LES

YVES TIEC

Version provisoire. Nous travaillons sur ce document, mais les remarques sont les bienvenues !

MOTIVATION

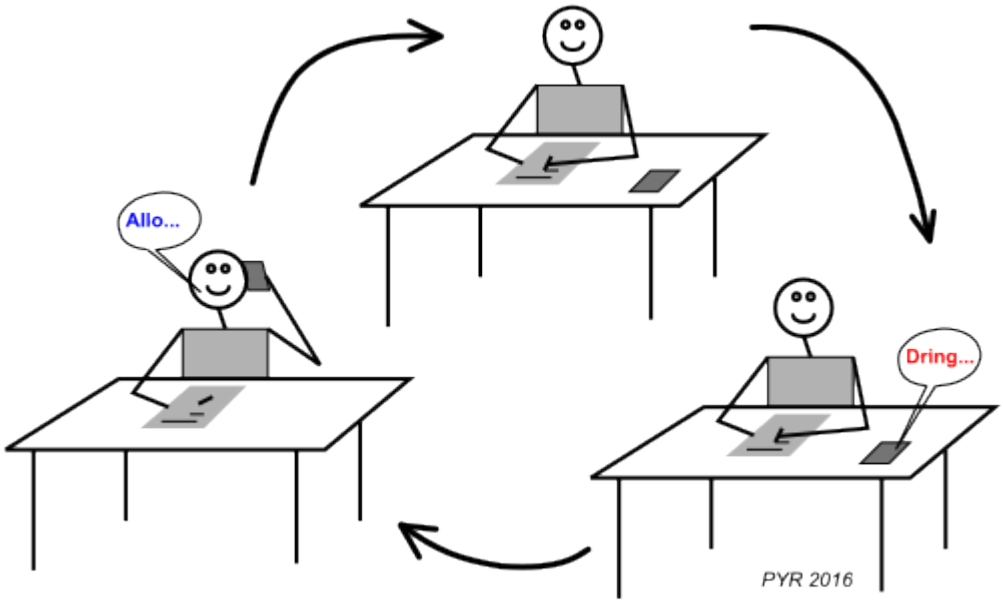
Un système à microcontrôleur est généralement pourvu d'entrées et de sorties. Le but premier du p

Les enseignes et afficheurs à LED sont plutôt une exception dans ce domaine. Beaucoup d'enseigne

Il existe toutefois des cas où une enseigne ou un afficheur doit réagir à des entrées. Par exemple, u
autre cas où le système doit réagir à un événement est la gestion du temps dans un afficheur multip

DÉFINITION

On appelle **interruption** dans un système informatique l'arrêt temporaire d'un programme au prof
courantes. Prenons un exemple : je suis en train de travailler à mon bureau. Le téléphone sonne. Je v



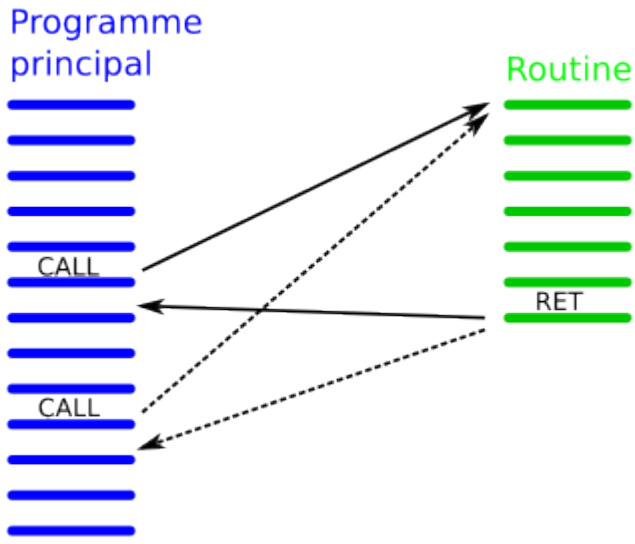
{ width=14cm }

C'est toujours un *événement* qui va produire une interruption. Cet événement a un caractère imprév

Implémentation

Pour utiliser les interruptions, il n'est pas indispensable de comprendre en détail le mécanisme qui mécanismes matériels qui les rendent possibles.

