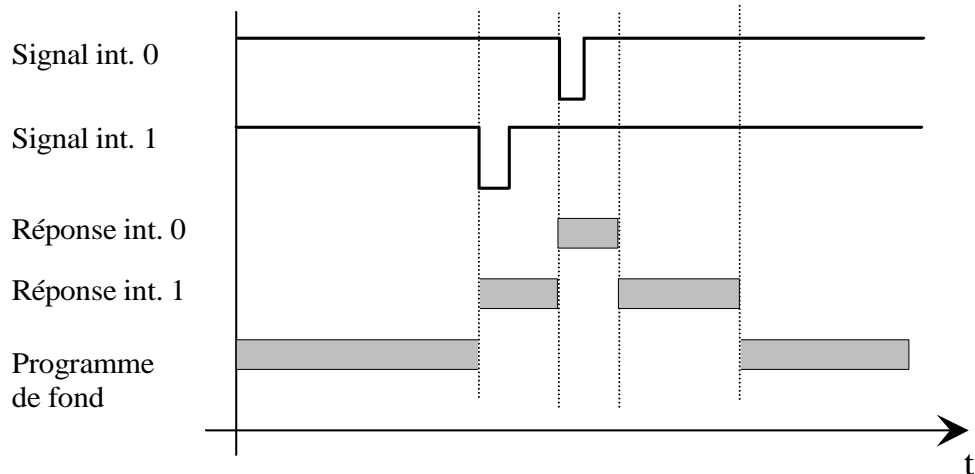
Le mécanisme des interruptions hardware permet d'établir un lien entre un phénomène extérieur au CPU et un sous-programme, dans le but d'exécuter ce sous-programme chaque fois que le phénomène extérieur survient, et ceci en interrompant momentanément le déroulement du programme en exécution à ce moment-là.

Une interruption est signalée au processeur par le changement d'état d'une ligne connectée sur les entrées d'interruption du processeur ou par le changement d'un élément interne au microcontrôleur (cas des timers par exemple).

5.1.1. ASPECT TEMPOREL DU TRAITEMENT D'UNE INTERRUPTION



La plus part des processeurs ont plusieurs entrées d'interruption. Ces entrées n'ont pas la même priorité, ce qui permet d'interrompre l'exécution du sous-programme lié à une interruption (que l'on appelle "routine de réponse à une interruption", en anglais **ISR** pour "*Interrupt Service Routine*") par une interruption plus prioritaire.

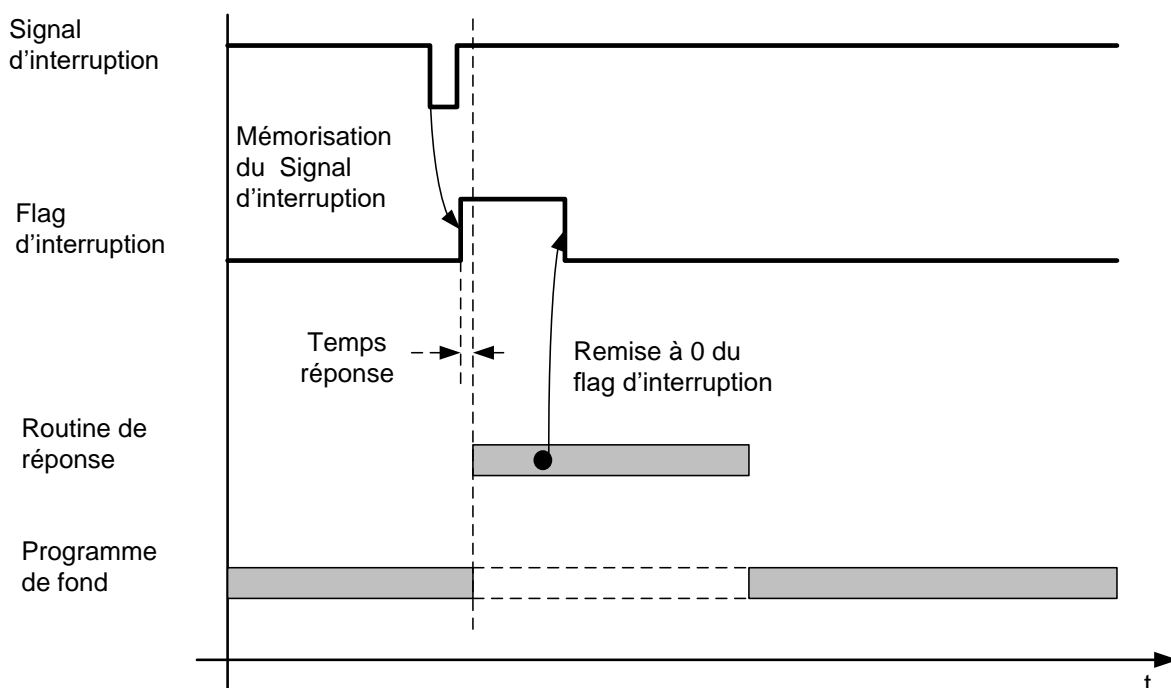


5.1.1. GESTION DU SIGNAL D'INTERRUPTION

En principe, le signal d'interruption est une impulsion. Sa durée doit être suffisante pour que le processeur puisse la détecter et la mémoriser (image de l'interruption dans un bit d'un registre = flag d'interruption).

La remise à 0 du flag d'interruption doit en général être faite par logiciel dans la routine d'interruption.

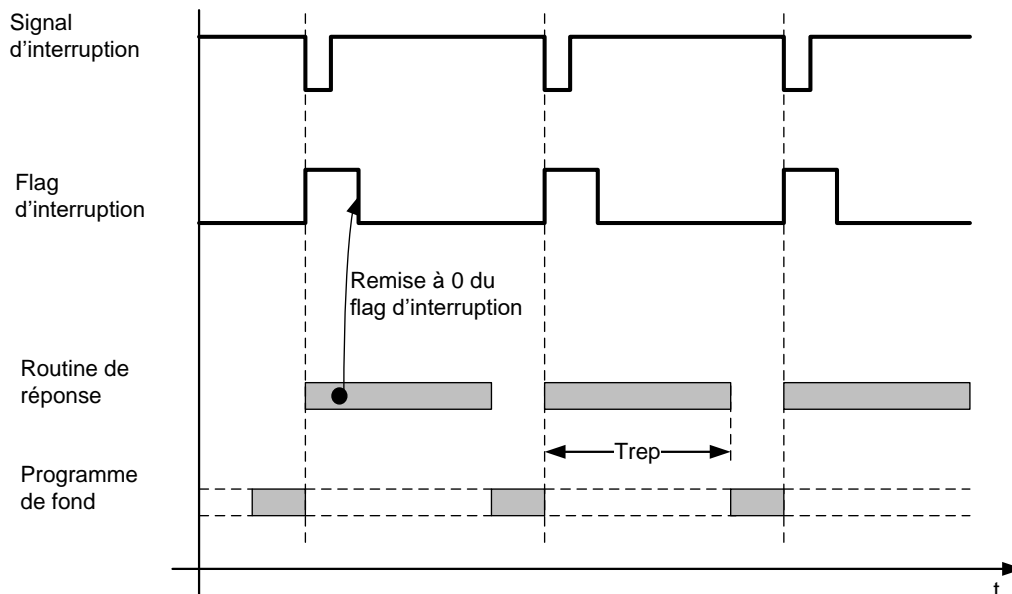
Toutefois, des cas existent où la remise à zéro est automatique (faite par hardware) au moment de l'entrée dans la routine d'interruption. Par exemple, lorsque la source d'interruption est une réception USART (port de communication série), l'action de lire le circuit de communication pourrait indirectement remettre à 0 le flag d'interruption.



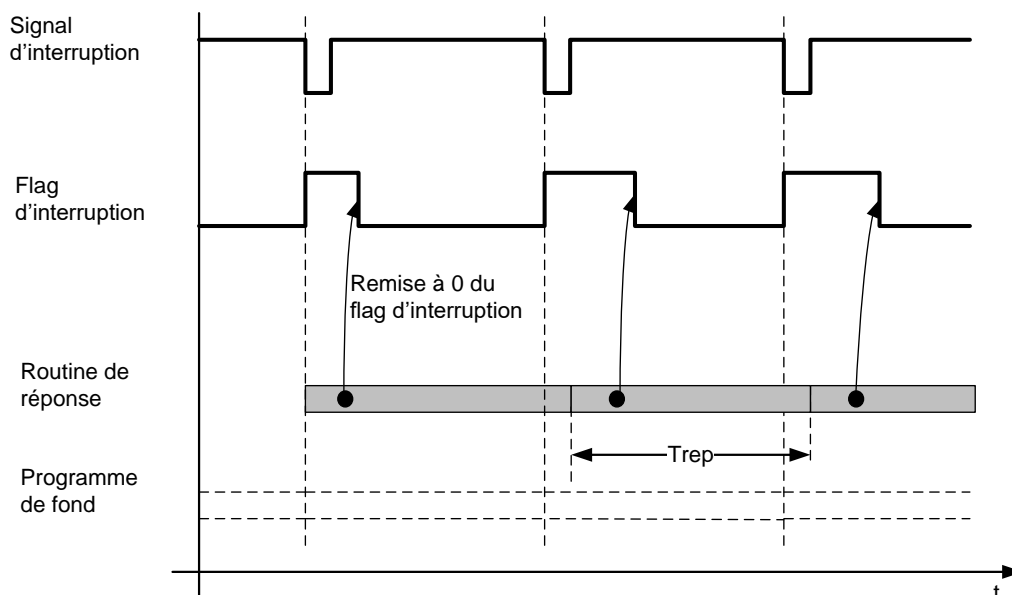
5.1.2. FRÉQUENCE LIMITE D'UNE INTERRUPTION

Lorsque par exemple on utilise un timer pour générer cycliquement une interruption, il faut veiller à ce que le temps d'exécution de la routine de réponse à l'interruption reste inférieur à la période du signal d'interruption. Dans la pratique si l'on souhaite conserver une certaine activité du programme de fond, il faut conserver de l'ordre de 20 % du temps pour ce dernier, donc le temps de traitement de la réponse ne doit pas dépasser 80% de la période du signal d'interruption.

Dans tous les cas, une bonne pratique est de ne traiter **que le strictement nécessaire dans la routine d'interruption** et d'effectuer le reste de la tâche dans l'application (tâche de fond, hors interruption). **Tout blocage, boucle potentiellement bloquante, traitement long ou calcul mathématique complexe est à éviter au maximum dans une routine d'interruption.**



Si le temps de traitement est supérieur à la période du signal d'interruption, il y a perte du synchronisme et consommation complète du temps CPU, le programme de fond ne sera plus exécuté.



5.2. MÉCANISME DES INTERRUPTIONS DU PIC32MX

Dans ce paragraphe, nous allons étudier le mécanisme des interruptions du PIC32MX, en commençant par les éléments d'architecture et les possibilités offertes.

Le PIC32MX gère les requêtes d'interruption en provenance des différents périphériques ainsi que des interruptions externes.

5.2.1. QUELQUES CARACTÉRISTIQUES

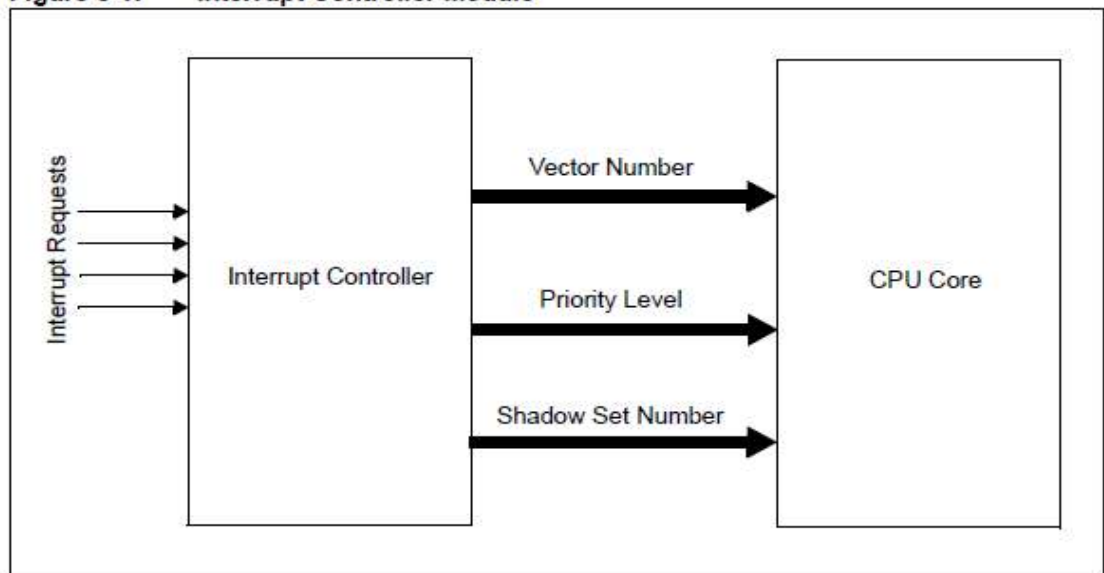
Voici quelques caractéristiques principales:

- Jusqu'à 96 sources d'interruptions.
- Jusqu'à 64 vecteurs d'interruption.
- Mode de gestion simple vecteur ou multi-vecteur. Le mode simple vecteur est un héritage qui ne sera à priori utilisé que rarement.
- Cinq interruptions externes avec configuration de la polarité du flanc.
- Sept niveaux de priorité pour chaque vecteur.
- Quatre niveaux de sous-priorité pour chaque priorité.
- L'adresse de la table des vecteurs d'interruption est configurable par l'utilisateur.
- L'espacement entre les vecteurs d'interruption est configurable par l'utilisateur.

