Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

22 novembre 2020

Plan des interventions :

7 cours/TP d'introduction au STM32 en *C baremetal* :

1. Électronique numérique et conception du circuit
   L3 : aspects analogiques, consommation électrique, lecture de datasheet
   Survol des divers périphériques qui seront abordés (RS232, SPI, timer, ADC)
   représentation des données (tailles/encodage), masques, architecture
   Rappels sur Atmega32U4 (Makefile, compilation, masques ...)
2. Premiers pas sur le STM32, adresses des périphériques, architecture
3. Fonctionnement de gcc et optimisations :
   préprocess–compilateur–assembleur–linker, passage C à assembleur, pointeurs
4. bibliothèques et séparation algorithme/matériel, simulateurs
   libopencm3, newlib & stubs, ressources requises par les bibliothèques
5. Bus de communication série, synchrone/asynchrone
6. arithmétique sur systèmes embarqués
   entiers, flottants, convertir un algorithme exprimé en nombres à virgules vers des entiers, timers
7. interruptions et acquisition de données analogiques, fréquence d'échantillonnage
   vecteurs d'interruption, gestion des horloges du STM32, ADC

Cas des interruptions

1. Une interruption est un méthode pour interrompre l'exécution séquentielle d'un programme lorsqu'un évènement qui ne peut pas attendre survient
2. Plusieurs sources d'interruptions disponibles sur microcontrôleur (GPIO, timer, communication) ...
3. ... selon les architectures, un gestionnaire d'interruption teste la source, ou plusieurs gestionnaires.
4. Un gestionnaire d'interruption doit être le **plus bref possible** : définir un drapeau et quitter – repousser les calculs à la fonction principale qui teste le drapeau
5. Passage de paramètres : variable globale
6. sur STM32 : NVIC [1] (Nested Vector Interrupt Controller)

1. RM0008 section 10 à `https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-pdf`

Vecteurs d'interruptions

Ensemble d'emplacements mémoire contenant, par convention, l'adresse de la fonction appelée lorsqu'un évènement se déclenche (RM0008, sec. 10, pp.198–199)

10.1.2 Interrupt and exception vectors

Table 61 and *Table 63* are the vector tables for connectivity line and other STM32F10xxx devices, respectively.

Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068

Table 61. Vector table for connectivity line devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000_008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000_0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000_0094
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC

Variables globales

- Adresse impaire : instruction *thumb* (16-bits, plus compacte que instructions ARM 32 bits)
- *Little/big-endian* : une représentation *big-endian* organise la mémoire avec l'octet le plus fort à l'adresse la plus petite (≠ little endian avec l'octet le plus faible à l'adresse la plus petite)
- Déclaration [2] de gestionnaire d'interruption (ISR – *Interrupt Service Routine*)

```
void usart1_isr(void)
{
        static uint8_t data = 'A';

        /* Check if we were called because of RXNE. */
        if (((USART_CR1(USART1) & USART_CR1_RXNEIE) != 0) &&
            ((USART_SR(USART1) & USART_SR_RXNE) != 0)) {
...
}
```

- le nom de l'ISR est imposé : `libopencm3/include/libopencm3/stm32/f1/nvic.h` contient `void usart1_isr(void);`
- `lib/stm32/f1/vector_nvic.c` déclare la table des vecteurs d'interruptions

```
/* Initialization template for the interrupt vector table. This definition is
 * used by the startup code generator (vector.c) to set the initial values for
 * the interrupt handling routines to the chip family specific _isr weak
 * symbols. */

#define IRQ_HANDLERS \
    [NVIC_WWDG_IRQ] = wwdg_isr, \
...
    [NVIC_USART1_IRQ] = usart1_isr, \
```

2. https://github.com/libopencm3/libopencm3-examples/tree/master/examples/stm32/f1/stm32-h103/usart_irq

Vecteurs d'interruptions

Après compilation [3] : afficher le contenu de la mémoire par `objdump -dSt` (préfixé de l'architecture de la cible)

```
08000580  w    F .text  00000094 reset_handler
0800057c  w    F .text  00000002 usart3_isr
0800057c  w    F .text  00000002 rtc_isr
0800057c  w    F .text  00000002 tim7_isr
0800057c  w    F .text  00000002 adc1_2_isr
0800057c  w    F .text  00000002 tim1_trg_com_isr
08000532 g     F .text  0000000c usart_set_stopbits
08000644 g       .fini_array    00000000 __exidx_end
0800057c  w    F .text  00000002 usb_hp_can_tx_isr
0800057c  w    F .text  00000002 tim6_isr
08000644 g       .data  00000000 _etext
08000268 g     F .text  00000062 gpio_set_mode
0800057c  w    F .text  00000002 usb_wakeup_isr
0800057c g     F .text  00000002 blocking_handler
0800057c  w    F .text  00000002 tim5_isr
0800057c  w    F .text  00000002 otg_fs_isr
0800057c  w    F .text  00000002 spi1_isr
080003c4 g     F .text  00000014 rcc_set_pll
0800057c  w    F .text  00000002 exti2_isr
0800057c  w    F .text  00000002 dma1_channel6_isr
08000562 g     F .text  0000000a usart_enable
0800057e g     F .text  00000002 null_handler
20000004 g     O .data  00000004 rcc_ahb_frequency
0800057c  w    F .text  00000002 can_rx1_isr
08000644 g  .fini_array 00000000 __fini_array_end
0800057c  w    F .text  00000002 dma1_channel5_isr
080002ce g     F .text  00000012 gpio_toggle
08000574 g     F .text  00000008 usart_recv
0800057c  w    F .text  00000002 dma2_channel5_isr
08000150 g     F .text  00000060 usart1_isr
```

```
Disassembly of section .text:
08000000 <vector_table>:
 8000000:       00 50 00 20 81 05 00 08 7f 05 00 08 7d 05 00 08
 8000010:       7d 05 00 08 7d 05 00 08 7d 05 00 08 00 00 00 00
        ...
 800002c:       7f 05 00 08 7f 05 00 08 00 00 00 00 7f 05 00 08
 800003c:       7f 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800004c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800005c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800006c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800007c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800008c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800009c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 80000ac:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 80000bc:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 80000cc:       7d 05 00 08 7d 05 00 08 51 01 00 08 7d 05 00 08
 80000dc:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 80000ec:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 80000fc:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800010c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800011c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800012c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800013c:       7d 05 00 08 7d 05 00 08 7d 05 00 08 7d 05 00 08
 800014c:       7d 05 00 08

08000150 <usart1_isr>:
        gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_50_MHZ,
                      GPIO_CNF_OUTPUT_PUSHPULL, GPIO12);
}

void usart1_isr(void)
{
 8000150:       b538            push    {r3, r4, r5, lr}
        static uint8_t data = 'A';
...
```

3. https://github.com/libopencm3/libopencm3-examples/tree/master/examples/stm32/f1/stm32-h103/usart_irq