Toutefois, nous allons ici faire une petite incursion dans le monde de la programmation en assemb notées en bleu. Dans le cas où une fraction du programme doit s'exécuter plusieurs fois, on a l'habi correspond aux procédures et aux fonctions dans les langages évolués comme le C.

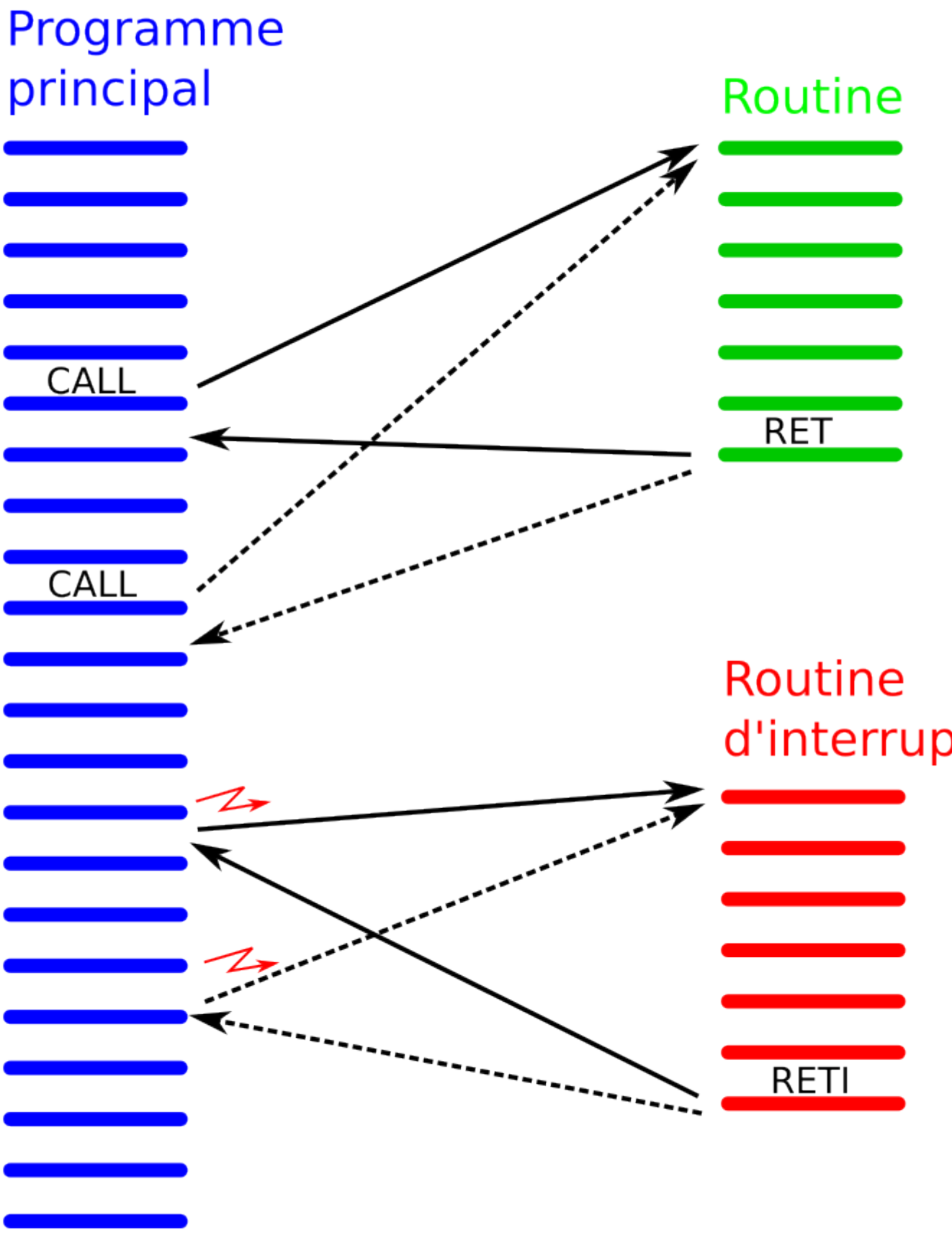


{ width=7cm }

L'appel de la routine se fait par une instruction **Call** dans le programme principal. A la fin de l'exécut
routine peut être appelée plusieurs fois dans le programme principal.