5.2.2. LE CONTRÔLEUR D'INTERRUPTION

Le PIC32MX possède un contrôleur d'interruption séparé du cœur du processeur.

Figure 8-1: Interrupt Controller Module



5.2.3. LES REGISTRES SFR POUR LA GESTION DES INTERRUPTIONS

Le mécanisme de gestion des interruptions utilise certains registres SFR. Il y a 3 registres 32 bits IFS et IEC pour couvrir les 96 sources d'interruption. Il y a 16 registres IPC pour établir les niveaux de priorités.

- **INTCON:** Interrupt Control Register
- **INTSTAT:** Interrupt Status Register
- **TPTMR:** Temporal Proximity Timer Register
- **IFSx:** Interrupt Flag Status Registers
- **IECx:** Interrupt Enable Control Registers
- **IPCx:** Interrupt Priority Control Registers

La table 8-1 résume la situation :

Table 8-1: Interrupts Register Summary

Address Offset	Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0	
0x00	INTCON ^(1,2,3)	31:24	—	—	—	—	—	—	—	—	
		23:16	—	—	—	—	—	—	—	SS0	
		15:8	—	FRZ	—	MVEC	—	TPC<2:0>			
		7:0	—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP	
0x10	INTSTAT ^(1,2,3)	31:24	—	—	—	—	—	—	—	—	
		23:16	—	—	—	—	—	—	—	—	
		15:8	—	—	—	—	—	RIPL<2:0>			
		7:0	—	—	VEC<5:0>						
0x20	TPTMR ^(1,2,3)	31:24	TPTMR<31:24>								
		23:16	TPTMR<23:16>								
		15:8	TPTMR<15:8>								
		7:0	TPTMR<7:0>								
0x30-0x50	IFSx ^(1,2,3)	31:24	IFS31	IFS30	IFS29	IFS28	IFS27	IFS26	IFS25	IFS24	
		23:16	IFS23	IFS22	IFS21	IFS20	IFS19	IFS18	IFS17	IFS16	
		15:8	IFS15	IFS14	IFS13	IFS12	IFS11	IFS10	IFS09	IFS08	
		7:0	IFS07	IFS06	IFS05	IFS04	IFS03	IFS02	IFS01	IFS00	
0x60-0x80	IECx ^(1,2,3)	31:24	IEC31	IEC30	IEC29	IEC28	IEC27	IEC26	IEC25	IEC24	
		23:16	IEC23	IEC22	IEC21	IEC20	IEC19	IEC18	IEC17	IEC16	
		15:8	IEC15	IEC14	IEC13	IEC12	IEC11	IEC10	IEC09	IEC08	
		7:0	IEC07	IEC06	IEC05	IEC04	IEC03	IEC02	IEC01	IEC00	
0x90-0x180	IPCx ^(1,2,3)	31:24	—	—	—	IP03<2:0>				IS03<1:0>	
		23:16	—	—	—	IP02<2:0>				IS02<1:0>	
		15:8	—	—	—	IP01<2:0>				IS01<1:0>	
		7:0	—	—	—	IP00<2:0>				IS00<1:0>	

5.2.3.1. LE REGISTRE INTCON

Ce registre (*INTerrupt CONTROL register*) permet la configuration du mécanisme de traitement des interruptions.

Voici le rôle des différents bits :

bit 31-17	Reserved: Write '0'; ignore read
bit 16	SS0: Single Vector Shadow Register Set bit 1 = Single vector is presented with a shadow register set 0 = Single vector is not presented with a shadow register set
bit 15	Reserved: Write '0'; ignore read
bit 14	FRZ: Freeze in Debug Exception Mode bit 1 = Freeze operation when CPU is in Debug Exception mode 0 = Continue operation even when CPU is in Debug Exception mode Note: FRZ is writable in Debug Exception mode only, it is forced to '0' in normal mode.
bit 13	Reserved: Write '0'; ignore read
bit 12	MVEC: Multi Vector Configuration bit 1 = Interrupt controller configured for multi vectored mode 0 = Interrupt controller configured for single vectored mode
bit 11	Reserved: Write '0'; ignore read

Register 8-1: INTCON: Interrupt Control Register^(1,2,3) (Continued)

bit 10-8	TPC<2:0>: Temporal Proximity Control bits 111 = Interrupts of group priority 7 or lower start the TP timer 110 = Interrupts of group priority 6 or lower start the TP timer 101 = Interrupts of group priority 5 or lower start the TP timer 100 = Interrupts of group priority 4 or lower start the TP timer 011 = Interrupts of group priority 3 or lower start the TP timer 010 = Interrupts of group priority 2 or lower start the TP timer 001 = Interrupts of group priority 1 start the IP timer 000 = Disables proximity timer
bit 7-5	Reserved: Write '0'; ignore read
bit 4	INT4EP: External Interrupt 4 Edge Polarity Control bit 1 = Rising edge 0 = Falling edge
bit 3	INT3EP: External Interrupt 3 Edge Polarity Control bit 1 = Rising edge 0 = Falling edge
bit 2	INT2EP: External Interrupt 2 Edge Polarity Control bit 1 = Rising edge 0 = Falling edge
bit 1	INT1EP: External Interrupt 1 Edge Polarity Control bit 1 = Rising edge 0 = Falling edge
bit 0	INT0EP: External Interrupt 0 Edge Polarity Control bit 1 = Rising edge 0 = Falling edge

On y trouve la sélection simple ou multi vecteurs, ainsi que le choix du flanc des interruptions externes.

5.2.3.2. LE REGISTRE INTSTAT

Ce registre (*INTerrupt STATus register*) fournit des informations sur la situation des interruptions.

Voici le rôle de chacun des bits composant ce registre.

bit 31-11	Reserved: Write '0'; ignore read
bit 10-8	RIPL<2:0>: Requested Priority Level bits 000-111 = The priority level of the latest interrupt presented to the CPU Note: This value should only be used when the interrupt controller is configured for Single Vector mode.
bit 7-6	Reserved: Write '0'; ignore read
bit 5-0	VEC<5:0>: Interrupt Vector bits 00000-11111 = The interrupt vector that is presented to the CPU Note: This value should only be used when the interrupt controller is configured for Single Vector mode.

5.2.3.3. LES REGISTRES IFSx

Ces registres (*Interrupt Flag Status register*) fournissent des informations sur la situation des demandes d'interruptions. Lors d'une demande d'interruption, le bit correspondant à la source est mis à "1".

5.2.3.4. LES REGISTRES IECx

Ces registres (*Interrupt Enable Control register*) établissent l'autorisation d'interruption pour chaque source.

bit 31-0	IEC31-IEC00: Interrupt Enable bits 1 = Interrupt is enabled 0 = Interrupt is disabled
----------	--

5.2.3.5. LES REGISTRES IPCx

Ces registres (*Interrupt Priority Control register*) établissent la situation de priorité et de sous-priorité pour chaque source. Quatre sources par registres.

Register 8-6: IPCx: Interrupt Priority Control Register^(1,2,3,4)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IP03<2:0>		IS03<1:0>		
bit 31							bit 24

bit 31-29	Reserved: Write '0'; ignore read
bit 28-26	IP03<2:0>: Interrupt Priority bits 111 = Interrupt priority is 7 110 = Interrupt priority is 6 101 = Interrupt priority is 5 100 = Interrupt priority is 4 011 = Interrupt priority is 3 010 = Interrupt priority is 2 001 = Interrupt priority is 1 000 = Interrupt is disabled
bit 25-24	IS03<1:0>: Interrupt Subpriority bits 11 = Interrupt subpriority is 3 10 = Interrupt subpriority is 2 01 = Interrupt subpriority is 1 00 = Interrupt subpriority is 0

5.2.4. RELATION ENTRE SOURCES D'INTERRUPTION ET REGISTRES

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION

Interrupt Source ⁽¹⁾	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>
IC1 – Input Capture 1	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>
OC1 – Output Compare 1	6	6	IFS0<6>	IEC0<6>	IPC1<20:18>	IPC1<17:16>
INT1 – External Interrupt 1	7	7	IFS0<7>	IEC0<7>	IPC1<28:26>	IPC1<25:24>
T2 – Timer2	8	8	IFS0<8>	IEC0<8>	IPC2<4:2>	IPC2<1:0>
IC2 – Input Capture 2	9	9	IFS0<9>	IEC0<9>	IPC2<12:10>	IPC2<9:8>
OC2 – Output Compare 2	10	10	IFS0<10>	IEC0<10>	IPC2<20:18>	IPC2<17:16>
INT2 – External Interrupt 2	11	11	IFS0<11>	IEC0<11>	IPC2<28:26>	IPC2<25:24>
T3 – Timer3	12	12	IFS0<12>	IEC0<12>	IPC3<4:2>	IPC3<1:0>
IC3 – Input Capture 3	13	13	IFS0<13>	IEC0<13>	IPC3<12:10>	IPC3<9:8>
OC3 – Output Compare 3	14	14	IFS0<14>	IEC0<14>	IPC3<20:18>	IPC3<17:16>
INT3 – External Interrupt 3	15	15	IFS0<15>	IEC0<15>	IPC3<28:26>	IPC3<25:24>
T4 – Timer4	16	16	IFS0<16>	IEC0<16>	IPC4<4:2>	IPC4<1:0>
IC4 – Input Capture 4	17	17	IFS0<17>	IEC0<17>	IPC4<12:10>	IPC4<9:8>
OC4 – Output Compare 4	18	18	IFS0<18>	IEC0<18>	IPC4<20:18>	IPC4<17:16>
INT4 – External Interrupt 4	19	19	IFS0<19>	IEC0<19>	IPC4<28:26>	IPC4<25:24>
T5 – Timer5	20	20	IFS0<20>	IEC0<20>	IPC5<4:2>	IPC5<1:0>
IC5 – Input Capture 5	21	21	IFS0<21>	IEC0<21>	IPC5<12:10>	IPC5<9:8>
OC5 – Output Compare 5	22	22	IFS0<22>	IEC0<22>	IPC5<20:18>	IPC5<17:16>
SPI1E – SPI1 Fault	23	23	IFS0<23>	IEC0<23>	IPC5<28:26>	IPC5<25:24>
SPI1RX – SPI1 Receive Done	24	23	IFS0<24>	IEC0<24>	IPC5<28:26>	IPC5<25:24>
SPI1TX – SPI1 Transfer Done	25	23	IFS0<25>	IEC0<25>	IPC5<28:26>	IPC5<25:24>
U1E – UART1 Error	26	24	IFS0<26>	IEC0<26>	IPC6<4:2>	IPC6<1:0>
SPI3E – SPI3 Fault						
I2C3B – I2C3 Bus Collision Event						
U1RX – UART1 Receiver	27	24	IFS0<27>	IEC0<27>	IPC6<4:2>	IPC6<1:0>
SPI3RX – SPI3 Receive Done						
I2C3S – I2C3 Slave Event						
U1TX – UART1 Transmitter	28	24	IFS0<28>	IEC0<28>	IPC6<4:2>	IPC6<1:0>
SPI3TX – SPI3 Transfer Done						
I2C3M – I2C3 Master Event						
I2C1B – I2C1 Bus Collision Event	29	25	IFS0<29>	IEC0<29>	IPC6<12:10>	IPC6<9:8>
I2C1S – I2C1 Slave Event	30	25	IFS0<30>	IEC0<30>	IPC6<12:10>	IPC6<9:8>
I2C1M – I2C1 Master Event	31	25	IFS0<31>	IEC0<31>	IPC6<12:10>	IPC6<9:8>
CN – Input Change Interrupt	32	26	IFS1<0>	IEC1<0>	IPC6<20:18>	IPC6<17:16>
AD1 – ADC1 Convert Done	33	27	IFS1<1>	IEC1<1>	IPC6<28:26>	IPC6<25:24>

Note 1: Not all interrupt sources are available on all devices. See [Table 1](#), [Table 2](#) and [Table 3](#) for the list of available peripherals.

