TDx : Interruptions

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

29 janvier 2021

Principe d'une interruption

- un programme (C→assembleur→opcode) s'exécute séquentiellement suivant la position de l'instruction mémorisée dans le *Program Counter* (PC)
- un évenèment qui ne peut attendre survient et nécessite une réaction "immédiate" du processeur
- une **interruption** est une rupture de l'exécution séquentielle du programme pour gérer un évènement (matériel ou logiciel)
- méthode de programmation peu appréciée car "difficile" à déverminer mais fondamentale pour utiliser efficacement le microcontrôleur
- un périphérique active l'interruption associée en définissant le bit correspondant
- gestion d'un évènement asynchrone à l'exécution du programme
- évite de perdre du temps en attendant (polling) et permet d'effectuer des calculs ou de dormir pendant ce temps (économie d'énergie)

Exemple : un interrupteur indiquant qu'un robot a touché un obstacle et ses moteurs doivent être immédiatement coupés

External Interrupt Control Register A – EICRA

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits**

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in the below table. Edges on INT3..INT0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in the below table will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Note: 1. n = 3, 2, 1, or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

```
00000000 <__vectors>:
   0:   0c 94 56 00   jmp     0xac     ; 0xac <__ctors_end>
   4:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
   8:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
   c:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  10:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
...
  2c:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  30:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  34:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  38:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  3c:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
  40:   0c 94 6a 00   jmp     0xd4     ; 0xd4 <__vector_16>
  44:   0c 94 68 00   jmp     0xd0     ; 0xd0 <__bad_interrupt>
...
000000ac <__ctors_end>:
  ac:   11 24         eor     r1, r1
  ae:   1f be         out     0x3f, r1        ; 63
  b0:   cf ef         ldi     r28, 0xFF       ; 255
  b2:   da e0         ldi     r29, 0x0A       ; 10
  b4:   de bf         out     0x3e, r29       ; 62
  b6:   cd bf         out     0x3d, r28       ; 61
...
000000d0 <__bad_interrupt>:
  d0:   0c 94 00 00   jmp     0       ; 0x0 <__vectors>

000000d4 <__vector_16>:
  d4:   1f 92         push    r1
  d6:   0f 92         push    r0
  d8:   00 90 5f 00   lds     r0, 0x005F
...
00000126 <main>:
 126:   cf 93         push    r28
 128:   df 93         push    r29
 12a:   cd b7         in      r28, 0x3d       ; 61
...
```

avr-gcc -mmcu=atmega32u4 -o t t.c
avr-objdump -d t

Vecteurs d'interruption

```
#include <avr/io.h>
#include <avr/interrupt.h>
volatile int res;
ISR(TIMER1_CAPT_vect) {res=ICR1;TCNT1=0;}
int main() {}
```

Interrupt Vectors in ATmega16U4/ATmega32U4

Table 9-1. Reset and Interrupt Vectors(cont'd)

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	Reserved	Reserved
7	$000C	Reserved	Reserved
8	$000E	INT6	External Interrupt Request 6
9	$0010	Reserved	Reserved
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	USB General	USB General Interrupt request
12	$0016	USB Endpoint	USB Endpoint Interrupt request
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	Reserved	Reserved
15	$001C	Reserved	Reserved
16	$001E	Reserved	Reserved
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A

```
00000000 <__vectors>:
   0:   0c 94 56 00  jmp     0xac    ; 0xac <__ctors_end>
   4:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
   8:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
   c:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  10:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
...
  2c:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  30:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  34:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  38:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  3c:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
  40:   0c 94 6a 00  jmp     0xd4    ; 0xd4 <__vector_16>
  44:   0c 94 68 00  jmp     0xd0    ; 0xd0 <__bad_interrupt>
...
000000ac <__ctors_end>:
  ac:   11 24       eor     r1, r1
  ae:   1f be       out     0x3f, r1        ; 63
  b0:   cf ef       ldi     r28, 0xFF       ; 255
  b2:   da e0       ldi     r29, 0x0A       ; 10
  b4:   de bf       out     0x3e, r29       ; 62
  b6:   cd bf       out     0x3d, r28       ; 61
...
000000d0 <__bad_interrupt>:
  d0:   0c 94 00 00  jmp     0       ; 0x0 <__vectors>

000000d4 <__vector_16>:
  d4:   1f 92       push    r1
  d6:   0f 92       push    r0
  d8:   00 90 5f 00  lds     r0, 0x005F
...
00000126 <main>:
 126:   cf 93       push    r28
 128:   df 93       push    r29
 12a:   cd b7       in      r28, 0x3d       ; 61
...
```