Notons que l'adresse de retour doit être mémorisée pour que le retour soit possible. C'est une **pile**

Regardons maintenant la figure suivante. Un nouvelle routine, appelée **Routine d'interruption** est


Programme
principal
Routine
CALL
RET
CALL
Routine
d'interrup
RETI

Rien dans le programme principal ne permet de savoir que cette routine va s'exécuter. C'est un **évé**

La routine d'interruption se termine aussi par une instruction de retour, appelée **Reti** (*return from in*
avant l'interruption.

Nature des événements

Quels sont ces événements qui vont produire une interruption ? Il en existe principalement deux so

- Les événements **extérieurs** au microcontrôleur. Il s'agit par exemple d'un changement sur une en
- Les événements **intérieurs** au microcontrôleur. Par exemple, beaucoup de microcontrôleurs sont
  fanion et dure un certain temps. Plutôt que d'attendre la fin de la conversion, le programme princ

Dans cette catégorie des interruptions intérieures au microcontrôleur, les plus importantes sont cell
à LED.

Discrimination des sources d'interruption

Il existe généralement plusieurs sources interruptions sur un microcontrôleur. Lorsqu'une interrup
doit consulter les registres pour chaque interruption, pour connaître celle qui a été activée.

Les **vecteurs d'interruption** (*interrupt vectors*) permettent d'être plus efficace : une adresse différe

Souvent ces deux mécanismes vont être utilisés successivement, comme nous le verrons plus bas lo

Voici la table résumée des vecteurs d'interruption pour un MSP430G, y compris l'adresse pour le Re

- 0xFFFE : Reset
- 0xFFFC : NMI
- 0xFFFA : Timer1 CCR0
- 0xFFF8 : Timer1 CCR1, CCR2, TAIFG
- 0xFFF6 : Comparator_A
- 0xFFF4 : Watchdog Timer
- 0xFFF2 : Timer0 CCR0
- 0xFFF0 : Timer0 CCR1, CCR2, TAIFG
- 0xFFEE : USCI status
- 0xFFEC : USCI receive/transmit
- 0xFFEA : ADC10
- 0xFFE8 : -

– 0xFFE6 : Port P2
– 0xFFE4 : Port P1

Les adresses se trouvent en mémoire flash, ce sont les dernières adresses de l'espace d'adressage d