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION (CONTINUED)

Interrupt Source ⁽¹⁾	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
PMP – Parallel Master Port	34	28	IFS1<2>	IEC1<2>	IPC7<4:2>	IPC7<1:0>
CMP1 – Comparator Interrupt	35	29	IFS1<3>	IEC1<3>	IPC7<12:10>	IPC7<9:8>
CMP2 – Comparator Interrupt	36	30	IFS1<4>	IEC1<4>	IPC7<20:18>	IPC7<17:16>
U3E – UART2A Error SPI2E – SPI2 Fault I2C4B – I2C4 Bus Collision Event	37	31	IFS1<5>	IEC1<5>	IPC7<28:26>	IPC7<25:24>
U3RX – UART2A Receiver SPI2RX – SPI2 Receive Done I2C4S – I2C4 Slave Event	38	31	IFS1<6>	IEC1<6>	IPC7<28:26>	IPC7<25:24>
U3TX – UART2A Transmitter SPI2TX – SPI2 Transfer Done I2C4M – I2C4 Master Event	39	31	IFS1<7>	IEC1<7>	IPC7<28:26>	IPC7<25:24>
U2E – UART3A Error SPI4E – SPI4 Fault I2C5B – I2C5 Bus Collision Event	40	32	IFS1<8>	IEC1<8>	IPC8<4:2>	IPC8<1:0>
U2RX – UART3A Receiver SPI4RX – SPI4 Receive Done I2C5S – I2C5 Slave Event	41	32	IFS1<9>	IEC1<9>	IPC8<4:2>	IPC8<1:0>
U2TX – UART3A Transmitter SPI4TX – SPI4 Transfer Done I2C5M – I2C5 Master Event	42	32	IFS1<10>	IEC1<10>	IPC8<4:2>	IPC8<1:0>
I2C2B – I2C2 Bus Collision Event	43	33	IFS1<11>	IEC1<11>	IPC8<12:10>	IPC8<9:8>
I2C2S – I2C2 Slave Event	44	33	IFS1<12>	IEC1<12>	IPC8<12:10>	IPC8<9:8>
I2C2M – I2C2 Master Event	45	33	IFS1<13>	IEC1<13>	IPC8<12:10>	IPC8<9:8>
FSCM – Fail-Safe Clock Monitor	46	34	IFS1<14>	IEC1<14>	IPC8<20:18>	IPC8<17:16>
RTCC – Real-Time Clock and Calendar	47	35	IFS1<15>	IEC1<15>	IPC8<28:26>	IPC8<25:24>
DMA0 – DMA Channel 0	48	36	IFS1<16>	IEC1<16>	IPC9<4:2>	IPC9<1:0>
DMA1 – DMA Channel 1	49	37	IFS1<17>	IEC1<17>	IPC9<12:10>	IPC9<9:8>
DMA2 – DMA Channel 2	50	38	IFS1<18>	IEC1<18>	IPC9<20:18>	IPC9<17:16>
DMA3 – DMA Channel 3	51	39	IFS1<19>	IEC1<19>	IPC9<28:26>	IPC9<25:24>
DMA4 – DMA Channel 4	52	40	IFS1<20>	IEC1<20>	IPC10<4:2>	IPC10<1:0>
DMA5 – DMA Channel 5	53	41	IFS1<21>	IEC1<21>	IPC10<12:10>	IPC10<9:8>
DMA6 – DMA Channel 6	54	42	IFS1<22>	IEC1<22>	IPC10<20:18>	IPC10<17:16>
DMA7 – DMA Channel 7	55	43	IFS1<23>	IEC1<23>	IPC10<28:26>	IPC10<25:24>
FCE – Flash Control Event	56	44	IFS1<24>	IEC1<24>	IPC11<4:2>	IPC11<1:0>
USB – USB Interrupt	57	45	IFS1<25>	IEC1<25>	IPC11<12:10>	IPC11<9:8>
CAN1 – Control Area Network 1	58	46	IFS1<26>	IEC1<26>	IPC11<20:18>	IPC11<17:16>
CAN2 – Control Area Network 2	59	47	IFS1<27>	IEC1<27>	IPC11<28:26>	IPC11<25:24>
ETH – Ethernet Interrupt	60	48	IFS1<28>	IEC1<28>	IPC12<4:2>	IPC12<1:0>
IC1E – Input Capture 1 Error	61	5	IFS1<29>	IEC1<29>	IPC1<12:10>	IPC1<9:8>
IC2E – Input Capture 2 Error	62	9	IFS1<30>	IEC1<30>	IPC2<12:10>	IPC2<9:8>
IC3E – Input Capture 3 Error	63	13	IFS1<31>	IEC1<31>	IPC3<12:10>	IPC3<9:8>
IC4E – Input Capture 4 Error	64	17	IFS2<0>	IEC2<0>	IPC4<12:10>	IPC4<9:8>

Note 1: Not all interrupt sources are available on all devices. See [Table 1](#), [Table 2](#) and [Table 3](#) for the list of available peripherals.

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION (CONTINUED)

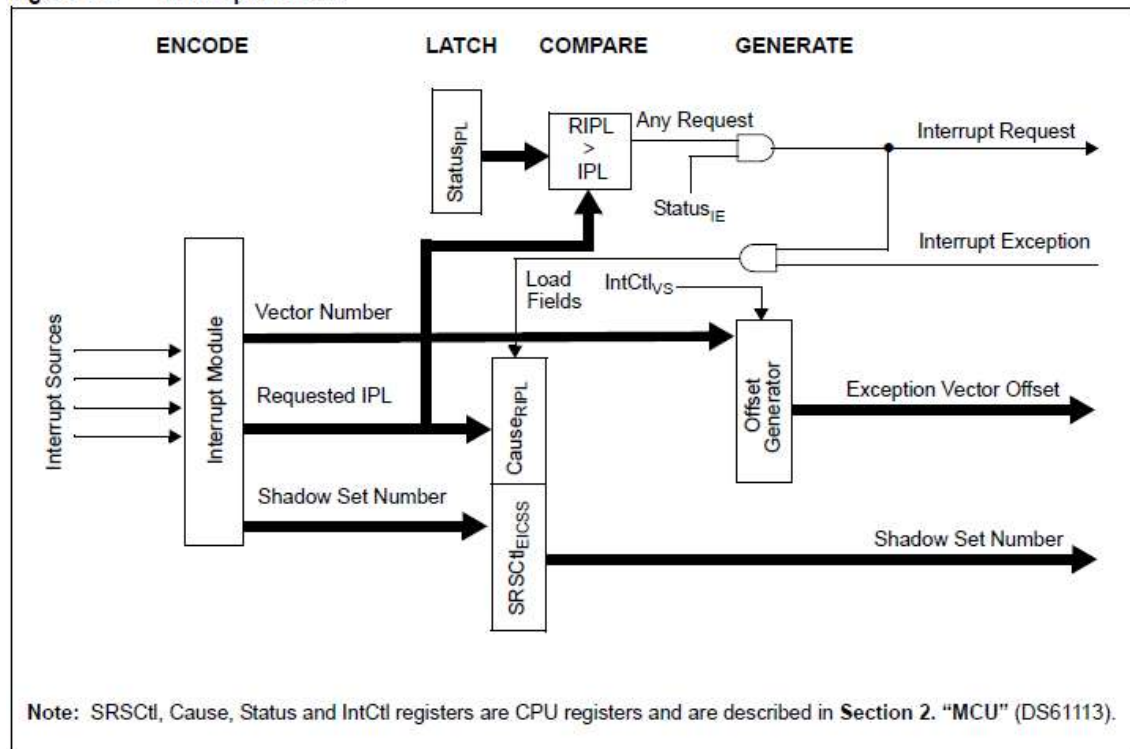
Interrupt Source ⁽¹⁾	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
IC4E – Input Capture 5 Error	65	21	IFS2<1>	IEC2<1>	IPC5<12:10>	IPC5<9:8>
PMPE – Parallel Master Port Error	66	28	IFS2<2>	IEC2<2>	IPC7<4:2>	IPC7<1:0>
U4E – UART4 Error	67	49	IFS2<3>	IEC2<3>	IPC12<12:10>	IPC12<9:8>
U4RX – UART4 Receiver	68	49	IFS2<4>	IEC2<4>	IPC12<12:10>	IPC12<9:8>
U4TX – UART4 Transmitter	69	49	IFS2<5>	IEC2<5>	IPC12<12:10>	IPC12<9:8>
U6E – UART6 Error	70	50	IFS2<6>	IEC2<6>	IPC12<20:18>	IPC12<17:16>
U6RX – UART6 Receiver	71	50	IFS2<7>	IEC2<7>	IPC12<20:18>	IPC12<17:16>
U6TX – UART6 Transmitter	72	50	IFS2<8>	IEC2<8>	IPC12<20:18>	IPC12<17:16>
U5E – UART5 Error	73	51	IFS2<9>	IEC2<9>	IPC12<28:26>	IPC12<25:24>
U5RX – UART5 Receiver	74	51	IFS2<10>	IEC2<10>	IPC12<28:26>	IPC12<25:24>
U5TX – UART5 Transmitter	75	51	IFS2<11>	IEC2<11>	IPC12<28:26>	IPC12<25:24>
(Reserved)	—	—	—	—	—	—
Lowest Natural Order Priority						

Note 1: Not all interrupt sources are available on all devices. See [Table 1](#), [Table 2](#) and [Table 3](#) for the list of available peripherals.

5.2.5. MÉCANISME DE TRAITEMENT DES INTERRUPTIONS

La figure 8-2 illustre le mécanisme de traitement des sources d'interruptions :

Figure 8-2: Interrupt Process



Voici la traduction du paragraphe 8.3 du "PIC32 family reference manual", qui accompagne la figure 8-2 :

Toutes les demandes d'interruption (IRQs) sont échantillonnées au flanc montant de SYSCLK et sont mémorisées dans les registres IFSx correspondant. La demande d'interruption (*pending IRQ*) est indiquée par la mise à 1 du bit dans un des registres IFSx.

La demande d'interruption n'est prise en compte que si le bit correspondant dans un des registres d'autorisation (interrupt enable (IECx) register) est à 1. Lorsque la demande est recevable, elle est encodée en un numéro de vecteur (0 à 63).

Remarque : comme il y a plus de sources d'interruptions que de vecteurs, certaines sources d'interruptions partagent le même no de vecteur (se référer à la table 7-1, §5.2.4 ci-dessus).

A chaque no de vecteur correspond un niveau de priorité (***Requested IPL***) et un "***shadow set number***". Le niveau de priorité est déterminé par le contenu du registre IPCx associé à ce vecteur. En mode multi-vecteur l'utilisateur peut sélectionner un niveau de priorité qui recevra un jeu de "*shadow register*" correspondant. En mode simple vecteur, toutes les interruptions peuvent avoir un jeu de "*shadow register*" dédié.

Le contrôleur d'interruption sélectionne parmi les demandes celle qui a la plus haute priorité et fournit au coeur du processeur (*processor core*) les 3 informations correspondantes (numéro de vecteur, niveau de priorité et no de jeu de "*shadow register*").

Si la priorité du vecteur fourni au coeur est plus élevée que la priorité courante indiquée par le "*CPU Interrupt Priority bits IPL*" (Status<12:10>), alors l'interruption est traitée. Autrement la demande reste en attente (*pending*) jusqu'à ce que la priorité en cours soit plus faible que la demande.

Lorsque le coeur du processeur traite une interruption, le PC (Program Counter) est copié dans le EPC (Exception Program Counter) et le "*Exception Level (EXL) bit*" (Status<1>) est mis à 1. La mise à un du bit EXL a pour effet de désactiver la prise en compte d'autres interruptions jusqu'à ce que le bit EXL soit remis à 0. Ensuite, le coeur modifie le PC pour se brancher à l'adresse du vecteur calculée à partir du no du vecteur. En regardant de plus près un prologue d'interruption généré par le compilateur xc32, on trouve effectivement entre autres, le réglage du nouveau niveau de priorité du CPU (correspondant à l'interruption), puis le clear du bit EXL. De cette manière, le CPU peut à nouveau être interrompu par une interruption plus prioritaire

Le registre INTSTAT contient le numéro du vecteur d'interruption (bits VEC, INTSTAT<5:0>) et le niveau de priorité demandée (bits RIPL, INTSTAT<10:8>), ceci pour la demande en attente. Ces valeurs peuvent différer de celles qui ont amené le coeur à commuter sur le traitement d'exception.

A la fin de la réponse à l'interruption, le processeur retourne à l'état précédant lorsque l'instruction **ERET** (Exception Return) est exécutée. L'instruction ERET remet à zéro le bit EXL, restaure le PC et commute le jeu de "*shadow register*" sur le précédent.

5.2.5.1. A PROPOS DES SHADOW REGISTERS

L'architecture du PIC32 met à disposition 2 groupes de registres GPR. Cela permet, par exemple lors du saut dans une routine d'interruption, de switcher de banc de registres au lieu de sauver une partie des registres généraux. Le prologue et l'épilogue sont alors raccourcis, résultant en un code plus rapide.

5.2.6. CALCUL DE L'ADRESSE DU VECTEUR D'INTERRUPTION

Le calcul est différent entre le mode simple vecteur ou multiple vecteur. Il s'appuie sur la valeur du registre CPU EBASE (Exception Base (EBase<31:12>), qui fournit une adresse de base alignée sur des pages de 4 KB située dans le segment de noyau (KSEG Kernel SEGment).

5.2.6.1. VALEURS UTILISÉES PAR LE COMPILATEUR

Le fichier script de link par défaut p32MX795F512L.ld (spécifique à chaque uC - trouvable dans les sous-répertoires du compilateur xc32) fournit les valeurs par défaut pour notre processeur :

- vector_spacing = 0x00000001
- ebase_address = 0x9FC01000

5.2.6.2. ADRESSE DES VECTEURS

Un fichier .map de diagnostic est généré à la compilation du projet. On y trouve la liste des vecteurs d'interruption. Voici les adresses des quelques premiers :

Exception-Memory Usage section	address	length [bytes]	(dec)	Description
-----	-----	-----	-----	-----
.app_excpt	0x9fc01180	0x10	16	General-Exception
.vector_0	0x9fc01200	0x8	8	Interrupt Vector 0
.vector_1	0x9fc01220	0x8	8	Interrupt Vector 1
.vector_2	0x9fc01240	0x8	8	Interrupt Vector 2
.vector_3	0x9fc01260	0x8	8	Interrupt Vector 3
.vector_4	0x9fc01280	0x8	8	Interrupt Vector 4
.vector_5	0x9fc012a0	0x8	8	Interrupt Vector 5
.vector_6	0x9fc012c0	0x8	8	Interrupt Vector 6
.vector_7	0x9fc012e0	0x8	8	Interrupt Vector 7

5.2.6.3. CALCUL EN MODE SIMPLE VECTEUR

Adresse unique donnée par la formule :

$$\text{Single Vector Address} = \text{EBase} + 0x200$$

Avec Exception Base = 0x9FC01000, on obtient 0x9FC01200.