Vecteur d'interruption=pointeur de fonction

Le programme pour AVR

```
int f1(void) {return(1);}
int f2(void) {return(2);}
int f3(int i,int j) {return(i+j);}

int main()
{int (*ma_fonction)(void);
 int (*fonction)(int,int);
 ma_fonction=&f1; ma_fonction();
 ma_fonction=&f2; ma_fonction();
 fonction=&f3;    fonction(1,2);
}
```

se compile par

avr-gcc -mmcu=atmega32u4 -g prg.c :

```
000000ac  w      .text  00000000 __init
000000c0  w      .text  00000000 __vector_13
000000e8 g     F .text  0000002e f3
...
000000d6 g     F .text  00000012 f2
00000000 g       .text  00000000 __vectors
...
000000c4 g     F .text  00000012 f1
...
Disassembly of section .text:
00000000 <__vectors>:
0:   0c 94 56 00  jmp 0xac ; 0xac <__ctors_end>
4:   0c 94 60 00  jmp 0xc0 ; 0xc0 <__bad_interrupt>
8:   0c 94 60 00  jmp 0xc0 ; 0xc0 <__bad_interrupt>
c:   0c 94 60 00  jmp 0xc0 ; 0xc0 <__bad_interrupt>
10:  0c 94 60 00  jmp 0xc0 ; 0xc0 <__bad_interrupt>
...
000000ac <__ctors_end>:
  ac:   11 24           eor     r1, r1
...
  b4:   de bf           out     0x3e, r29       ; 62
  b6:   cd bf           out     0x3d, r28       ; 61
  b8:   0e 94 8b 00     call    0x116   ; 0x116 <main>
  bc:   0c 94 b6 00     jmp     0x16c   ; 0x16c <_exit>
000000c0 <__bad_interrupt>:
```

```
int f1(void) {return(1);}
  c4:   cf 93           push    r28
...
  d2:   cf 91           pop     r28
  d4:   08 95           ret
000000d6 <f2>:
int f2(void) {return(2);}
  d6:   cf 93           push    r28
...
  e4:   cf 91           pop     r28
  e6:   08 95           ret
000000e8 <f3>:
int f3(int i,int j) {return(i+j);}
  e8:   cf 93           push    r28
...
 112:   cf 91           pop     r28
 114:   08 95           ret

00000116 <main>:
int main()
{int (*ma_fonction)(void);
 116:   cf 93           push    r28
 118:   df 93           push    r29
...
 int (*fonction)(int,int);
 ma_fonction=&f1; ma_fonction();
 122:   82 e6           ldi     r24, 0x62       ; 98
 124:   90 e0           ldi     r25, 0x00       ; 0
...
 130:   09 95           icall
 ma_fonction=&f2; ma_fonction();
 132:   8b e6           ldi     r24, 0x6B       ; 107
 134:   90 e0           ldi     r25, 0x00       ; 0
...
 140:   09 95           icall
 fonction=&f3;    fonction(1,2);
 142:   84 e7           ldi     r24, 0x74       ; 116
 144:   90 e0           ldi     r25, 0x00       ; 0
...
 158:   09 95           icall
```

Variables globales

```
volatile int global_index , tim0 , heure , minute , seconde ;

void IRQ_Handler (void)          // unique gestionnaire pour ADuC7026
{ if (IRQSIG & UART_BIT)
    {                            // UART Interrupt
      global_tab[global_index] = my_getchar ();
      if (global_index < (NB_CHARS - 1))
        global_index++;
    }
  if (IRQSIG & GP_TIMER_BIT)
    {                            // Timer1 Interrupt
      GLOBAL_TIMER1++;
      T1CLRI = 1;                // Clear Timer 1 interrupt
    }
  if (IRQSIG & RTOS_TIMER_BIT)
    { tim0++;                    // Timer0 Interrupt
      seconde++;
      if (seconde > 600)
        {seconde = 0;minute++;} // seconde en 1/10 seconde
      if (minute > 60) {minute = 0;heure++;}
      if (heure > 24)  {heure = 0;}
      T0CLRI = 0;
    }
}
```

Séquence d'initialisation d'un microcontrôleur

1. Au démarrage, un microcontrôleur initialise *par convention* son PC au vecteur de RESET
2. La fonction appelée doit initialiser le processeur, en particulier le SP
3. Cas particulier du Cortex M : SP est défini à la compilation [4] et placé à l'adresse 0x00000000 ! On a bien (`objdump -dSt`)

   ```
   08000000 <vector_table>:
   8000000:  00 68 00 20 a1 08 00 08 9f 08 00 08 9d 08 00 08
   ```

   donc SP=0x20006800

   (cf .ld : `ram (rwx) : ORIGIN = 0x20000000, LENGTH = 26K`

   et $26 \times 1024 = 0x6800$)
4. l'adresse 0x04 contient le vecteur de RESET

   ```
   080008a0 <reset_handler>:
   80008a0:        2200            movs    r2, #0
   ```

   le reset handler est a 800008A1 pour indiquer [5] que les instructions sont sur 16 bits (mode thumb)

   "The value at that location should be odd denoting the thumb destination. All exception and interrupt vectors following reset vector should be odd."

4. `infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/BABIFJFG.html` ou `fastbitlab.com/arm-cortex-m-processor-reset-sequence/`
5. `searchingforbit.blogspot.com/2011/11/oddeven-value-for-cortex-m3-reset.html`

Déclenchement ADC sur timer (1/2 : timer)

```
static void timer_setup(void)
{ uint32_t timer;
  volatile uint32_t *rcc_apbenr;
  uint32_t rcc_apb;

  timer = TIM2;
  rcc_apbenr = &RCC_APB1ENR;
  rcc_apb = RCC_APB1ENR_TIM2EN;

  rcc_peripheral_enable_clock(rcc_apbenr, rcc_apb);
  timer_reset(timer);
  timer_set_mode(timer, TIM_CR1_CKD_CK_INT, TIM_CR1_CMS_EDGE, TIM_CR1_DIR_UP);
  timer_set_period(timer, 0xF*5);
  timer_set_prescaler(timer, 0x8);
  timer_set_clock_division(timer, 0x0);
  timer_set_master_mode(timer, TIM_CR2_MMS_UPDATE); // Generate TRGO on every update
  timer_enable_counter(timer);
}
```

Configuration du timer avec son *prescaler* et `période` + signal TRG0 pour déclencher ...