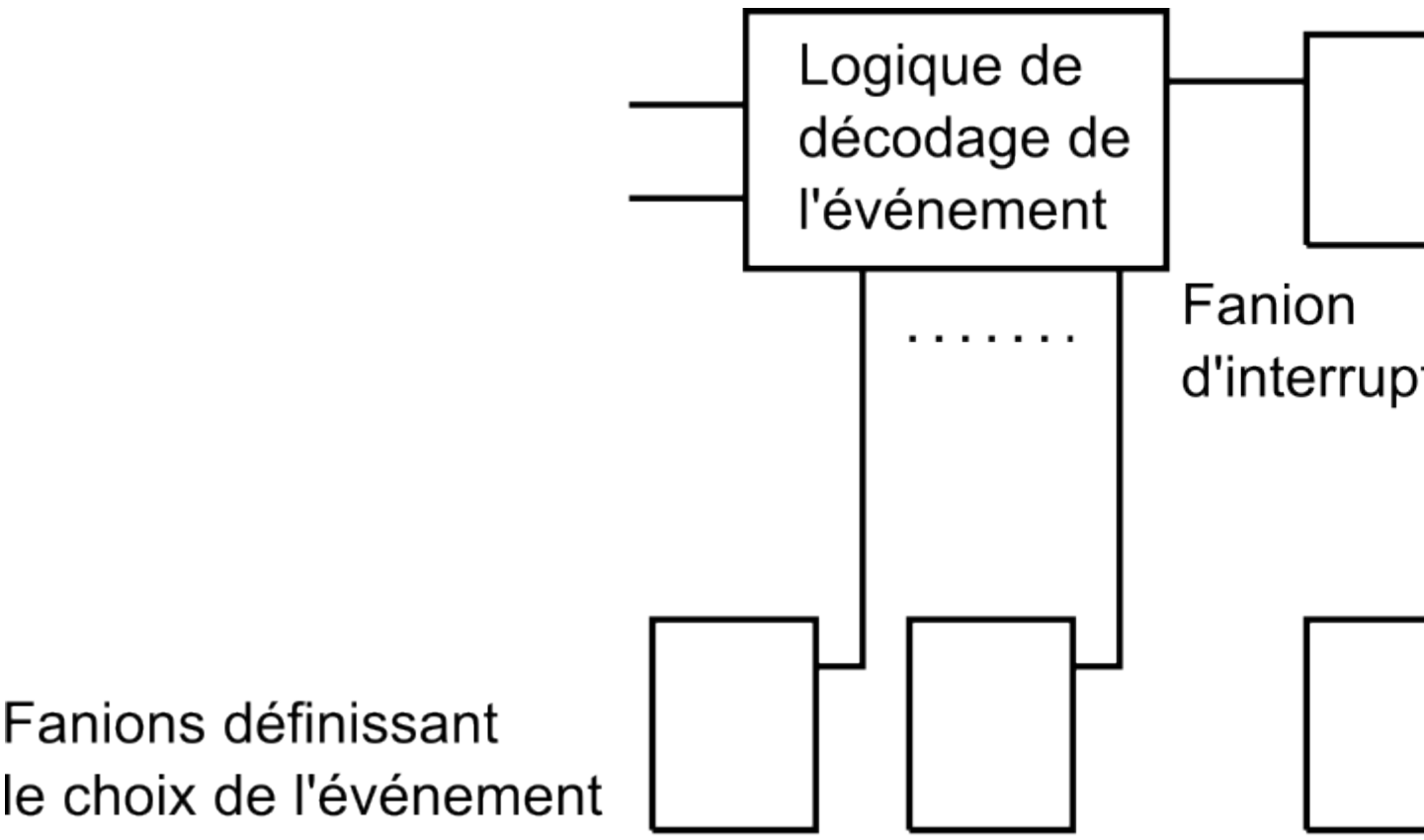
Plusieurs sources d'interruptions nécessitent la scrutation pour déterminer la cause exacte de l'inter
des interruptions des Timers : les registre de comparaison 1 et 2, ainsi que l'interruption générale du

Mise en œuvre d'une interruption

Plusieurs étapes sont nécessaire pour mettre en œuvre une routine d'interruption :

1. Enclencher l'interruption qui nous intéresse. Par exemple une interruption sur une entrée.
2. Préciser comment cette interruption doit fonctionner. Par exemple dire sur quel flanc l'int
3. Enclencher globalement les interruptions. Les microprocesseurs disposent d'un fanion gé
programme, qui ne doivent pas être interrompues.

Le schéma logique ci-dessous montre la logique qui permet de générer les interruptions et les fanio



{ width=17cm }

On y trouve :

– la logique qui permet de saisir un événement

- les fanions qui règlent la manière dont l'événement est décodé
- le fanion qui enclenche cette interruption particulière
- la porte ET associée à ce fanion pour produire cette interruption
- la porte OU qui permet à toutes les interruptions d'être prises en compte
- le fanion général d'autorisation des interruptions
- la porte ET qui produit finalement les interruptions.

Syntaxe des routines d'interruptions

Le langage C ne définit pas la syntaxe des routines d'interruptions. Plusieurs notations sont utilisées

```
#pragma vector=NUMERO_DU_VECTEUR
__interrupt void Nom_de_la_routine (void) {
  ...
}
```

La première ligne indique au compilateur à quel vecteur d'interruption la routine sera associée. La s
__interrupt permet au compilateur d'utiliser les instructions correspondant à une routine d'interru

Interruption produite par une entrée

Sur les microcontrôleurs MSP430, plusieurs registres permettent de définir la manière dont une bro
sont disponibles : P1 et P2. On connaît déjà les registres suivant :

- ***P1DIR*** : détermine le rôle de la broche (entrée ou sortie)
- ***P1OUT*** : donne la valeur pour les broches de sortie
- ***P1IN*** : permet de lire la valeur des entrées
- ***P1REN*** : permet d'enclencher une résistance de tirage (pull-up ou pull-down, selon l'état de bit de P

Pour mettre en œuvre les interruptions sur des broches du port P1, trois registres supplémentaires