5.2.6.4. CALCUL EN MODE MULTIPLE VECTEUR

L'adresse en mode multi vecteur est calculée à partir des valeurs de EBase et VS (IntCtl<9:5>) (les registres IntCtl et le Status sont situés dans le CPU). Les bits VS (Vector Spacing) fournissent l'espace entre les adresses de 2 vecteurs adjacents.

L'espace entre les adresses de vecteur peut être de 32, 64, 128, 256 et 512 bytes.

Le principe de calcul est le suivant :

$$\text{Vector address} = \text{vector number} * (\text{VS} \ll 5) + 0x200 + \text{vector base}$$

Par exemple pour le vecteur numéro 4 (timer 1) :

$$\text{Exception Base} = 0x9FC01000$$

$$\text{Vector Spacing(VS)} = 1 : (1 \ll 5) = 32 (0x20)$$

$$\text{Vector address(Timer1)} = 4 * 0x20 + 0x200 + 0x9FC01000 = 0x9FC01280$$

Ce qui correspond à la table ci-dessus.

5.2.7. CONTENU DES VECTEURS D'INTERRUPTION

Les vecteurs d'interruption, vu leur espacement modeste, vont contenir une instruction de saut à la routine de réponse à l'interruption. Une routine très courte peut éventuellement directement y être logée. Il serait également possible de les espacer davantage.

5.2.8. CONTENU SYSTÈME DE LA RÉPONSE À L'INTERRUPTION

Le compilateur va ajouter au code visible de la réponse à l'interruption, les instructions nécessaires à la sauvegarde de certains registres, à l'exécution de la routine de réponse à l'interruption, puis à la restauration des registres sauvegardés et pour finir l'instruction ERET.

En mode simple vecteur, le compilateur devra générer les instructions nécessaires à la sélection de la routine de réponse en relation avec la source d'interruption.

En mode multi vecteur la sélection de la source sera limitée aux différentes sources pour un même vecteur d'où un temps de traitement plus court.

5.2.9. PRIORITÉS DES INTERRUPTIONS

Dans le mode multi vecteur, l'utilisateur peut assigner un niveau de priorité et un niveau de sous-priorité pour chacun des vecteurs d'interruptions. Le niveau de priorité va de 1 (la priorité la plus basse) à 7 (la plus haute).

Si la priorité est établie à 0, le vecteur d'interruption est invalidé pour l'interruption et le mécanisme de réveil (*wake-up purposes*).

Si une interruption survient avec un niveau de priorité plus élevé que celui en cours, la nouvelle interruption est prise en compte et interrompt la routine de réponse à l'interruption de niveau inférieur.

5.2.9.1. CAS DES SOUS-PRIORITÉS

Dans le cas de **demandes simultanées** de deux interruptions avec le même niveau de priorité, la demande avec le niveau de sous-priorité le plus élevé sera traité en premier. Le niveau de sous-priorité va de 0 (la plus basse sous-priorité) à 3 (la plus haute).

⊗ La sous-priorité ne crée pas de niveau de priorité intermédiaire, c'est uniquement un arbitrage lors de demande simultanée.

5.3. GESTION DES INTERRUPTIONS AVEC HARMONY & XC32

Le compilateur XC32 fournit un certain nombre de macros et fonctions pour configurer les interruptions et agir dans la réponse sur le flag d'interruption. Il offre aussi des directives pour transformer une fonction en une routine de réponse à une interruption donnée.



Cette partie se réfère à la documentation de Harmony, disponible dans le fichier <Répertoire Harmony>v1_<n>\doc\help_harmony.pdf, section *Framework Libraries Help* > *Interrupt Peripheral Library* (plib_int). Elle a été réalisée sur la base de la version Harmony 1.08.

5.3.1. FICHIER À INCLURE

Pour utiliser les fonctions de la PLIB_INT il est nécessaire d'inclure le fichier **plib_int.h**.

5.3.2. SÉLECTION DU MODE

On dispose de 2 fonctions pour la sélection du mode multi ou single vector :

	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.

5.3.2.1. PLIB_INT_MULTIVECTORSELECT

```
void PLIB_INT_MultiVectorSelect(INT_MODULE_ID index);
```

Cette fonction configure le mécanisme des interruptions avec l'utilisation des différents vecteurs à disposition.

Dans un projet obtenu avec le configurateur Harmony, cette fonction est appelée dans la fonction SYS_INT_Initialize.

```
void SYS_INT_Initialize ( void )
{
    /* enable the multi vector */
    PLIB_INT_MultiVectorSelect( INT_ID_0 );
}
```

Il est à noter que la valeur de l'index doit être de INT_ID_0 pour les PIC32MX, qui ne possèdent qu'un seul module de gestion des interruptions.

5.3.2.2. PLIB_INT_SINGLEVECTORSELECT

```
void PLIB_INT_SingleVectorSelect(INT_MODULE_ID index);
```

Cette fonction configure le mécanisme des interruptions avec l'utilisation d'un vecteur unique. La valeur d'index est aussi INT_ID_0.

5.3.3. CONFIGURATION DE L'INTERRUPTION

La configuration d'une interruption correspond à 3 actions :

- Autoriser la source d'interruption,
- Etablir le niveau de priorité,
- Etablir le niveau de sous-priorité.

A chaque action correspond une fonction.

5.3.3.1. PLIB_INT_SOURCEENABLE

```
void PLIB_INT_SourceEnable(INT_MODULE_ID index, INT_SOURCE source);
```

Cette fonction autorise la source de l'interruption.

Le type énuméré **INT_SOURCE** liste les sources possibles. Voici le début et la fin de la liste :

```
typedef enum {
    INT_SOURCE_SOFTWARE_0,
    INT_SOURCE_SOFTWARE_1,
    INT_SOURCE_EXTERNAL_0,
    INT_SOURCE_EXTERNAL_1,
    INT_SOURCE_EXTERNAL_2,
    INT_SOURCE_EXTERNAL_3,
    INT_SOURCE_EXTERNAL_4,
    INT_SOURCE_TIMER_CORE,
    INT_SOURCE_TIMER_1,
    INT_SOURCE_TIMER_2,
    INT_SOURCE_TIMER_3,
    INT_SOURCE_TIMER_4,
    INT_SOURCE_TIMER_5,
    INT_SOURCE_INPUT_CAPTURE_1,
    INT_SOURCE_INPUT_CAPTURE_1_ERROR,
    INT_SOURCE_INPUT_CAPTURE_2,
    INT_SOURCE_INPUT_CAPTURE_2_ERROR,
    INT_SOURCE_INPUT_CAPTURE_3,
    INT_SOURCE_INPUT_CAPTURE_3_ERROR,
    INT_SOURCE_INPUT_CAPTURE_4,
    INT_SOURCE_INPUT_CAPTURE_4_ERROR,
    INT_SOURCE_INPUT_CAPTURE_5,
    INT_SOURCE_INPUT_CAPTURE_5_ERROR,
    INT_SOURCE_OUTPUT_COMPARE_1,
    INT_SOURCE_OUTPUT_COMPARE_2,
    INT_SOURCE_OUTPUT_COMPARE_3,
    INT_SOURCE_OUTPUT_COMPARE_4,
    INT_SOURCE_OUTPUT_COMPARE_5,
    INT_SOURCE_SPI_1_ERROR,
    INT_SOURCE_SPI_1_RECEIVE,
    INT_SOURCE_SPI_1_TRANSMIT,
    INT_SOURCE_SPI_2_ERROR,
    INT_SOURCE_SPI_2_RECEIVE,
    INT_SOURCE_SPI_2_TRANSMIT,
    INT_SOURCE_SPI_3_ERROR,
    INT_SOURCE_SPI_3_RECEIVE,
    INT_SOURCE_SPI_3_TRANSMIT,
    INT_SOURCE_I2C_1_MASTER,
    INT_SOURCE_I2C_2_ERROR,
    INT_SOURCE_I2C_2_SLAVE,
    INT_SOURCE_I2C_2_MASTER,
    INT_SOURCE_I2C_3_ERROR,
    INT_SOURCE_I2C_3_SLAVE,
    INT_SOURCE_I2C_3_MASTER,
    INT_SOURCE_I2C_4_ERROR,
    INT_SOURCE_I2C_4_SLAVE,
    INT_SOURCE_I2C_4_MASTER,
    INT_SOURCE_I2C_5_ERROR,
    INT_SOURCE_I2C_5_SLAVE,
    INT_SOURCE_I2C_5_MASTER,
    INT_SOURCE_CHANGE_NOTICE,
    INT_SOURCE_ADC_1,
    INT_SOURCE_PARALLEL_PORT,
    INT_SOURCE_PARALLEL_PORT_ERROR,
    INT_SOURCE_COMPARATOR_1,
    INT_SOURCE_COMPARATOR_2,
    INT_SOURCE_CLOCK_MONITOR,
    INT_SOURCE_RTCC,
    INT_SOURCE_DMA_0,
    INT_SOURCE_DMA_1,
    INT_SOURCE_DMA_2,
    INT_SOURCE_DMA_3,
    INT_SOURCE_DMA_4,
    INT_SOURCE_DMA_5,
    INT_SOURCE_DMA_6,
    INT_SOURCE_DMA_7,
    INT_SOURCE_FLASH_CONTROL,
    INT_SOURCE_USB_1,
    INT_SOURCE_CAN_1,
    INT_SOURCE_CAN_2,
    INT_SOURCE_ETH_1
} INT_SOURCE;
```

5.3.3.2. PLIB_INT_VECTORPRIORITYSET

```
void PLIB_INT_VectorPrioritySet(INT_MODULE_ID index, INT_VECTOR vector, INT_PRIORITY_LEVEL
priority);
```

Cette fonction établit le niveau de priorité du vecteur d'interruption.

👉 Il ne faut pas confondre SOURCE d'interruption et VECTEUR d'interruption. Par contre il doit y avoir correspondance entre la source et le vecteur.

Le type énuméré **INT_VECTOR** liste les vecteurs à disposition. Voici le début et la fin de la liste :

```
typedef enum {
    INT_VECTOR_CT,
    INT_VECTOR_CS0,
    INT_VECTOR_CS1,
    INT_VECTOR_INT0,
    INT_VECTOR_T1,
    INT_VECTOR_IC1,
    INT_VECTOR_OC1,
    INT_VECTOR_INT1,
    INT_VECTOR_T2,
    INT_VECTOR_IC2,
    INT_VECTOR_OC2,
    INT_VECTOR_INT2,
    INT_VECTOR_T3,
    INT_VECTOR_IC3,
    INT_VECTOR_OC3,
    INT_VECTOR_INT3,
    INT_VECTOR_T4,
    INT_VECTOR_IC4,
    INT_VECTOR_OC4,
    INT_VECTOR_INT4,
    INT_VECTOR_T5,
    INT_VECTOR_IC5,
    INT_VECTOR_UART2,
    INT_VECTOR_SPI4,
    INT_VECTOR_I2C5,
    INT_VECTOR_I2C2,
    INT_VECTOR_FSCM,
    INT_VECTOR_RTCC,
    INT_VECTOR_DMA0,
    INT_VECTOR_DMA1,
    INT_VECTOR_DMA2,
    INT_VECTOR_DMA3,
    INT_VECTOR_DMA4,
    INT_VECTOR_DMA5,
    INT_VECTOR_DMA6,
    INT_VECTOR_DMA7,
    INT_VECTOR_FCE,
    INT_VECTOR_USB,
    INT_VECTOR_CAN1,
    INT_VECTOR_CAN2,
    INT_VECTOR_ETH,
    INT_VECTOR_UART4,
    INT_VECTOR_UART6,
    INT_VECTOR_UART5
} INT_VECTOR;
```

Le type énuméré **INT_PRIORITY_LEVEL** liste les niveaux de priorité à disposition :

```
typedef enum {
    INT_PRIORITY_LEVEL0,
    INT_PRIORITY_LEVEL1,
    INT_PRIORITY_LEVEL2,
    INT_PRIORITY_LEVEL3,
    INT_PRIORITY_LEVEL4,
    INT_PRIORITY_LEVEL5,
    INT_PRIORITY_LEVEL6,
    INT_PRIORITY_LEVEL7
} INT_PRIORITY_LEVEL;
```

Le LEVEL0 est indiqué Disabled !

Members	Description
INT_PRIORITY_LEVEL0	Disabled
INT_PRIORITY_LEVEL1	Priority 1
INT_PRIORITY_LEVEL2	Priority 2

👉 Le niveau 1 est la plus basse priorité tandis que le niveau 7 est la plus haute priorité.

5.3.3.3. PLIB_INT_VECTORSUBPRIORITYSET

```
void PLIB_INT_VectorSubPrioritySet(INT_MODULE_ID index, INT_VECTOR vector,  
INT_SUBPRIORITY_LEVEL subPriority);
```

Cette fonction établit le niveau de sous-priorité du vecteur d'interruption.

Le type énuméré **INT_SUBPRIORITY_LEVEL** liste les niveaux de priorité à disposition :

```
typedef enum {  
    INT_SUBPRIORITY_LEVEL0,  
    INT_SUBPRIORITY_LEVEL1,  
    INT_SUBPRIORITY_LEVEL2,  
    INT_SUBPRIORITY_LEVEL3  
} INT_SUBPRIORITY_LEVEL;
```

La plus basse sous-priorité est le niveau 0 et la plus haute sous-priorité est le niveau 3.