Déclenchement ADC sur timer (2/2 : ADC)

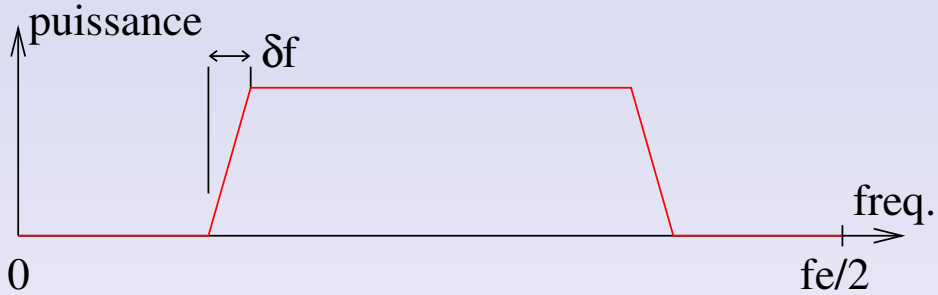
... la conversion analogique-numérique dont le résultat déclenche l'interruption `adc1_2_isr`

```
static void adc_setup(void)
{ int i;
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_ADC1EN);
  adc_off(ADC1);
  adc_enable_scan_mode(ADC1);
  adc_set_single_conversion_mode(ADC1);
  adc_enable_external_trigger_injected(ADC1,ADC_CR2_JEXTSEL_TIM2_TRGO); // start /w TIM2 TRGO
  adc_enable_eoc_interrupt_injected(ADC1);                              // Generate ADC1_2_IRQ
  adc_set_right_aligned(ADC1);
  adc_set_sample_time_on_all_channels(ADC1, ADC_SMPR_SMP_28DOT5CYC);
  adc_power_on(ADC1);

  for (i = 0; i < 800000; i++) __asm__("nop");
  adc_reset_calibration(ADC1); while ((ADC_CR2(ADC1) & ADC_CR2_RSTCAL) != 0);
  adc_calibration(ADC1);       while ((ADC_CR2(ADC1) & ADC_CR2_CAL) != 0);
}

void adc1_2_isr(void)
{ ADC_SR(ADC1) &= ~ADC_SR_JEOC;
  if (jmf_index<NB)
     {mesure[jmf_index]=adc_read_injected(ADC1,1);jmf_index++;}
}


static void irq_setup(void)
{ nvic_set_priority(NVIC_ADC1_2_IRQ, 0);
  nvic_enable_irq(NVIC_ADC1_2_IRQ);
}
```

La boucle principale teste `jmf_index` pour savoir si `mesure` est plein

Pourquoi faut-il échantillonner périodiquement ?

- Tout traitement numérique du signal échantillonné en temps discret est basé sur l'hypothèse d'un échantillonnage périodique



- exemple d'un filtre :

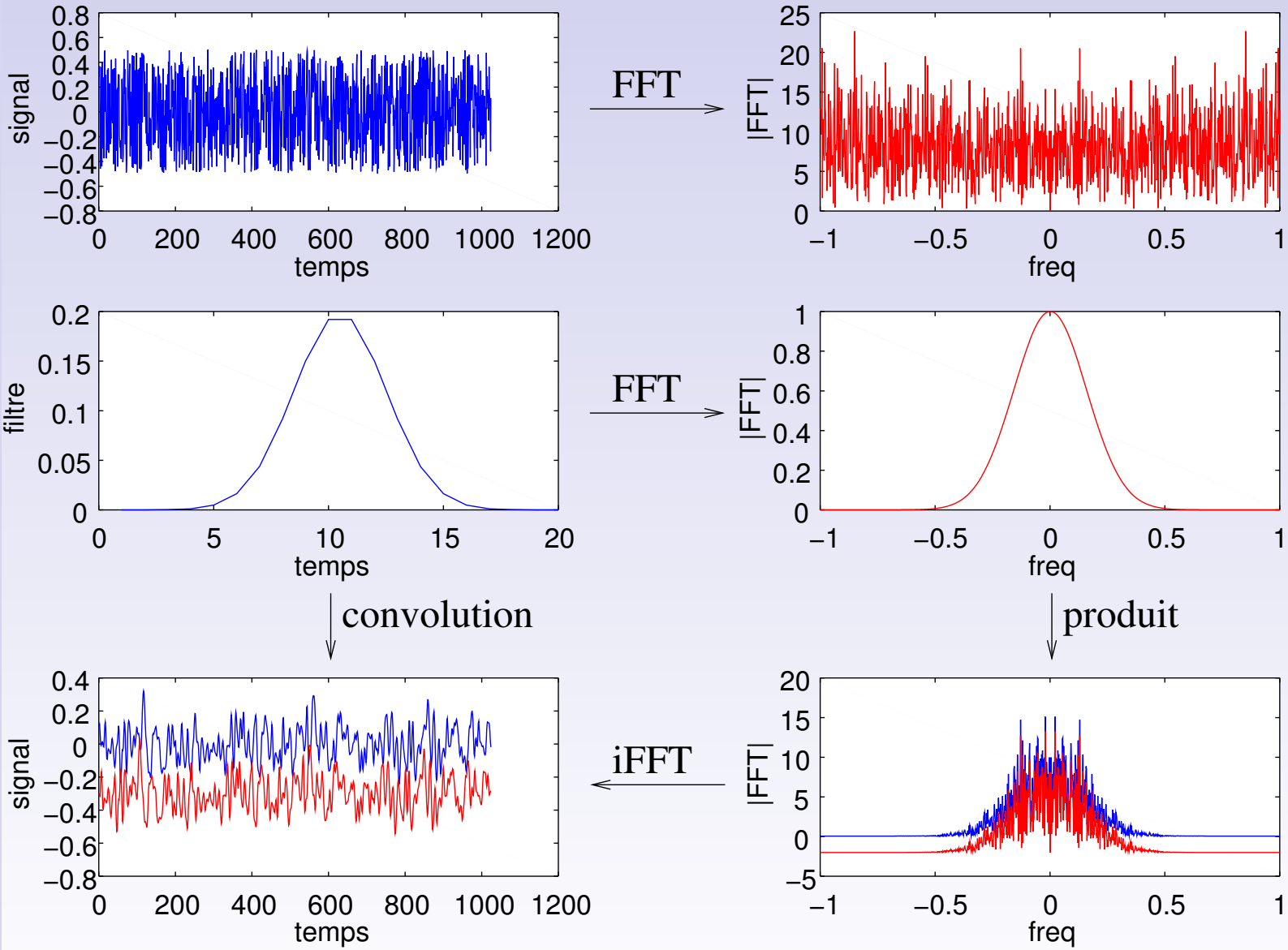
un FIR est défini par une convolution (combinaison linéaire des entrées) :

$$y_n = \sum_{k=1..m} b_k \cdot x_{n-k}$$

$\Rightarrow$ il *faut* attendre m données pour enclencher le filtre

- attendre = déphasage
- déphasage = point critique d'un asservissement qui peut rendre le système instable (oscillation)

FIR



Approche temporelle v.s fréquentielle [6]