avr-gcc -mmcu=atmega32u4 -o t t.c
avr-objdump -d t

Vecteurs d'interruption

```
00000000 <__ctors_end>:
   0:   21 e0           ldi     r18, 0x01       ; 1
   2:   a0 e0           ldi     r26, 0x00       ; 0
   4:   b1 e0           ldi     r27, 0x01       ; 1
   6:   01 c0           rjmp    .+2             ; 0xa <.do_clear_bss_sta

00000008 <.do_clear_bss_loop>:
   8:   1d 92           st      X+, r1

0000000a <.do_clear_bss_start>:
   a:   a2 30           cpi     r26, 0x02       ; 2
   c:   b2 07           cpc     r27, r18
   e:   e1 f7           brne    .-8             ; 0x8 <.do_clear_bss_loo

00000010 <__vector_31>:
  10:   1f 92           push    r1
  12:   0f 92           push    r0
  14:   00 90 5f 00     lds     r0, 0x005F      ; 0x80005f <__TEXT_REGIO
  18:   0f 92           push    r0
...
00000062 <main>:
  62:   cf 93           push    r28
  64:   df 93           push    r29
  66:   cd b7           in      r28, 0x3d       ; 61
  68:   de b7           in      r29, 0x3e       ; 62
...
```

avr-gcc -mmcu=atmega32u4 -**nostartfiles** -o t t.c
avr-objdump -d t

PAS de définition des vecteurs d'interruption
PAS de définition du SP (pile)

(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C (0x5C)	Reserved	-	-	-	-	-	-	-	-
0x3B (0x5B)	RAMPZ	-	-	-	-	-	-	RAMPZ1	RAMPZ0

Sauvegarde du contexte : importance de la pile

- Le Program Counter (PC) indique quelle est l'instuction en cours d'exécution
- Évènement → lecture de PC dans le vecteur d'interruption → on exécute l'ISR (Interrupt Service Routines)
- les registres vont être écrasés lors de l'exécution de l'ISR et doivent être mémorisés pour être restaurés avant retour au programme principal (`RETI`)
- `push` et `pop` pour empiler et dépiler registres utilisés dans ISR
- `push` = stocker un registre à SP puis **décrémenter** SP ⇒ la pile se trouve au sommet de la RAM
- Définition de l'architecture mémoire dans le *linker script* (.ld)
- Erreur classique : exagérer la taille de RAM disponible place la pile en dehors de la mémoire

Voir `/usr/lib/avr/lib/ldscripts/*.x` :

```
avr-gcc -mmcu=atmega32u4 -Wl,--verbose t.c
...
opened script file /usr/lib/avr/bin/../lib/ldscripts/avr5.xn
OUTPUT_ARCH(avr:5)
__TEXT_REGION_ORIGIN__ = DEFINED(__TEXT_REGION_ORIGIN__) ? __TEXT_REGION_ORIGIN__ : 0;
__DATA_REGION_ORIGIN__ = DEFINED(__DATA_REGION_ORIGIN__) ? __DATA_REGION_ORIGIN__ : 0x800060;
__TEXT_REGION_LENGTH__ = DEFINED(__TEXT_REGION_LENGTH__) ? __TEXT_REGION_LENGTH__ : 128K;
__DATA_REGION_LENGTH__ = DEFINED(__DATA_REGION_LENGTH__) ? __DATA_REGION_LENGTH__ : 0xffa0;
__EEPROM_REGION_LENGTH__ = DEFINED(__EEPROM_REGION_LENGTH__) ? __EEPROM_REGION_LENGTH__ : 64K;
...
MEMORY
{
  text   (rx)   : ORIGIN = __TEXT_REGION_ORIGIN__, LENGTH = __TEXT_REGION_LENGTH__
  data   (rw!x) : ORIGIN = __DATA_REGION_ORIGIN__, LENGTH = __DATA_REGION_LENGTH__
...
```

SEI/CLI

- Chaque périphérique qui utilise les interruptions est configuré incluant l'activation des interruptions

17.2.1 SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global Interrupt Enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- Une interruption se déclenchant en cours de configuration amènerait dans un état instable
- bit global d'activation/désactivation d'interruption : Set Interrupt/Clear Interrupt

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- `#define sei() __asm__ __volatile__ ("sei" ::: "memory")` dans `/usr/lib/avr/include/avr/interrupt.h`

ISR courtes

Sur architecture AVR, les interruptions sont désactivées pendant la gestion d'une ISR ⇒ risque de perte de nouveaux évènements

Une ISR doit être la plus courte possible : acquitter interruption, action immédiate et définir un drapeau (flag) qui sera géré dans le programme principal.
Le drapeau est préfixé de `volatile`.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction - RETI - is executed.