5.3.3.4. EXEMPLE DE CONFIGURATION

Voici comme exemple la configuration de l'interruption du timer1 (obtenu avec le MHC) :

```
/* Setup Interrupt */  
PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_1);  
PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,  
                           INT_PRIORITY_LEVEL3);  
PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,  
                              INT_SUBPRIORITY_LEVEL0);
```

On constate qu'au niveau de la source, le timer1 est indiqué par **TIMER_1** alors que pour le vecteur il est indiqué **T1**.

👉 Lors de la réalisation de la routine de réponse à l'interruption avec la macro **__ISR**, il faut impérativement utiliser le même niveau de priorité.

5.3.4. RELATION ROUTINE DE RÉPONSE ET VECTEUR

Le compilateur XC32 met à disposition des macros et attributs pour transformer une fonction en une routine de réponse à une interruption et créer le lien avec le vecteur correspondant.

👉 Il est à noter que pour la routine de réponse, les éléments syntaxiques sont uniquement liés au compilateur et non pas à la PLIB_INT de Harmony.

Le fichier header `<sys\attribs.h>` fournit des macros pour simplifier l'application des *attributes* aux fonctions d'interruption. On trouve aussi des "vector macros" dans les fichiers d'entête propres à un modèle de processeur.

On trouve les fichiers d'entête des processeurs du compilateur XC32 sous :

`<Répertoire xc32>\v<n>\pic32mx\include\proc`

On dispose de 2 macros `__ISR`:

- `__ISR(V, IPL)`
- `__ISR_AT_VECTOR(V, IPL)`

5.3.4.1. INDICATION DU NIVEAU DE PRIORITÉ

Pour indiquer le niveau de priorité *n* (*n* de 1 à 7), on dispose de 3 formes:

`IPLnSRS`, `IPLnSOFT` et `IPLnAUTO`.

- Avec SRS le compilateur utilisera un Shadow Register Set dans le traitement de sauvegarde et restauration → prologue et épilogue et raccourcis. On peut par exemple imaginer cette stratégie pour une interruption dont le temps de réaction, la fréquence ou encore la durée seraient critiques.

👉 En mode multi vecteurs, qui est le mode que nous utiliserons en pratique, une seule priorité d'interruption peut utiliser le SRS. Le niveau concerné se configure via le MHC, dans `DEVCFG3`.

- Avec SOFT le compilateur effectuera le traitement de sauvegarde restauration en utilisant la pile.
- Avec AUTO le compilateur détermine s'il peut utiliser un SRS ou s'il doit utiliser la pile.

5.3.4.2. SITUATION AVEC HARMONY

Lorsque l'on utilise le MHC, on constate l'utilisation de la macro `__ISR`, mais avec une variante de la macro pour préciser le niveau d'interruption, avec **ipl** en minuscule.

D'où les 3 formes : `iplnSRS`, `iplnSOFT` et `iplnAUTO`.

👉 Nous utiliserons cette forme par la suite.

5.3.4.3. DÉFINITION DES NUMÉROS DE VECTEURS

Dans le fichier p32mx795f512l.h, on trouve toutes les définitions des vecteurs d'interruptions.

👉 Avec la macro `__ISR`, on doit utiliser la définition des vecteurs qui se trouve dans les fichiers du processeur et non pas le type énuméré de la `PLIB_INT`.

```

/* Vector Numbers */
#define _CORE_TIMER_VECTOR          0
#define _CORE_SOFTWARE_0_VECTOR    1
#define _CORE_SOFTWARE_1_VECTOR    2
#define _EXTERNAL_0_VECTOR          3
#define _TIMER_1_VECTOR             4
#define _INPUT_CAPTURE_1_VECTOR     5
#define _OUTPUT_COMPARE_1_VECTOR    6
#define _EXTERNAL_1_VECTOR          7
#define _TIMER_2_VECTOR             8
#define _INPUT_CAPTURE_2_VECTOR     9
#define _OUTPUT_COMPARE_2_VECTOR   10
#define _EXTERNAL_2_VECTOR          11
#define _TIMER_3_VECTOR             12
#define _INPUT_CAPTURE_3_VECTOR     13
#define _OUTPUT_COMPARE_3_VECTOR    14
#define _EXTERNAL_3_VECTOR          15
#define _TIMER_4_VECTOR             16
#define _INPUT_CAPTURE_4_VECTOR     17
#define _OUTPUT_COMPARE_4_VECTOR    18
#define _EXTERNAL_4_VECTOR          19
#define _TIMER_5_VECTOR             20
#define _INPUT_CAPTURE_5_VECTOR     21
#define _OUTPUT_COMPARE_5_VECTOR    22
#define _SPI_1_VECTOR               23
#define _I2C_3_VECTOR               24
#define _I2C_1A_VECTOR              24
#define _SPI_3_VECTOR               24
#define _SPI_1A_VECTOR              24
#define _UART_1_VECTOR              24
#define _UART_1A_VECTOR              24
#define _I2C_1_VECTOR               25
#define _CHANGE_NOTICE_VECTOR       26
#define _ADC_VECTOR                 27
#define _PMP_VECTOR                 28
#define _COMPARATOR_1_VECTOR        29

#define _COMPARATOR_2_VECTOR        30
#define _I2C_4_VECTOR               31
#define _I2C_2A_VECTOR              31
#define _SPI_2_VECTOR               31
#define _SPI_2A_VECTOR              31
#define _UART_3_VECTOR              31
#define _UART_2A_VECTOR             31
#define _I2C_5_VECTOR               32
#define _I2C_3A_VECTOR              32
#define _SPI_4_VECTOR               32
#define _SPI_3A_VECTOR              32
#define _UART_2_VECTOR              32
#define _UART_3A_VECTOR             32
#define _I2C_2_VECTOR               33
#define _FAIL_SAFE_MONITOR_VECTOR   34
#define _RTCC_VECTOR                35
#define _DMA_0_VECTOR               36
#define _DMA_1_VECTOR               37
#define _DMA_2_VECTOR               38
#define _DMA_3_VECTOR               39
#define _DMA_4_VECTOR               40
#define _DMA_5_VECTOR               41
#define _DMA_6_VECTOR               42
#define _DMA_7_VECTOR               43
#define _USB_1_VECTOR               45
#define _CAN_1_VECTOR               46
#define _CAN_2_VECTOR               47
#define _ETH_VECTOR                 48
#define _UART_4_VECTOR              49
#define _UART_1B_VECTOR             49
#define _UART_6_VECTOR              50
#define _UART_2B_VECTOR             50
#define _UART_5_VECTOR              51
#define _UART_3B_VECTOR             51
#define _FCE_VECTOR                 44

```

5.3.4.4. DÉTAILS MACRO `__ISR`

La macro `__ISR(V, IPL)` affecte le numéro du vecteur et l'associe avec le niveau de priorité spécifiée. Cette macro a pour effet de placer un saut à la routine de réponse à l'interruption, dans la zone du vecteur correspondant.

5.3.4.4.1. Macro `__ISR`, exemple `harmony`

L'exemple ci-dessous correspond à la routine de réponse à l'interruption du timer 1 :

```
void __ISR(_TIMER_1_VECTOR, ip13AUTO) Timer1Handler(void)
```

Spécification du niveau de priorité 3 en AUTO.

5.3.4.5. DÉTAILS MACRO `__ISR_AT_VECTOR`

La macro `__ISR_AT_VECTOR(V, IPL)` affecte le numéro du vecteur et l'associe avec le niveau de priorité spécifiée. Cette macro a pour effet de placer la routine de réponse à l'interruption directement dans la zone du vecteur correspondant.

Remarque : le Vector Spacing doit être augmenté, ou la routine particulièrement courte, afin de disposer d'assez de place pour le code.

5.3.4.5.1. Macro `__ISR_AT_VECTOR`, exemple

L'exemple ci-dessous correspond à la routine de réponse à l'interruption du timer 2 :

```
void __ISR_AT_VECTOR(_TIMER_2_VECTOR, IPL7SRS)
                                Timer2Handler(void)
```

Spécification du niveau de priorité 7 en SRS.

⊗ Provoque une erreur "function at exception vector 8 too large".

Une solution serait de modifier la valeur du Vector Spacing. Le lecteur est invité à se reporter à la documentation du compilateur xc32 pour ceci.

5.3.4.6. VECTOR ATTRIBUTE

Une autre façon d'établir la relation de la fonction avec le vecteur est d'utiliser le vector attribute. On obtient le même résultat qu'avec le `__ISR`, mais la syntaxe est un peu plus lourde. Par exemple :

```
void __attribute__ ((interrupt(IPL6SOFT))) __attribute__
      ((vector(_TIMER_3_VECTOR))) Timer3Handler(void)
```

5.4. EXEMPLE PRATIQUE AVEC TIMER

Voici un exemple complet utilisant 2 timers avec interruption obtenu avec le Microchip Harmony Configurator. Nous suivrons les détails pour le timer 1 et le timer 4.

5.4.1. PRINCIPE DE L'EXEMPLE

Utilisation de 2 timers et 2 sorties, l'action dans les routines de réponse à l'interruption consiste à inverser la led correspondante.



Choix d'une période de 1 ms pour le timer 1 (priorité 3) et de 100 us pour le timer 4 (priorité 7, SRS).

5.4.2. CONFIGURATION TIMER 1 ET INTERRUPTION

Dans le fichier `system_init.c`, on trouve l'appel de la fonction `DRV_TMR0_Initialize()`, que l'on trouve dans le fichier `drv_tmr_static.c`.

Voici les actions de configuration du timer 1 et de son interruption.

```
void DRV_TMR0_Initialize(void)
{
    /* Initialize Timer Instance0 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_1);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_1,
                              DRV_TMR_CLKSOURCE_INTERNAL);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_1,
                           TMR_PRESCALE_VALUE_8);
    /* Enable 16 bit mode */
    PLIB_TMR_Model16BitEnable(TMR_ID_1);
    /* Clear counter */
    PLIB_TMR_Counter16BitClear(TMR_ID_1);
    /* Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_1, 10000);
    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,
                              INT_PRIORITY_LEVEL3);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,
                                 INT_SUBPRIORITY_LEVEL0);
}
```

☞ On constate que la source de l'interruption n'est pas autorisée au niveau de la fonction `DRV_TMR0_Initialize()`, de même que le timer 1 n'est pas activé.

5.4.2.1. AUTORISATION DE L'INTERRUPTION LORS DU START

On observe que dans la fonction `DRV_TMR0_Start()` il y a appel de la fonction `_DRV_TMR0_Resume()` qui s'occupe de l'autorisation de la source de l'interruption.

```
static void _DRV_TMR0_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                   INT_SOURCE_TIMER_1);
        PLIB_INT_SourceEnable(INT_ID_0,
                               INT_SOURCE_TIMER_1);
        PLIB_TMR_Start(TMR_ID_1);
    }
}

bool DRV_TMR0_Start(void)
{
    /* Start Timer*/
    _DRV_TMR0_Resume(true);
    DRV_TMR0_Running = true;

    return true;
}
```

5.4.3. ROUTINE RÉPONSE INTERRUPTION DU TIMER 1

Dans le fichier `system_interrupt.c` on trouve la macro `__ISR` correspondant au timer1.

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
           IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    // Test de la période du timer
    BSP_LEDToggle(BSP_LED_1);
}
```

Le niveau d'interruption est indiqué avec "ipl3AUTO".

5.4.3.1. MISE À ZÉRO DU FLAG D'INTERRUPTION

On constate qu'il est nécessaire de mettre à 0 le flag d'interruption en utilisant une fonction spécifique de la `PLIB_INT`:

```
PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
```

```
void PLIB_INT_SourceFlagClear(INT_MODULE_ID index, INT_SOURCE source);
```

Cette fonction utilise le type énuméré `INT_SOURCE`.

5.4.4. LISTING ASSEMBLEUR

5.4.4.1. CONTENU VECTEUR TIMER 1

Le vecteur du timer1 est le numéro 4, donc son adresse est :

Exception Base = 0x9FC01000

Vector Spacing(VS) = 1, (1 << 5) = 32 (0x20)

*Vector address(Timer1) = 4 * 0x20 + 0x200 + 0x9FC01000 = 0x9FC01280*

⊗ Le listing assembleur ne montre pas le contenu des vecteurs. En utilisant le debugger avec l'observation du contenu de la mémoire on n'a pas la possibilité d'accéder à la zone des vecteurs.

5.4.4.2. RÉPONSE INTERRUPTION TIMER1

👉 Compilation avec optimisation 0.

On suppose que le contenu du vecteur 4 effectue un saut en 9D003800.