6. http://www.trondeau.com/examples/2010/9/12/basic-filtering.html

FIR

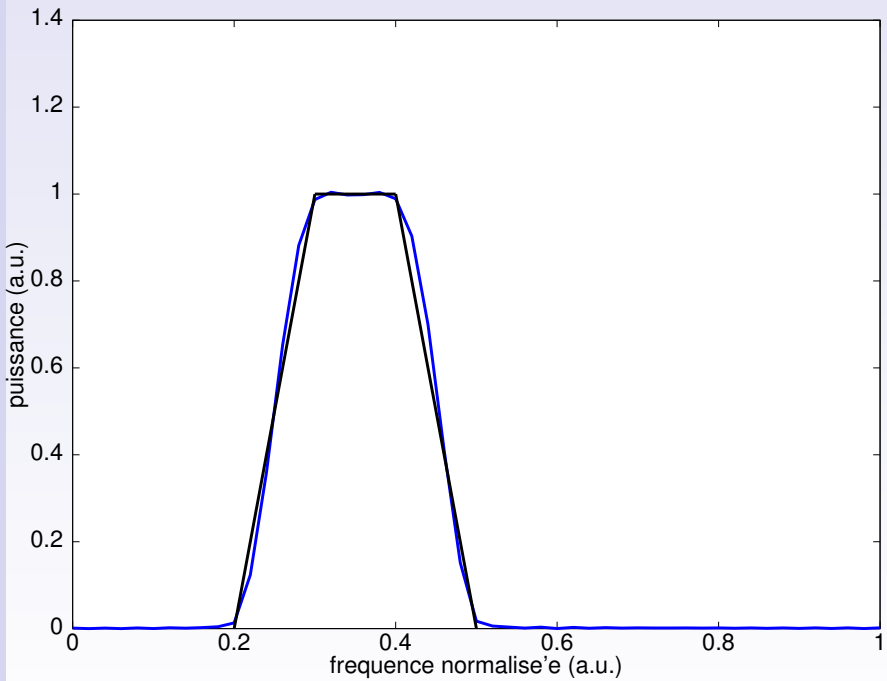
```
x=rand(1024,1);x=x-mean(x);     % signal : valeur moyenne=0
subplot(321); plot(x);xlabel('temps');ylabel('signal')
t=linspace(-10,10,20);
y=exp(-t.^2/9);y=y/sum(y);      % filtre
subplot(323);plot(y);xlabel('temps');ylabel('filtre')
f=linspace(-1,1,1024);

yc=conv(x,y);yc=yc(10:end-10); % signal filtre' (conv)
subplot(325);plot(yc);xlabel('temps');ylabel('signal')

subplot(322);
plot(f,abs(fftshift(fft(x))),'r')  % spectre du signal
subplot(324);
plot(f,abs(fftshift(fft(y,1024))),'r') % spectre du filtre
subplot(326);
plot(f,abs(fftshift(fft(yc))))     % spectre signal filtre'
hold on
yyc=fft(x,1024).*fft(y,1024)';     % produit des spectres
plot(f,abs(fftshift(yyc))-2,'r')

subplot(325);hold on;
plot(real(ifft(yyc))-0.3,'r')      % retour dans le temps
```

Exercice : refaire avec une fenêtre rectangulaire

FIR v.s IIR

1. FIR : $y_n = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées, pas de risque de divregence, mais long à converger
2. IIR : $\sum a_k y_{n-k} = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées *et des sorties passées*, risque de divregence, mais convergence rapide
3. x_k est mesuré par le matériel : résolution imposée
4. combien de bits sur les cofficients b_k pour garantir le résultat de la sortie ?

Exemple de coefficients représentés en nombres à virgule flottante



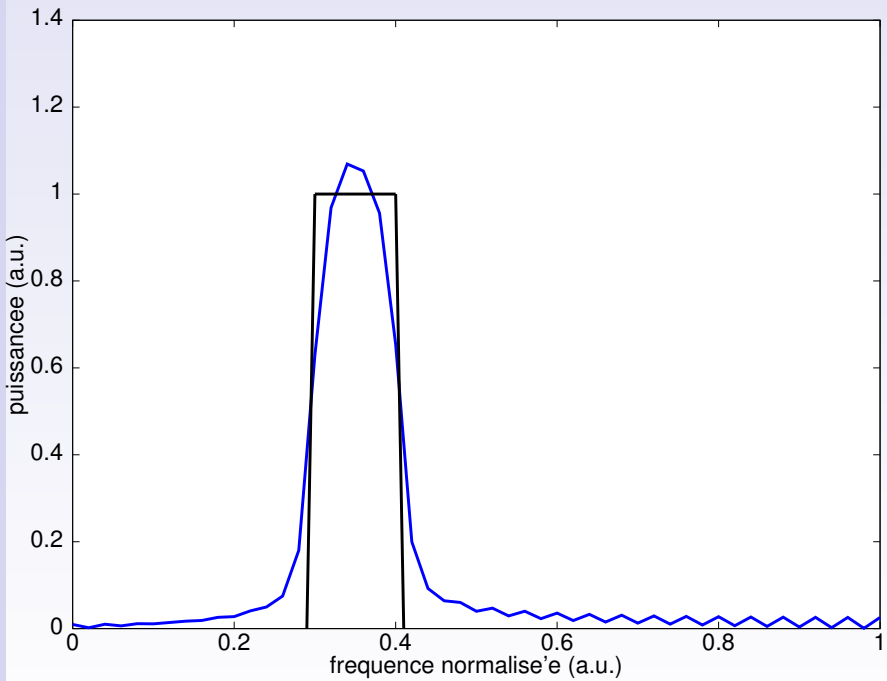
1. Fonction `firls` de GNU/Octave (ou Matlab) pour fournir les fréquences normalisées, les coefficients de pondération, et le nombre de coefficients `a_k`
2. $y_n = \sum_1 \frac{a_k}{a_0} y_{n-k} - \sum_0 \frac{b_k}{a_0} x_{n-k}$ ou en transformée en Z :

$$H(z) = \frac{\sum_0 b_{k+1}/a_0 \cdot z^{-k}}{1 + \sum_1 a_k/a_0 \cdot z^{-k}}$$

FIR v.s IIR

1. FIR : $y_n = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées, pas de risque de divregence, mais long à converger
2. IIR : $\sum a_k y_{n-k} = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées *et des sorties passées*, risque de divregence, mais convergence rapide
3. x_k est mesuré par le matériel : résolution imposée
4. combien de bits sur les cofficients b_k pour garantir le résultat de la sortie ?