- ***P1IE*** : (*Interrupt Enable*) permet l'enclenchement de l'interruption pour chaque bit. L'usage habitu
- ***P1IES*** : (*Interrupt Edge Select*) permet de choisir pour chaque bit le flanc qui va produire l'interrupt
  est d'écrire dans ce registre pour choisir quel flanc va causer une interruption, pour chaque bit.
- ***P1IFG*** : (*Interrupt FlaG*) les **fanions d'interruption**. Lorsque qu'un transition telle qu'elle est spécif

Voici un programme qui met en œuvre une interruption sur l'entrée *P1.3* (le poussoir du Launchpad)

```c
int main() {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P1DIR |= (1<<6); // Led verte en sortie
    P1OUT |= (1<<3); P1REN |= (1<<3); //pull-up sur l'entrée P1.3

    P1IES |= (1<<3); // Mode d'interruption : sur le flanc descendant
    P1IE |= (1<<3); // Interruption P1 activée sur le bit 3
    P1IFG &=~(1<<3); // Fanion d'interruption remis à zéro
    __enable_interrupt(); // General Interrupt Enable

    while(1) { // il n'y a rien à faire dans la boucle principale !
    }
}

// Routine d'interruption associée au Port P1
// Syntaxe spécifique pour les interruptions :
#pragma vector=PORT1_VECTOR
__interrupt void Port1_ISR(void) {
  P1IFG &= ~(1<<3); // Fanion d'interruption correspondant au bit 3 remis à 0
  P1OUT ^= (1<<6); // inverse P1.6 (LED verte)
}
```

La première remarque, c'est que la boucle principale *while(1)...* ne fait rien ! En plus des initialisa
interruption :

– L'activation d'un bit dans *P1IES* sélectionne le flanc descendant
– L'activation d'un bit dans *P1IE* autorise l'interruption sur l'entrée *P1.3*
– L'appel de la procédure *__enable_interrupt()* autorise globalement les interruptions sélectionné

La mise à 0 du bit 3 dans le registre des fanions d'interruption *P1IFG* évite qu'un flanc sur l'entrée *P1*

La routine d'interruption a été placée à la suite du programme principal, alors que nous avons l'habi

Scrutation du bit concerné

Dans notre exemple, seul le bit 3 est concerné par les interruptions. Dans la routine d'interruption, i

Lorsque l'interruption peut provenir de plusieurs entrées, il est alors nécessaire de scruter le registr

```c
int main() {
  ...
  P1IES &=~((1<<3)|(1<<4)); // interruptions aux flancs montants
  P1IE |= (1<<3)|(1<<4); // Interruption activée sur 2 entrées
  P1IFG &=~((1<<3)|(1<<4)); // Fanions d'interruption remis à zéro
  ...

#pragma vector=PORT1_VECTOR
```

```
__interrupt void Port1_ISR(void) {
  // scrutation des causes possible de l'interruption :
  if (P1IFG & (1<<3)) {... ; P1IFG &= ~(1<<3); }
  if (P1IFG & (1<<4)) {... ; P1IFG &= ~(1<<4); }
}
```

INTERRUPTION DE FIN DE CONVERSION

Voici un autre exemple d'interruption, où l'événement est interne au microcontrôleur. L'interruption

```
int main() {
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1DIR |= (1<<6); P1OUT &=~(1<<6); // LED verte en sortie
  // Activation du convertisseur ADC 10 bits (ADC10) :
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10CTL1 = INCH_1; // Canal 1 = entrée A1 = P1.1
  ADC10AE0 |= (1<<1); // Enclanchement de l'entrée A1
  __enable_interrupt(); // General Interrupt Enable
  ADC10CTL0 |= ENC + ADC10SC; // lance une première conversion

  while(1) { // il n'y a rien à faire dans la boucle principale !
  }
}

// Routine d'interruption associée à la fin de conversion ADC
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
  uint16_t val = ADC10MEM; // lit le résultat de la conversion
  ADC10CTL0 |= ENC + ADC10SC; // lance la conversion suivante
  if (val > 512) { // Montre sur la LED verte si la valeur dépasse Vcc/2
    P1OUT |= (1<<6); // LED verte On
  } else {
    P1OUT &=~(1<<6); // LED verte Off
  }
}
```