```

74:                                void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
                                IntHandlerDrvTmrInstance0(void)
75:                                {
9D003800 415DE800  RDPGPR SP, SP
9D003804 401B7000  MFC0 K1, EPC
9D003808 401A6002  MFC0 K0, SRSCtl
9D00380C 27BDF88  ADDIU SP, SP, -120
9D003810 AFBB0074  SW K1, 116(SP)
9D003814 401B6000  MFC0 K1, Status
9D003818 AFBA006C  SW K0, 108(SP)
9D00381C AFBB0070  SW K1, 112(SP)
9D003820 7C1B7844  INS K1, ZERO, 1, 15
9D003824 377B0C00  ORI K1, K1, 3072
9D003828 409B6000  MTC0 K1, Status
9D00382C AFA3001C  SW V1, 28(SP)
9D003830 AFA20018  SW V0, 24(SP)
9D003834 8FA3006C  LW V1, 108(SP)
9D003838 3063000F  ANDI V1, V1, 15
9D00383C 14600012  BNE V1, ZERO, 0x9D003888
9D003840 00000000  NOP
9D003844 AFBF005C  SW RA, 92(SP)
9D003848 AFBE0058  SW S8, 88(SP)
9D00384C AFB90054  SW T9, 84(SP)
9D003850 AFB80050  SW T8, 80(SP)
9D003854 AFAF004C  SW T7, 76(SP)
9D003858 AFAE0048  SW T6, 72(SP)
9D00385C AFAD0044  SW T5, 68(SP)
9D003860 AFAC0040  SW T4, 64(SP)
9D003864 AFAB003C  SW T3, 60(SP)
9D003868 AFAA0038  SW T2, 56(SP)
9D00386C AFA90034  SW T1, 52(SP)
9D003870 AFA80030  SW T0, 48(SP)
9D003874 AFA7002C  SW A3, 44(SP)
9D003878 AFA60028  SW A2, 40(SP)
9D00387C AFA50024  SW A1, 36(SP)
9D003880 AFA40020  SW A0, 32(SP)

```

```

9D003884  AFA10014  SW AT, 20(SP)
9D003888  00000000  NOP
9D00388C  00001012  MFLO V0
9D003890  AFA20064  SW V0, 100(SP)
9D003894  00001810  MFHI V1
9D003898  AFA30060  SW V1, 96(SP)
9D00389C  03A0F021  ADDU S8, SP, ZERO
    
```

Cette série d'instructions correspond à la partie du prologue qui s'occupe principalement de gérer l'état du processeur et sauver les registres RA, T9 à T0 et A3 à A0 sur la pile.

```

76:  PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
9D0038A0  00002021  ADDU A0, ZERO, ZERO
9D0038A4  24050004  ADDIU A1, ZERO, 4
9D0038A8  0F40163D  JAL 0x9D0058F4
9D0038AC  00000000  NOP

77:                                     // Test de la période du timer
78:                                     BSP_LEDToggle(BSP_LED_1);
9D0038B0  24040001  ADDIU A0, ZERO, 1
9D0038B4  0F4013C5  JAL BSP_LEDToggle
9D0038B8  00000000  NOP

79:                                     }
9D0038BC  03C0E821  ADDU SP, S8, ZERO
9D0038C0  8FA20064  LW V0, 100(SP)
9D0038C4  00400013  MTLO V0
9D0038C8  8FA30060  LW V1, 96(SP)
9D0038CC  00600011  MTHI V1
9D0038D0  8FA2006C  LW V0, 108(SP)
9D0038D4  3042000F  ANDI V0, V0, 15
9D0038D8  14400014  BNE V0, ZERO, 0x9D00392C
9D0038DC  00000000  NOP
9D0038E0  8FBF005C  LW RA, 92(SP)
9D0038E4  8FBE0058  LW S8, 88(SP)
9D0038E8  8FB90054  LW T9, 84(SP)
9D0038EC  8FB80050  LW T8, 80(SP)
9D0038F0  8FAF004C  LW T7, 76(SP)
9D0038F4  8FAE0048  LW T6, 72(SP)
9D0038F8  8FAD0044  LW T5, 68(SP)
9D0038FC  8FAC0040  LW T4, 64(SP)
9D003900  8FAB003C  LW T3, 60(SP)
9D003904  8FAA0038  LW T2, 56(SP)
9D003908  8FA90034  LW T1, 52(SP)
9D00390C  8FA80030  LW T0, 48(SP)
9D003910  8FA7002C  LW A3, 44(SP)
9D003914  8FA60028  LW A2, 40(SP)
9D003918  8FA50024  LW A1, 36(SP)
9D00391C  8FA40020  LW A0, 32(SP)
9D003920  8FA3001C  LW V1, 28(SP)
9D003924  8FA20018  LW V0, 24(SP)
9D003928  8FA10014  LW AT, 20(SP)
9D00392C  00000000  NOP
9D003930  41606000  DI ZERO
9D003934  000000C0  EHB
9D003938  8FBA0074  LW K0, 116(SP)
    
```

```

9D00393C  8FBB0070    LW K1, 112(SP)
9D003940  409A7000    MTC0 K0, EPC
9D003944  8FBA006C    LW K0, 108(SP)
9D003948  27BD0078    ADDIU SP, SP, 120
9D00394C  409A6002    MTC0 K0, SRSCtl
9D003950  41DDE800    WRPGR SP, SP
9D003954  409B6000    MTC0 K1, Status
9D003958  42000018    ERET

```

On trouve dans l'épilogue la restauration des registres T9 à T0, ainsi que A3 à A0. La routine se termine par l'instruction **ERET** alors qu'une simple fonction se termine par JR RA.

5.4.5. CONFIGURATION TIMER 4 ET INTERRUPTION

Dans le fichier system_init.c, on trouve dans la fonction DRV_TMR1_Initialize, les actions de configuration du timer 4 et de son interruption.

```

void DRV_TMR1_Initialize(void)
{
    /* Initialize Timer Instance1 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_4);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_4,
                               DRV_TMR_CLKSOURCE_INTERNAL);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_4,
                            TMR_PRESCALE_VALUE_1);
    /* Enable 16 bit mode */
    PLIB_TMR_Model16BitEnable(TMR_ID_4);
    /* Clear counter */
    PLIB_TMR_Counter16BitClear(TMR_ID_4);
    /*Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_4, 8000);
    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T4,
                              INT_PRIORITY_LEVEL7);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T4,
                                  INT_SUBPRIORITY_LEVEL0);
}

```

👉 Comme pour le timer 1, le timer 4 n'est pas activé lors de son initialisation.

5.4.5.1. AUTORISATION DE L'INTERRUPTION LORS DU START

A nouveau, comme pour le timer 1, on observe que dans la fonction DRV_TMR1_Start() il y a appel de la fonction _DRV_TMR1_Resume() qui s'occupe de l'autorisation de la source de l'interruption :

```
static void _DRV_TMR1_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                   INT_SOURCE_TIMER_4);
        PLIB_INT_SourceEnable(INT_ID_0,
                               INT_SOURCE_TIMER_4);
        PLIB_TMR_Start(TMR_ID_4);
    }
}

bool DRV_TMR1_Start(void)
{
    /* Start Timer*/
    _DRV_TMR1_Resume(true);
    DRV_TMR1_Running = true;

    return true;
}
```

5.4.6. ROUTINE RÉPONSE INTERRUPTION DU TIMER 4

Dans le fichier system_interrupt.c, on trouve dans la macro __ISR correspondant au timer 4. La priorité a été réglée en ip17SRS :

```
void __ISR( _TIMER_4_VECTOR, ip17SRS)
           _IntHandlerDrvTmrInstancel(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_4);
    // Test de la periode du timer
    BSP_LEDToggle(BSP_LED_4);
}
```

5.4.6.1. RÉPONSE INTERRUPTION TIMER4 EN ASSEMBLEUR

Voici la réponse à l'interruption du Timer 4 en assembleur :

```

81:                                void __ISR(_TIMER_4_VECTOR, ipl7SRS)
                                IntHandlerDrvTmrInstance1(void)
82:                                {
9D004DBC 415DE800 RDPGPR SP, SP
9D004DC0 401A7000 MFC0 K0, EPC
9D004DC4 401B6000 MFC0 K1, Status
9D004DC8 27BDFFD8 ADDIU SP, SP, -40
9D004DCC AFBB0024 SW K1, 36(SP)
9D004DD0 7C1B7844 INS K1, ZERO, 1, 15
9D004DD4 377B1C00 ORI K1, K1, 7168
9D004DD8 409B6000 MTC0 K1, Status
9D004DDC 00001012 MFLO V0
9D004DE0 AFA20014 SW V0, 20(SP)
9D004DE4 00001810 MFHI V1
9D004DE8 AFA30010 SW V1, 16(SP)
9D004DEC 03A0F021 ADDU S8, SP, ZERO
83: PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_4);
9D004DF0 00002021 ADDU A0, ZERO, ZERO
9D004DF4 24050010 ADDIU A1, ZERO, 16
9D004DF8 0F40163D JAL 0x9D0058F4
9D004DFC 00000000 NOP
84:                                // Test de la periode du timer
85:                                BSP_LEDToggle(BSP_LED_4);
9D004E00 24040006 ADDIU A0, ZERO, 6
9D004E04 0F4013C5 JAL BSP_LEDToggle
9D004E08 00000000 NOP
86:
87:                                }
9D004E0C 03C0E821 ADDU SP, S8, ZERO
9D004E10 8FA20014 LW V0, 20(SP)
9D004E14 00400013 MTLO V0
9D004E18 8FA30010 LW V1, 16(SP)
9D004E1C 00600011 MTHI V1
9D004E20 8FBB0024 LW K1, 36(SP)
9D004E24 27BD0028 ADDIU SP, SP, 40
9D004E28 41DDE800 WRPGR SP, SP
9D004E2C 409B6000 MTC0 K1, Status
9D004E30 42000018 ERET

```

On constate qu'avec la spécification SRS, il y a beaucoup moins de valeurs sauvegardées (puis restaurées) sur la pile. Le prologue et l'épilogue sont bien plus courts !

5.4.7. ACTION POUR FAIRE FONCTIONNER LE TEST

Il suffit de placer les actions suivantes dans APP_Initialize().

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;

    lcd_init();
    lcd_bl_on();
    printf_lcd("Chap5 TestInt      ");
    lcd_gotoxy(1,2);
    printf_lcd("C. Huber 29.11.2016");
    DRV_TMR0_Start();
    DRV_TMR1_Start();
}
```

5.5. LES INTERRUPTIONS EXTERNES

Le PIC32MX795F512L ou H dispose de 5 interruptions externes appelées INT0, INT1, INT2, INT3 et INT4.

Une particularité des interruptions externes est la possibilité de configurer la polarité du flanc qui déclenche la demande d'interruption.

5.5.1. LISTE DES INTERRUPTIONS EXTERNES

Voici les 5 interruptions externes du PIC32MX795F512L :

Nom complet	No pin	Rôles particuliers
SDO1/OC1/INT0/RD0	72	SPI 1 SDO
AERXD0/INT1/RE8	18	
AERXD1/INT2/RE9	19	
AETXCLK/SCL1/INT3/RA14	66	I2C 1 SCL
AETXEN/SDA1/INT4/RA15	67	I2C 1 SDA

5.5.1.1. DÉFINITIONS DES SOURCES

Dans le type énuméré **INT_SOURCE**, les interruptions externes sont définies ainsi :

```

INT_SOURCE_EXTERNAL_0 = 3,
INT_SOURCE_EXTERNAL_1 = 7,
INT_SOURCE_EXTERNAL_2 = 11,
INT_SOURCE_EXTERNAL_3 = 15,
INT_SOURCE_EXTERNAL_4 = 19,

```

5.5.1.2. DÉFINITIONS DES VECTEURS (PLIB_INT)

Dans le type énuméré **INT_VECTOR**, les vecteurs des interruptions externes sont définies ainsi :

```

INT_VECTOR_INT0 = 0x18,
INT_VECTOR_INT1 = 0x38,
INT_VECTOR_INT2 = 0x58,
INT_VECTOR_INT3 = 0x78,
INT_VECTOR_INT4 = 0x98,

```

5.5.1.3. DÉFINITIONS DES VECTEURS (MACRO __ISR)

Dans le fichier p32mx795f512l.h on trouve les définitions des vecteurs à utiliser avec la macro **__ISR**.

```

#define _EXTERNAL_0_VECTOR 3
#define _EXTERNAL_1_VECTOR 7
#define _EXTERNAL_2_VECTOR 11
#define _EXTERNAL_3_VECTOR 15
#define _EXTERNAL_4_VECTOR 19

```

5.5.2. CONFIGURATION DE LA POLARITÉ DU FLANC

La `plib_int` met à disposition deux fonctions pour le choix du flanc montant (rising) ou du flanc descendant (falling).

```
void PLIB_INT_ExternalFallingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
void PLIB_INT_ExternalRisingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
```

☹ Il n'est pas possible de configurer pour obtenir une interruption à tous les flancs (montants et descendants).

5.5.2.1. TYPE ÉNUMÉRÉ INT_EXTERNAL_SOURCES

Voici le type énuméré `INT_EXTERNAL_SOURCES` dont les valeurs servent à établir la valeur du bit correspondant dans le registre `INTCON` :

```
typedef enum {
    INT_EXTERNAL_INT_SOURCE0 = 0x01,
    INT_EXTERNAL_INT_SOURCE1 = 0x02,
    INT_EXTERNAL_INT_SOURCE2 = 0x04,
    INT_EXTERNAL_INT_SOURCE3 = 0x08,
    INT_EXTERNAL_INT_SOURCE4 = 0x10
} INT_EXTERNAL_SOURCES;
```

5.5.2.2. PLIB_INT_EXTERNALRISINGEDGESELECT, EXEMPLE

Dans cet exemple repris de la documentation Harmony (paragraphe "PLIB_INT_ExternalRisingEdgeSelect Function"), on voit la possibilité d'établir une liste des interruptions externes qui doivent réagir au flanc montant en combinant avec un OU bit à bit (`|`).