50 coefficients représentés en nombres à virgule fixes



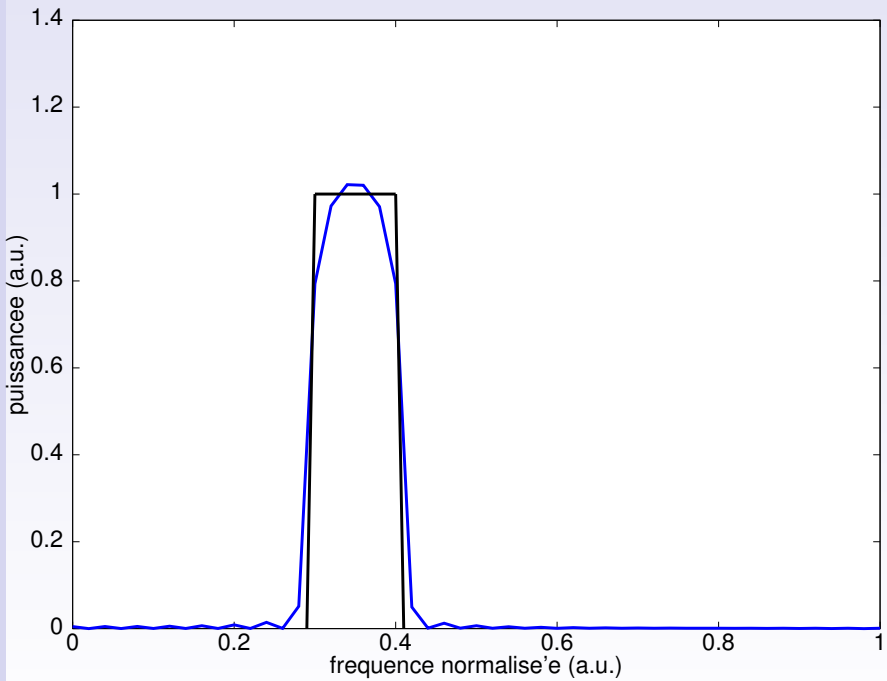
1. Fonction `firls` de GNU/Octave (ou Matlab) pour fournir les fréquences normalisées, les coefficients de pondération, et le nombre de coefficients `a_k`
2. $y_n = \sum_1 \frac{a_k}{a_0} y_{n-k} - \sum_0 \frac{b_k}{a_0} x_{n-k}$ ou en transformée en Z :

$$H(z) = \frac{\sum_0 b_{k+1}/a_0 \cdot z^{-k}}{1 + \sum_1 a_k/a_0 \cdot z^{-k}}$$

FIR v.s IIR

1. FIR : $y_n = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées, pas de risque de divregence, mais long à converger
2. IIR : $\sum a_k y_{n-k} = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées *et des sorties passées*, risque de divregence, mais convergence rapide
3. x_k est mesuré par le matériel : résolution imposée
4. combien de bits sur les cofficients b_k pour garantir le résultat de la sortie ?

150 coefficients représentés en nombres à virgule fixe

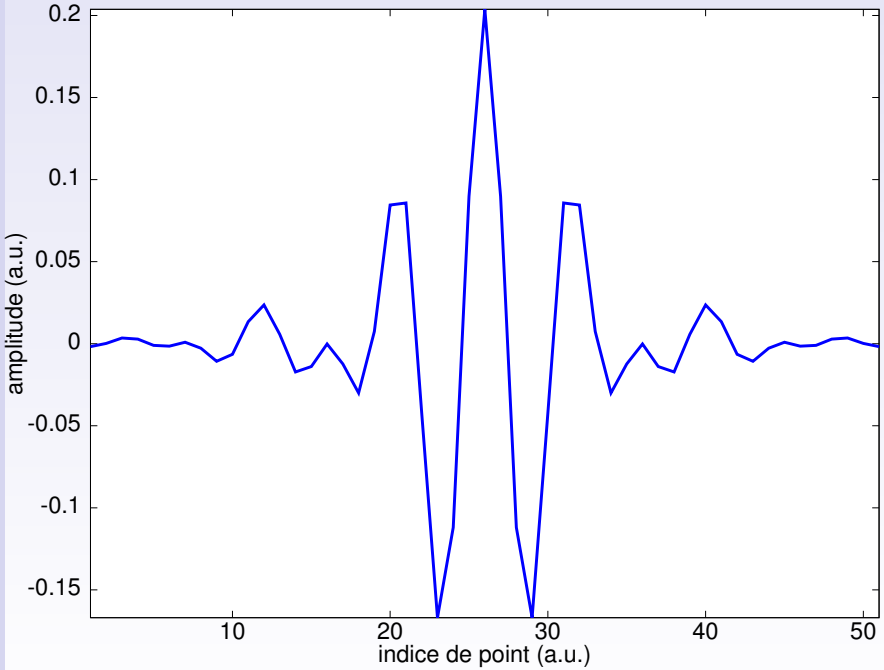


1. Fonction `firls` de GNU/Octave (ou Matlab) pour fournir les fréquences normalisées, les coefficients de pondération, et le nombre de coefficients `a_k`
2. $y_n = \sum_1 \frac{a_k}{a_0} y_{n-k} - \sum_0 \frac{b_k}{a_0} x_{n-k}$ ou en transformée en Z :

$$H(z) = \frac{\sum_0 b_{k+1}/a_0 \cdot z^{-k}}{1 + \sum_1 a_k/a_0 \cdot z^{-k}}$$

FIR v.s IIR

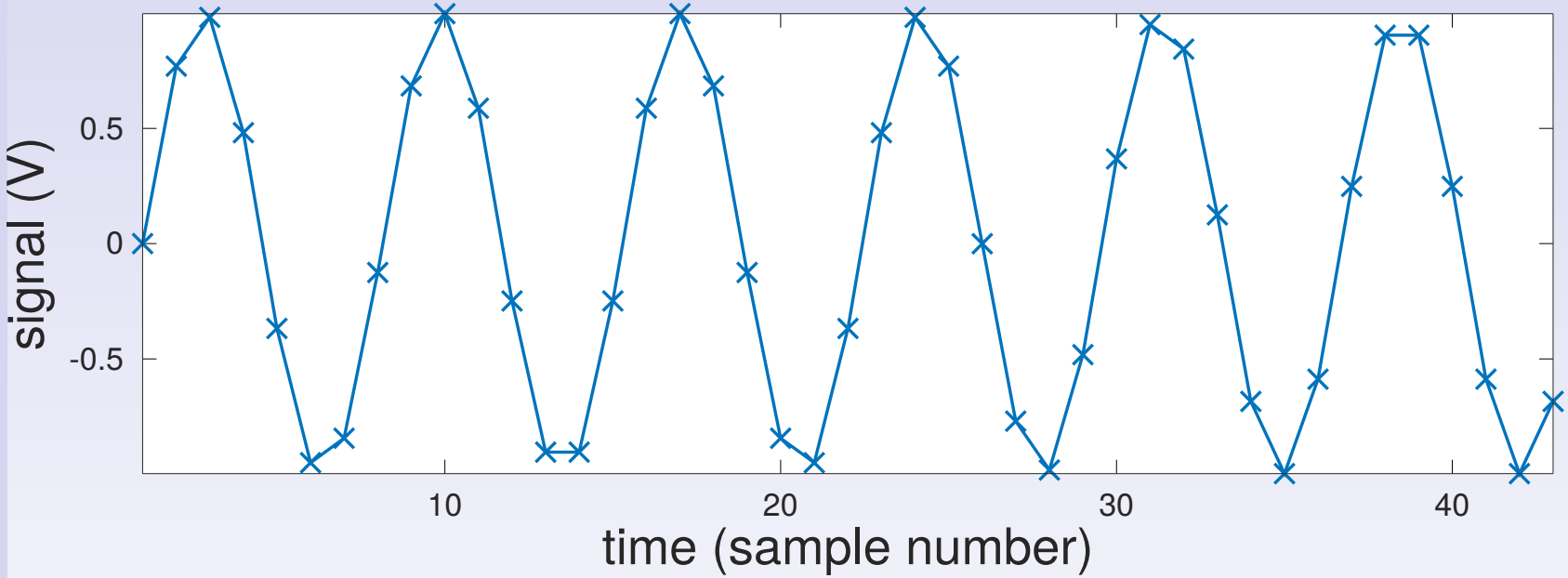
1. FIR : $y_n = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées, pas de risque de divregence, mais long à converger
2. IIR : $\sum a_k y_{n-k} = \sum b_k \cdot x_{n-k}$: somme pondérée des entrées *et des sorties passées*, risque de divregence, mais convergence rapide
3. x_k est mesuré par le matériel : résolution imposée
4. combien de bits sur les cofficients b_k pour garantir le résultat de la sortie ?