Déclenchement asynchrone de l'ISR ⇒ pas de passage de paramètre(s) ⇒ les variables globales (déclarées hors fonctions) sont **exceptionnellement** autorisées pour échanger entre ISR et programme principal.

```
volatile int global_index;
volatile unsigned char global_tab[NB_CHARS];

void USART1_IRQHandler(void) // VERSION LIBSTM32
{if( USART_GetITStatus(USART1, USART_IT_RXNE))
   {USART_ClearITPendingBit(USART1,USART_IT_RXNE);
    USART_ClearFlag(USART1,USART_IT_RXNE);
    global_tab[global_index]=USART_ReceiveData(USART1);
    if (global_index<(NB_CHARS-1)) global_index++;
   }
}
```

Chronogramme/latence

Le temps de réaction entre le déclenchement de l'ISR associé est crucial.

Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is five clock cycles minimum. After five clock cycles the program vector address for the actual interrupt handling routine is executed. During these five clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by five clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.
A return from an interrupt handling routine takes five clock cycles. During these five clock cycles, the Program Counter (three bytes) is popped back from the Stack, the Stack Pointer is incremented by three, and the I-bit in SREG is set.

Problème classique de rupture de *pipeline* lors des sauts : les instructions en attente doivent être retirées

Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 4-5 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 4-5. The Parallel Instruction Fetches and Instruction Executions

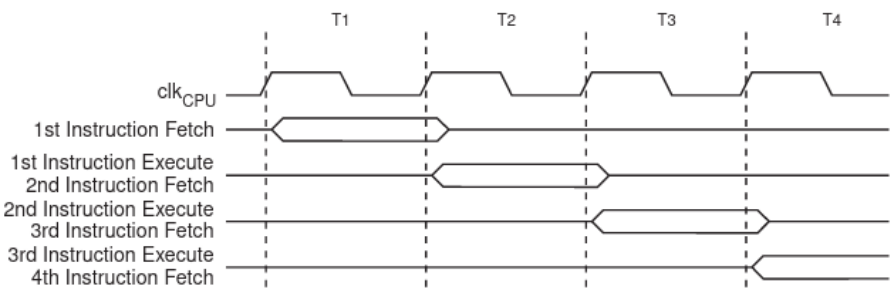

Figure 4-6 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Exemples

- Fondamental pour l'utilisation efficace des ressources
- Linux : interruption timer pour l'ordonnanceur de tâches

```
jmfriedt@rugged:~$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:         10          0          0          0  IR-IO-APIC   2-edge      timer
  1:     332537          0          0      61870  IR-IO-APIC   1-edge      i8042
  8:          0          1          0          0  IR-IO-APIC   8-edge      rtc0
  9:    2126565     526530          0          0  IR-IO-APIC   9-fasteoi   acpi
 12:     526429          0       9771          0  IR-IO-APIC  12-edge      i8042
 16:       5774        928          0          0  IR-IO-APIC  16-fasteoi   ehci_hcd:usb1, mmc0
 17:         14          0          0          0  IR-IO-APIC  17-fasteoi   firewire_ohci
 18:          0          0          0          0  IR-IO-APIC  18-fasteoi   i801_smbus
 19:          0          0          0          0  IR-IO-APIC  19-fasteoi   yenta
 23:       1808        516          0          0  IR-IO-APIC  23-fasteoi   ehci_hcd:usb2
 24:          0          0          0          0  DMAR-MSI   0-edge      dmar0
 25:          0          0          0          0  DMAR-MSI   1-edge      dmar1
[...]
 35:          0          0          0     338861  IR-PCI-MSI 409600-edge      eth0
NMI:        106       1686       1730       1590   Non-maskable interrupts
LOC:   18936952   17486969   17522014   17863790   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:        106       1686       1730       1590   Performance monitoring interrupts
IWI:    1295077     484024     494488     488681   IRQ work interrupts
RTR:          0          0          0          0   APIC ICR read retries
RES:    1637089    1581900    1581198    1606231   Rescheduling interrupts
CAL:    4916487    4847329    4621559    4639746   Function call interrupts
TLB:    5665373    5685581    5628187    5680448   TLB shootdowns
TRM:          0          0          0          0   Thermal event interrupts
[...]
```