Example

```
PLIB_INT_ExternalRisingEdgeSelect( INT_ID_0,
                                   ( INT_EXTERNAL_INT_SOURCE0 |
                                     INT_EXTERNAL_INT_SOURCE1 ) );
```

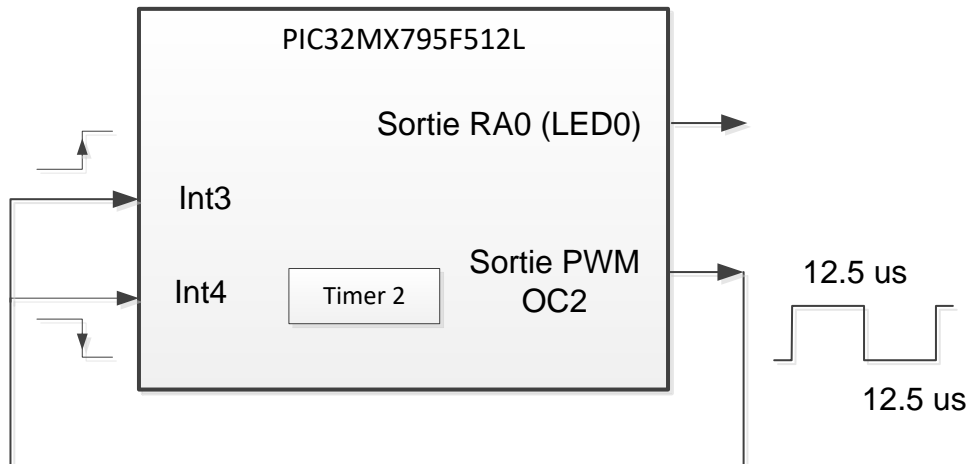
Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be <code>INT_ID_0</code> for all of the devices that have only one Interrupt module).
source	One or more of the values from <code>INT_EXTERNAL_SOURCES</code> . Values can be combined using a bitwise "OR" operation.

5.6. TEMPS DE RÉACTION D'UNE INTERRUPTION EXTERNE

Nous allons compléter l'exemple précédant afin de mettre en œuvre 2 interruptions externes, pour disposer d'un exemple de configuration et d'un timing à l'oscilloscope.

5.6.1. PRINCIPE DE L'EXEMPLE



Utilisation de la sortie OC2 pour fournir un signal. Int3 (priorité 7) doit réagir au flanc montant, int4 (priorité 6) au flanc descendant. Dans la réponse à l'int3 LED0 est mis à 1, alors que dans la réponse à int4 LED0 est mis à 0.

5.6.1.1. CONTRAINTES KIT POUR LA RÉALISATION

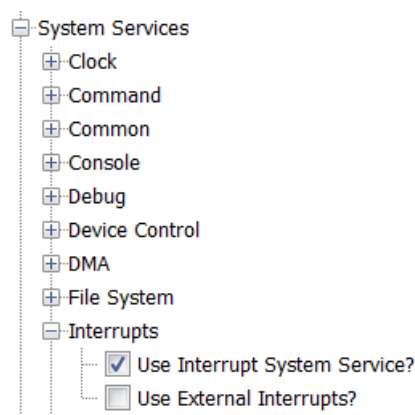


Il est nécessaire de configurer les pins d'interruption externes RA14 et RA15 en entrées pour ne pas avoir de problème avec l'introduction du signal. Il faut câbler la broche 76 (OC2) à 66 et 67.

5.6.2. CONFIGURATION DES INT. EXT. AVEC LE MHC

Voici comment mettre en œuvre les interruptions externes et les configurer avec le Microchip Harmony Configurator.

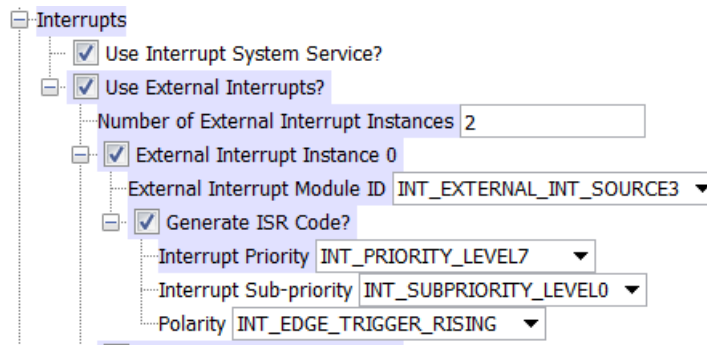
👉 Il s'agit d'utiliser un **System Services** et non pas un driver !



Si on coche Use External Interrupts, il est possible de configurer nos 2 interruptions externes.

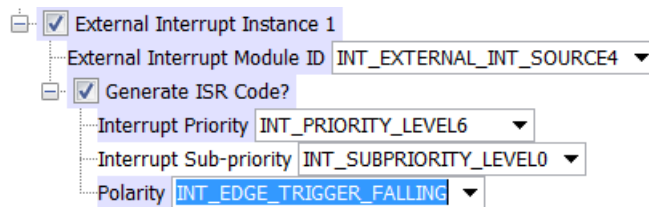
5.6.2.1. CONFIGURATION INT3 (INSTANCE 0)

Priorité 7 et flanc montant :



5.6.2.2. CONFIGURATION INT4 (INSTANCE 1)

Priorité 6 et flanc descendant :



5.6.2.3. MODIFICATION PRIORITÉ INTERRUPTION TIMER 4

Pour éviter d'influencer sur les interruptions externes, il faut réduire la priorité du timer 4 au niveau 5.

5.6.3. CODE GÉNÉRÉ POUR LES INTERRUPTIONS EXT. PAR LE MHC

La configuration des interruptions externes est directement effectuée dans le fichier `system_init.c`. Utilisation des macros `SYS_INT_xxxx`.

Le code est placé après :

```
/** Interrupt Service Initialization Code **/
SYS_INT_Initialize();
```

Et avant :

```
/* Initialize Middleware */

/* Enable Global Interrupts */
SYS_INT_Enable();
```

5.6.3.1. CONFIGURATION INT3

Voici le code généré :

```
/*Setup the INT_SOURCE_EXTERNAL_3 and Enable it*/
SYS_INT_VectorPrioritySet(INT_VECTOR_INT3,
                          INT_PRIORITY_LEVEL7);
SYS_INT_VectorSubprioritySet(INT_VECTOR_INT3,
                             INT_SUBPRIORITY_LEVEL0);
SYS_INT_ExternalInterruptTriggerSet
    (INT_EXTERNAL_INT_SOURCE3, INT_EDGE_TRIGGER_RISING);
SYS_INT_SourceEnable(INT_SOURCE_EXTERNAL_3);
```

5.6.3.2. CONFIGURATION INT4

Voici le code généré :

```
/*Setup the INT_SOURCE_EXTERNAL_4 and Enable it*/
SYS_INT_VectorPrioritySet(INT_VECTOR_INT4,
                          INT_PRIORITY_LEVEL6);
SYS_INT_VectorSubprioritySet(INT_VECTOR_INT4,
                             INT_SUBPRIORITY_LEVEL0);
SYS_INT_ExternalInterruptTriggerSet
    (INT_EXTERNAL_INT_SOURCE4, INT_EDGE_TRIGGER_FALLING);
SYS_INT_SourceEnable(INT_SOURCE_EXTERNAL_4);
```

En plus des 3 actions de base valables pour n'importe quelle interruption, on trouve les fonctions de sélection du flanc.

5.6.4. LES FONCTIONS SYS_INT

Ce sont des macros définies dans le fichier sys_int_mapping.h !

5.6.4.1. LA MACRO SYS_INT_VECTORPRIORITYSET

La macro SYS_INT_VectorPrioritySet utilise **PLIB_INT_VectorPrioritySet**.

```
#define SYS_INT_VectorPrioritySet( vector, priority ) \
    PLIB_INT_VectorPrioritySet( INT_ID_0,vector, priority)
```

5.6.4.2. LA MACRO SYS_INT_VECTORSUBPRIORITYSET

La macro SYS_INT_VectorSubprioritySet utilise **PLIB_INT_VectorSubPrioritySet**.

```
#define SYS_INT_VectorSubprioritySet(vector, subpriority) \
    PLIB_INT_VectorSubPrioritySet(INT_ID_0,vector,subpriority)
```

5.6.4.3. LA MACRO SYS_INT_SOURCEENABLE

La macro SYS_INT_SourceEnable utilise **PLIB_INT_SourceEnable**.

```
#define SYS_INT_SourceEnable( source ) \
    PLIB_INT_SourceEnable( INT_ID_0,source )
```


5.6.4.4. LA MACRO `SYS_INT_EXTERNAL_INTERRUPT_TRIGGERSET`

N'est pas une macro, c'est une fonction implémentée dans `sys_int_pic32.c`.

```
void SYS_INT_ExternalInterruptTriggerSet
    ( INT_EXTERNAL_SOURCES source,
      INT_EXTERNAL_EDGE_TRIGGER edgeTrigger )
{
    if ( edgeTrigger == INT_EDGE_TRIGGER_RISING )
    {
        PLIB_INT_ExternalRisingEdgeSelect ( INT_ID_0,
                                              source );
    } else {
        PLIB_INT_ExternalFallingEdgeSelect ( INT_ID_0,
                                              source );
    }
}
```

5.6.5. RÉPONSES AUX INTERRUPTIONS EXTERNES

Les réponses aux interruptions externes se trouvent dans le fichier `system_interrupt.c`.

5.6.5.1. RÉPONSE INTERRUPTION INT3

La priorité de la routine d'interruption est mise à **IPL7SRS**. L'action sur la Led0 a été placée avant la mise à 0 du Flag d'interruption.

```
void __ISR( _EXTERNAL_3_VECTOR, IPL7SRS)
    _IntHandlerExternalInterruptInstance0(void)
{
    LED0_W = 1;
    PLIB_INT_SourceFlagClear(INT_ID_0,
                              INT_SOURCE_EXTERNAL_3);
}
```

5.6.5.2. RÉPONSE INTERRUPTION INT4

Voici la routine de réponse à l'interruption externe 4 tel que généré avec uniquement l'ajout de l'action sur la Led0.

```
void __ISR( _EXTERNAL_4_VECTOR, IPL6AUTO)
    _IntHandlerExternalInterruptInstance1(void)
{
    LED0_W = 0;
    PLIB_INT_SourceFlagClear(INT_ID_0,
                              INT_SOURCE_EXTERNAL_4);
}
```

5.6.6. ACTION POUR FAIRE FONCTIONNER LE TEST

Il suffit de placer les actions suivantes dans APP_Initialize :

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;

    lcd_init();
    lcd_bl_on();
    printf_lcd("Chap5 TestInt      ");
    lcd_gotoxy(1,2);
    printf_lcd("C. Huber 29.11.2016");
    DRV_TMR0_Start();
    DRV_TMR1_Start();
    DRV_TMR2_Start();
    DRV_OC0_Start();
}
```

5.6.6.1. MODIFICATION POUR LE TIMER2

Comme le Timer 2 sert de base de temps, il n'a pas besoin de générer une interruption.

Pour cela il faut mettre en commentaire l'autorisation de l'interruption dans la fonction _DRV_TMR2_Resume :