1. Fonction `firls` de GNU/Octave (ou Matlab) pour fournir les fréquences normalisées, les coefficients de pondération, et le nombre de coefficients `a_k`
2. Les coefficients du FIR sont la réponse impulsionnelle du filtre

Fréquence d'échantillonnage

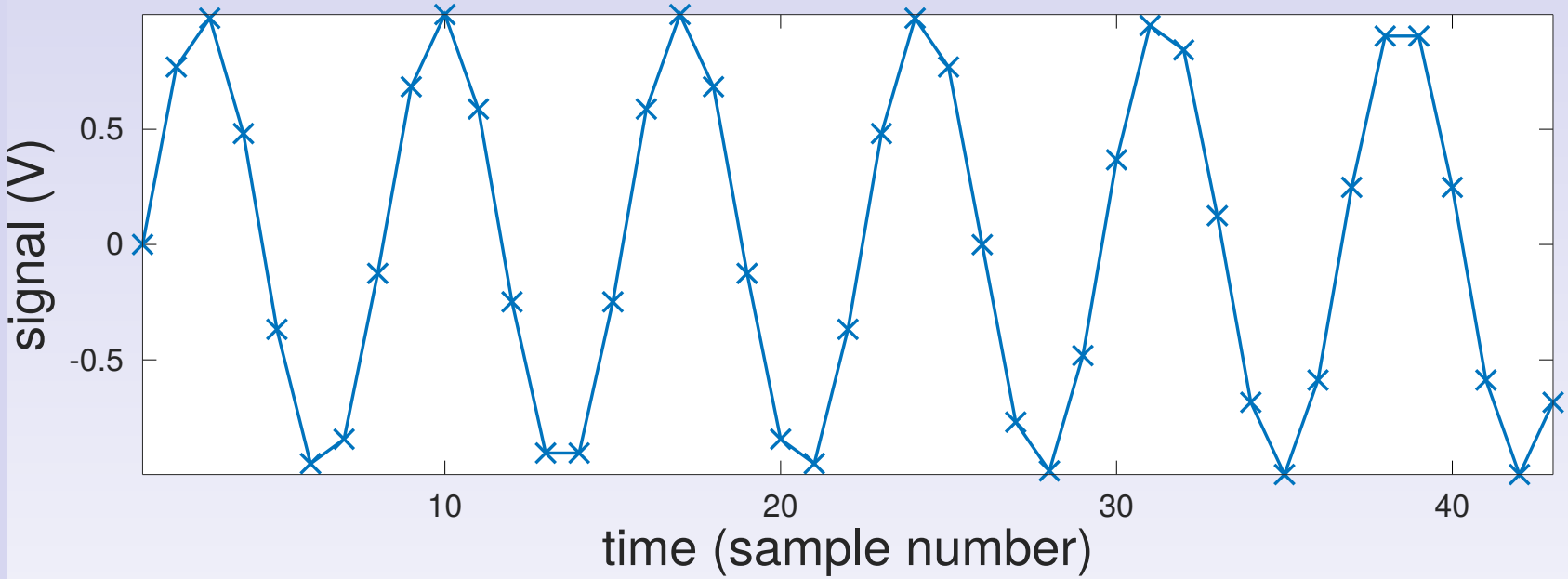
Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Si le signal émis est à 125 Hz, quelle est la fréquence d'échantillonnage ?

Fréquence d'échantillonnage

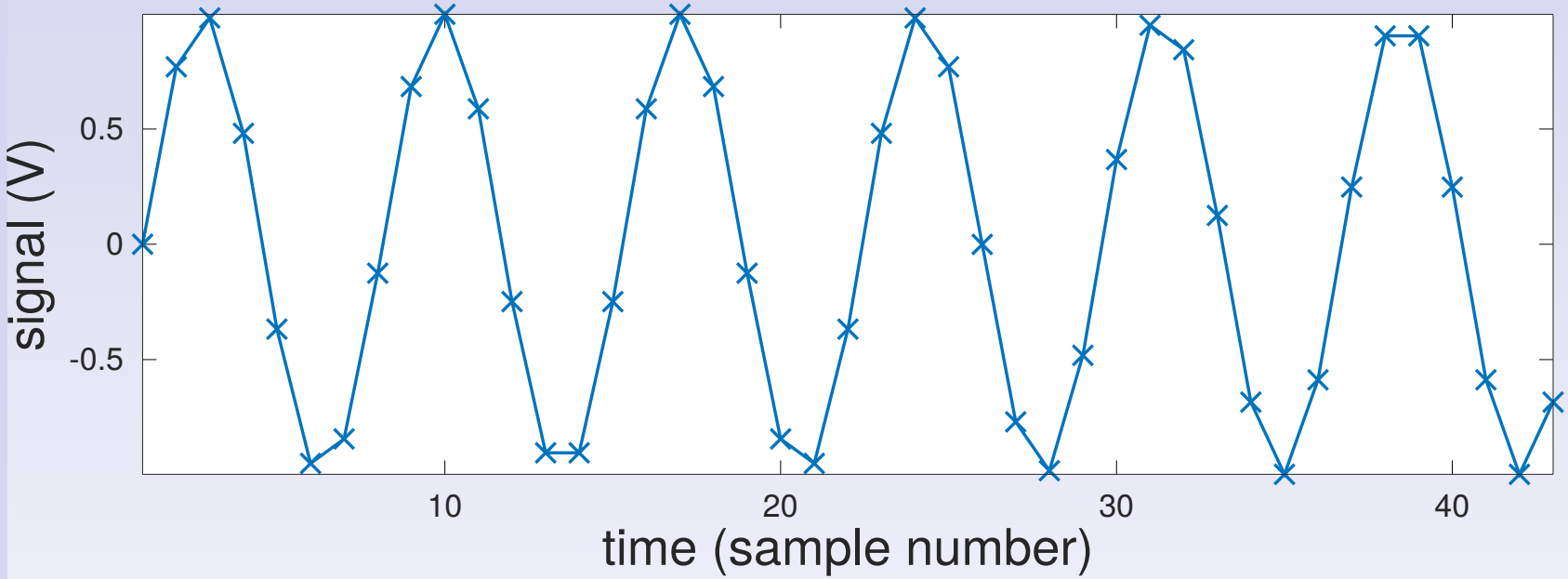
Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Si le signal émis est à 125 Hz, quelle est la fréquence d'échantillonnage ? $125 \times 8 = 1000$ Hz
2. Quelle est l'incertitude sur cette estimation, et comment l'améliorer ?