```
static void _DRV_TMR2_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                   INT_SOURCE_TIMER_2);
        // PLIB_INT_SourceEnable(INT_ID_0,
                                   INT_SOURCE_TIMER_2);
        PLIB_TMR_Start(TMR_ID_2);
    }
}
```

5.6.7. MESURE DES TEMPS DE RÉACTION

Il faut câbler la broche 76 (OC2) à 66 (int3) et 67 (int4).

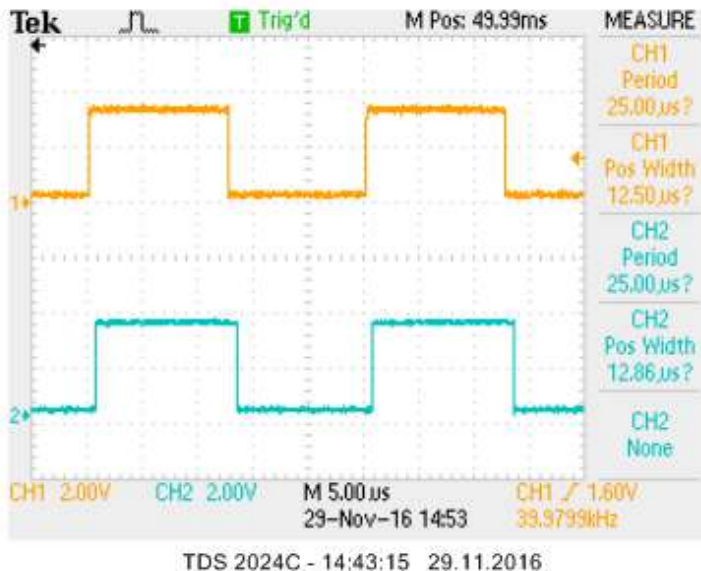
Au niveau du Pin Settings, on constate que le système a ajouté Int3 et Int4 sans configuration de la direction.

66	RA14	5V	INT3	n/a	n/a	<input type="checkbox"/>	Digital
67	RA15	5V	INT4	n/a	n/a	<input type="checkbox"/>	Digital

5.6.7.1. VUE D'ENSEMBLE

Canal1 = OC2
(broche 76)

Canal2 = Led0

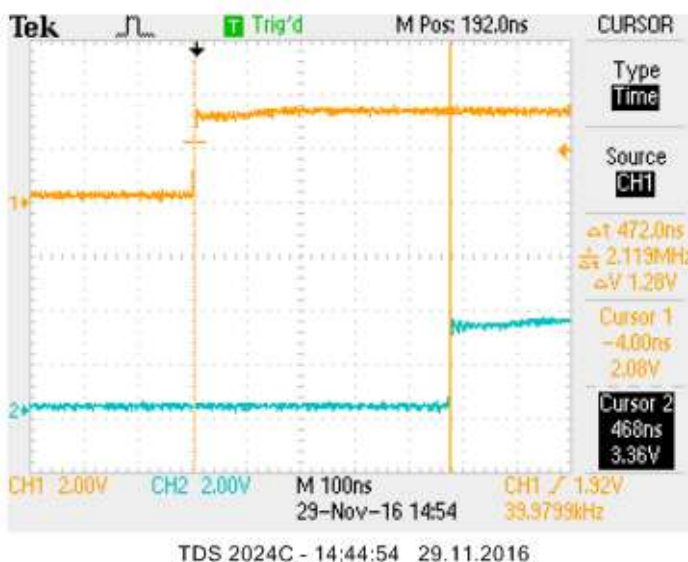


La vue d'ensemble permet de vérifier que le système fonctionne et que la période du signal est bien de 25 μs.

5.6.7.2. TEMPS RÉACTION INT3

Canal1 = OC2
(broche 76)

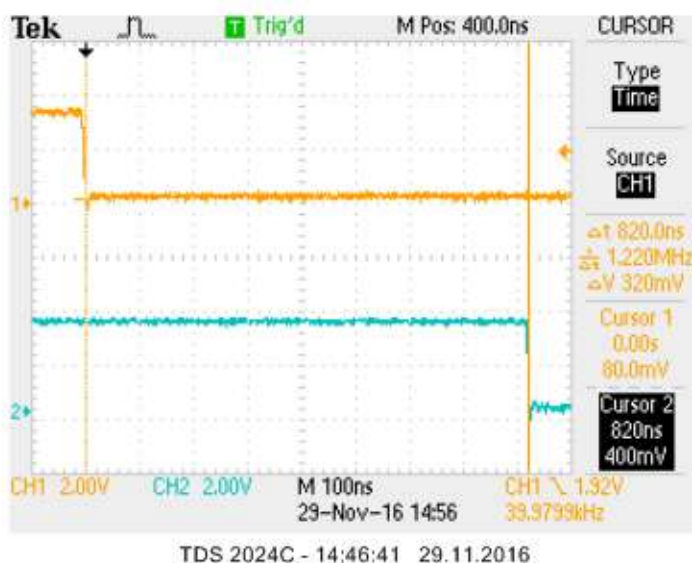
Canal2 = Led0



On obtient un décalage de 472 ns entre le flanc montant du signal d'interruption (canal 1) et le flanc montant du signal sur la led (canal 2).

5.6.7.3. TEMPS RÉACTION INT4

Canal1 = OC2
(broche 76)
Canal2 = Led0



On obtient un décalage de 820 ns entre le flanc montant du signal d'interruption (canal 1) et le flanc montant du signal sur la led (canal 2).

5.6.8. DIFFÉRENCE DANS LES ROUTINES DE RÉPONSE

Pour expliquer la modeste différence, il faut observer le code des routines de réponse en assembleur.

5.6.8.1. ISR INT3 EN ASSEMBLEUR

Voici la réponse à l'interruption externe 3 en assembleur :

```

72:                void __ISR(_EXTERNAL_3_VECTOR, IPL7SRS)
                  _IntHandlerExternalInterruptInstance0(void)
73:                {
9D005740  415DE800  RDPGPR SP, SP
9D005744  401A7000  MFC0 K0, EPC
9D005748  401B6000  MFC0 K1, Status
9D00574C  27BDFFD8  ADDIU SP, SP, -40
9D005750  AFBFB0024  SW K1, 36(SP)
9D005754  7C1B7844  INS K1, ZERO, 1, 15
9D005758  377B1C00  ORI K1, K1, 7168
9D00575C  409B6000  MTC0 K1, Status
9D005760  00001012  MFLO V0
9D005764  AFA20014  SW V0, 20(SP)
9D005768  00001810  MFHI V1
9D00576C  AFA30010  SW V1, 16(SP)
9D005770  03A0F021  ADDU S8, SP, ZERO
74:                LED0_W = 1;
9D005774  3C03BF88  LUI V1, -16504
9D005778  94626020  LHU V0, 24608(V1)
9D00577C  24040001  ADDIU A0, ZERO, 1
9D005780  7C820004  INS V0, A0, 0, 1
9D005784  A4626020  SH V0, 24608(V1)
75:                PLIB_INT_SourceFlagClear(INT_ID_0,
                                           INT_SOURCE_EXTERNAL_3);
9D005788  00002021  ADDU A0, ZERO, ZERO
9D00578C  2405000F  ADDIU A1, ZERO, 15
9D005790  0F40192A  JAL 0x9D0064A8
9D005794  00000000  NOP
76:                }
9D005798  03C0E821  ADDU SP, S8, ZERO
9D00579C  8FA20014  LW V0, 20(SP)
9D0057A0  00400013  MTLO V0
9D0057A4  8FA30010  LW V1, 16(SP)
9D0057A8  00600011  MTHI V1
9D0057AC  8FBFB0024  LW K1, 36(SP)
9D0057B0  27BD0028  ADDIU SP, SP, 40
9D0057B4  41DDE800  WRPGR SP, SP
9D0057B8  409B6000  MTC0 K1, Status
9D0057BC  42000018  ERET
    
```

5.6.8.2. ISR INT4 EN ASSEMBLEUR

Voici la réponse à l'interruption externe 3 en assembleur :

```

78:                void __ISR(_EXTERNAL_4_VECTOR, IPL6AUTO)
                  _IntHandlerExternalInterruptInstance1(void)
79:                {
9D003800  415DE800  RDPGPR SP, SP
9D003804  401B7000  MFC0 K1, EPC
9D003808  401A6002  MFC0 K0, SRSCtl
9D00380C  27BDF88  ADDIU SP, SP, -120
9D003810  AFBB0074  SW K1, 116(SP)
9D003814  401B6000  MFC0 K1, Status
9D003818  AFBA006C  SW K0, 108(SP)
9D00381C  AFBB0070  SW K1, 112(SP)
9D003820  7C1B7844  INS K1, ZERO, 1, 15
9D003824  377B1800  ORI K1, K1, 6144
9D003828  409B6000  MTC0 K1, Status
9D00382C  AFA3001C  SW V1, 28(SP)
9D003830  AFA20018  SW V0, 24(SP)
9D003834  8FA3006C  LW V1, 108(SP)
9D003838  3063000F  ANDI V1, V1, 15
9D00383C  14600012  BNE V1, ZERO, 0x9D003888
9D003840  00000000  NOP
9D003844  AFBF005C  SW RA, 92(SP)
9D003848  AFBE0058  SW S8, 88(SP)
9D00384C  AFB90054  SW T9, 84(SP)
9D003850  AFB80050  SW T8, 80(SP)
9D003854  AFAF004C  SW T7, 76(SP)
9D003858  AFAD0048  SW T6, 72(SP)
9D00385C  AFAD0044  SW T5, 68(SP)
9D003860  AFAC0040  SW T4, 64(SP)
9D003864  AFAB003C  SW T3, 60(SP)
9D003868  AFAD0038  SW T2, 56(SP)
9D00386C  AFA90034  SW T1, 52(SP)
9D003870  AFA80030  SW T0, 48(SP)
9D003874  AFA7002C  SW A3, 44(SP)
9D003878  AFA60028  SW A2, 40(SP)
9D00387C  AFA50024  SW A1, 36(SP)
9D003880  AFA40020  SW A0, 32(SP)
9D003884  AFA10014  SW AT, 20(SP)
9D003888  00000000  NOP
9D00388C  00001012  MFLO V0
9D003890  AFA20064  SW V0, 100(SP)
9D003894  00001810  MFHI V1
9D003898  AFA30060  SW V1, 96(SP)
9D00389C  03A0F021  ADDU S8, SP, ZERO
80:                LED0_W = 0;
9D0038A0  3C03BF88  LUI V1, -16504
9D0038A4  94626020  LHU V0, 24608(V1)
9D0038A8  7C020004  INS V0, ZERO, 0, 1
9D0038AC  A4626020  SH V0, 24608(V1)
81:                PLIB_INT_SourceFlagClear(INT_ID_0,
                                           INT_SOURCE_EXTERNAL_4);
9D0038B0  00002021  ADDU A0, ZERO, ZERO

```

```

9D0038B4 24050013 ADDIU A1, ZERO, 19
9D0038B8 0F40192A JAL 0x9D0064A8
9D0038BC 00000000 NOP
82:      }
9D0038C0 03C0E821 ADDU SP, S8, ZERO
9D0038C4 8FA20064 LW V0, 100(SP)
9D0038C8 00400013 MTLO V0
9D0038CC 8FA30060 LW V1, 96(SP)
9D0038D0 00600011 MTHI V1
9D0038D4 8FA2006C LW V0, 108(SP)
9D0038D8 3042000F ANDI V0, V0, 15
9D0038DC 14400014 BNE V0, ZERO, 0x9D003930
9D0038E0 00000000 NOP
9D0038E4 8FBF005C LW RA, 92(SP)
9D0038E8 8FBE0058 LW S8, 88(SP)
9D0038EC 8FB90054 LW T9, 84(SP)
9D0038F0 8FB80050 LW T8, 80(SP)
9D0038F4 8FAF004C LW T7, 76(SP)
9D0038F8 8FAE0048 LW T6, 72(SP)
9D0038FC 8FAD0044 LW T5, 68(SP)
9D003900 8FAC0040 LW T4, 64(SP)
9D003904 8FAB003C LW T3, 60(SP)
9D003908 8FAA0038 LW T2, 56(SP)
9D00390C 8FA90034 LW T1, 52(SP)
9D003910 8FA80030 LW T0, 48(SP)
9D003914 8FA7002C LW A3, 44(SP)
9D003918 8FA60028 LW A2, 40(SP)
9D00391C 8FA50024 LW A1, 36(SP)
9D003920 8FA40020 LW A0, 32(SP)
9D003924 8FA3001C LW V1, 28(SP)
9D003928 8FA20018 LW V0, 24(SP)
9D00392C 8FA10014 LW AT, 20(SP)
9D003930 00000000 NOP
9D003934 41606000 DI ZERO
9D003938 000000C0 EHB
9D00393C 8FBA0074 LW K0, 116(SP)
9D003940 8FBB0070 LW K1, 112(SP)
9D003944 409A7000 MTC0 K0, EPC
9D003948 8FBA006C LW K0, 108(SP)
9D00394C 27BD0078 ADDIU SP, SP, 120
9D003950 409A6002 MTC0 K0, SRSCtl
9D003954 41DDE800 WRPGR SP, SP
9D003958 409B6000 MTC0 K1, Status
9D00395C 42000018 ERET
    
```

On remarque que la réponse Int3 avec le SRS effectue un minimum de sauvegarde, tandis que pour l'int4 en mode AUTO il y a des sauvegardes assez complètes.

5.6.9. CONCLUSION SUR LES INTERRUPTIONS EXTERNES

On constate que le temps de réaction entre le flanc qui déclenche l'interruption et le basculement de la sortie est inférieur à 500 ns dans le cas des routines de réponse utilisant le SRS et de l'ordre de 800 ns en mode AUTO.

5.6.9.1. FRÉQUENCE LIMITE

En supposant que le temps pour achever la réponse à l'interruption est aussi de l'ordre de 500 ns, on peut en conclure que si l'on applique un signal de 1 MHz le système sera entièrement occupé à entrer et sortir de l'interruption. Si on veut obtenir un traitement pour d'autres éléments, il est ainsi raisonnable de ne pas dépasser les 200 kHz.

5.6.9.2. SITUATION FAVORABLE

Les deux routines de réponse aux interruptions externes représentent un cas favorable, il est difficile d'avoir moins d'actions.

A partir du moment où on appelle une fonction dans la routine, les temps de traitement vont s'allonger à cause de sauvegarde plus nombreuses sur la pile.

5.7. CONCLUSION

Ce chapitre offre un aperçu du mécanisme de traitement des interruptions du PIC32MX. Seul le mode multi-vecteur a été présenté au niveau du code assembleur.

La présentation succincte des fonctions de la `plib_int` et les quelques exemples devraient apporter aux étudiants la capacité à configurer et réaliser une routine de réponse à une interruption externe ou autre.

L'exemple pratique avec la mesure du temps de réaction permet d'avoir une idée des performances du PIC32MX dans ce domaine.

5.8. HISTORIQUE DES VERSIONS

5.8.1. VERSION 1.0 FÉVRIER 2014

Création du document et découverte du mécanisme des interruptions du PIC32MX.

5.8.2. VERSION 1.5 DÉCEMBRE 2014

Complément avec des aspects notions générales. Adaptation à la PLIB_INT de Harmony. Cas des interruptions externes et exemple pratique.

5.8.3. VERSION 1.7 DÉCEMBRE 2015

Adaptation à la PLIB_INT de Harmony 1.06 et à l'évolution du MHC en particulier pour la configuration des interruptions externes.

5.8.4. VERSION 1.8 NOVEMBRE 2016

Maj info documentation. Adaptation à la PLIB_INT de Harmony 1.08 et à l'évolution du MHC. Eléments pratiques sur la base de la même application qui évolue pour la gestion des interruptions externes.

5.8.5. VERSION 1.9 NOVEMBRE 2017

Reprise et relecture par SCA.