Fréquence d'échantillonnage

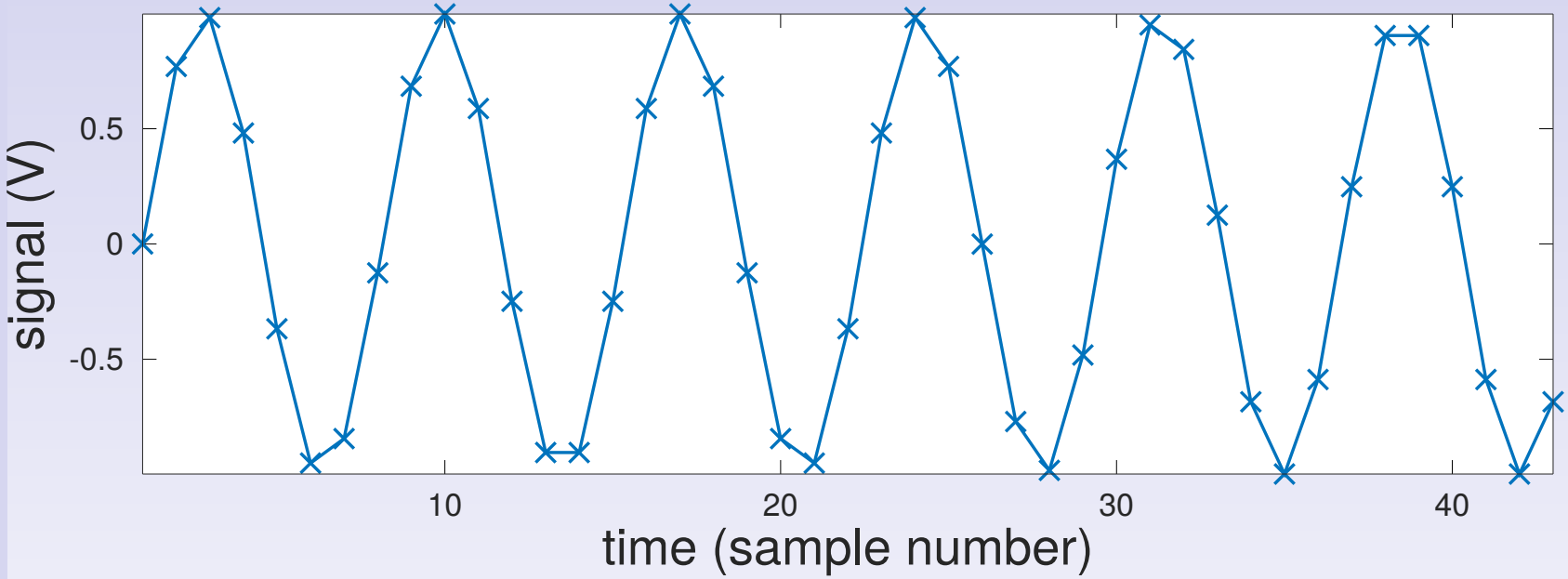
Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Si le signal émis est à 125 Hz, quelle est la fréquence d'échantillonnage ? $125 \times 8 = 1000$ Hz
2. Quelle est l'incertitude sur cette estimation, et comment l'améliorer ? $8 \pm 1 \Rightarrow \pm 125$ Hz mais sur 5 périodes : ± 25 Hz
3. Quelle est la fréquence normalisée du signal ?

Fréquence d'échantillonnage

Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Si le signal émis est à 125 Hz, quelle est la fréquence d'échantillonnage ? $125 \times 8 = 1000$ Hz
2. Quelle est l'incertitude sur cette estimation, et comment l'améliorer ? $8 \pm 1 \Rightarrow \pm 125$ Hz mais sur 5 périodes : ± 25 Hz
3. Quelle est la fréquence normalisée du signal ? $1/7.2 = 0{,}139$ [7]

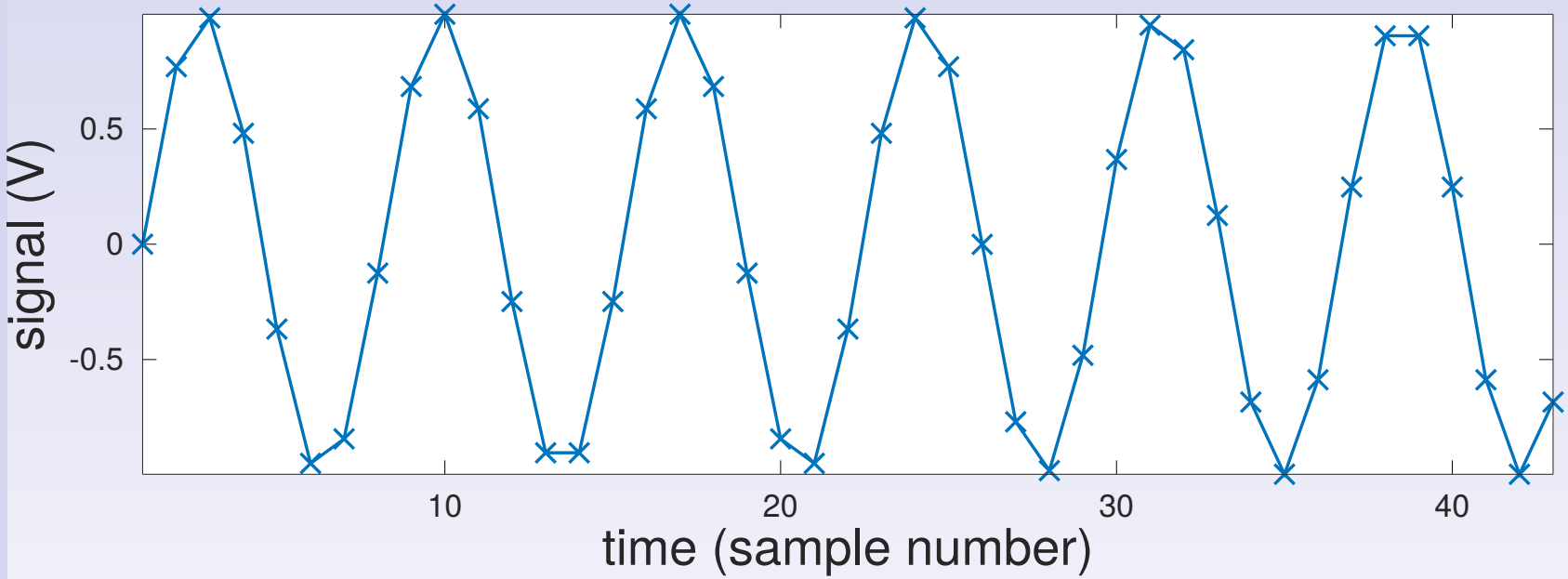
`plot(sin(2*pi*4*([0:0.035:1.5])),'x-')`

$0,035 \times 4 = 0,14$ `tandis que` $1/7,2 = 0,139$ `et` $1/7 = 0,14286$

7. 36 ± 1 points pour 5 periodes $= 7{,}2 \pm 0,2$ points/periode

Fréquence d'échantillonnage

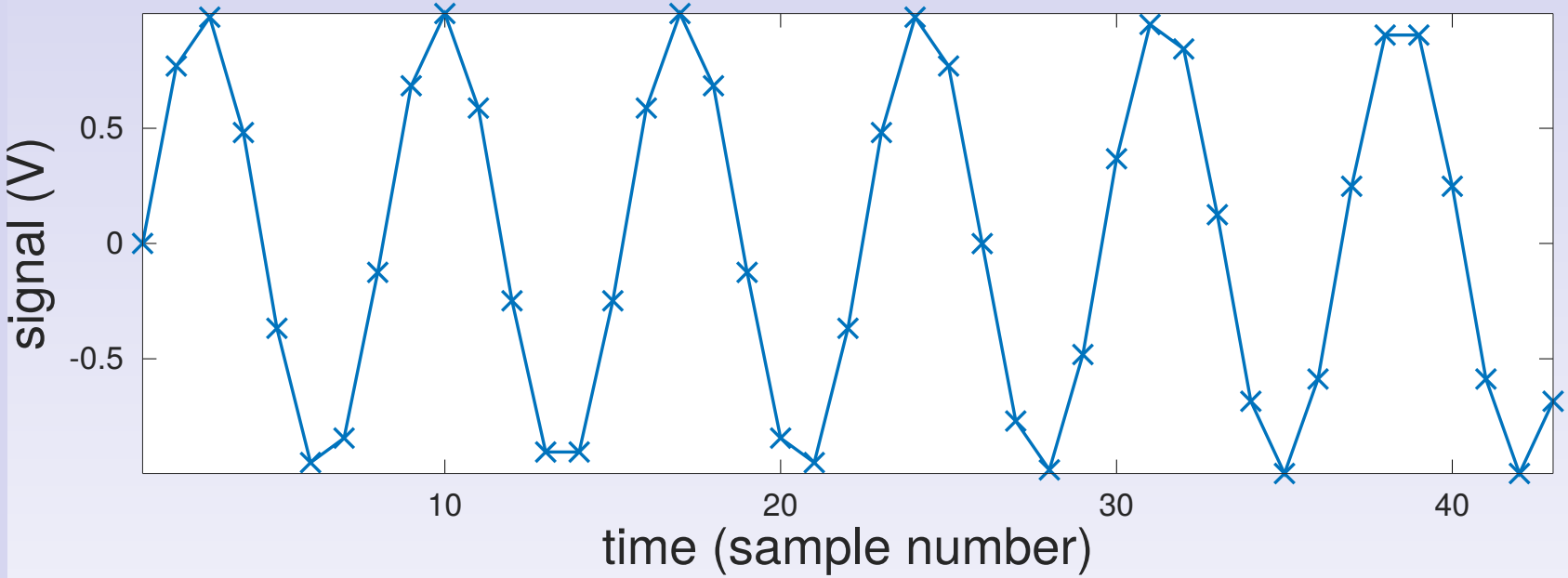
Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Combien de périodes faut-il pour obtenir la fréquence d'échantillonnage normalisée à 2 décimales près, connaissant la fréquence du signal f ?

Fréquence d'échantillonnage

Soit un signal acquis par une boucle qui lit le convertisseur analogique-numérique le plus rapidement possible :



1. Combien de périodes faut-il pour obtenir la fréquence d'échantillonnage normalisée à 2 décimales près, connaissant la fréquence du signal f ?
   $f_s = N/N_p \times f \Rightarrow df_s/f_s = dN/N + dN_p/N_p$ si f est connu
   $N \gg N_p \Rightarrow N_p \simeq 100$ puisque $dN_p = \pm 1$

Conclusion

1. Importance de bien comprendre le mécanisme des interruptions pour utiliser efficacement le matériel (démarrage, évènements asynchrones)
2. Un service d'interruption (ISR) est toujours le plus court possible : marquer un drapeau (*flag*) et le traiter plus tard
3. Exceptionnellement, variables globales pour échanger des informations entre ISR et programme principal ($\Rightarrow$ `volatile` pour **interdire les optimisations** abusives du compilateur)
4. Le timer déclenche un évènement périodique ...
5. ... telle que la conversion analogique-numériqe.
6. Nécessité de connaître la fréquence d'échantillonnage f_e pour tout traitement numérique du signal (axe des fréquences normalisé par $f_e/2$).

Exercice : proposer une initialisation du Timer1 et son ISR qui modifie l'état de la LED chaque seconde

Solution au problème précédent

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

#define decimales 1000000LL // LL=long long

long jm_sqrt(long long A)
{long long x,xp; // ,n=0;
 {if (A==0) return(0);
  if (A==1) return(1); // Newton ne marche pas en 1
  xp=A>>1;             // initialisation : A/2
  do {
    x=xp;
    if (x!=0) xp=(x+A/x)>>1;   // x(n+1)=1/2*(x(n)+A/x(n))
    else break;
  } while ((x-xp)>1);          //   n++;
 }
 return(xp);
}

int main()
{long long res=jm_sqrt((2*decimales*decimales));
 printf("%d.%d\n",res/decimales,res-(res/decimales)*decimales);
}
```

s'exécute

```
./racine
1.414213
```

```
./racine    # 2 -> 142
11.916375
```

qui semble correct :

```
octave> format long
octave> sqrt(2)
ans =  1.414213562373095
```

```
octave> format long
octave> sqrt(142)
ans =  11.91637528781298
```

Démonstration du thérorème de convolution

$$\left.\begin{array}{l} conv(x,y)(\tau) = \int x(t)\cdot y(\tau - t)dt \\ TF(x)(f) = \int x(t)\exp(j2\pi ft)dt \end{array}\right\} \Rightarrow TF(conv(x,y)) = \iint x(t)y(\tau - t)dt\exp(j2\pi f\tau)d\tau$$

on pose $s = \tau - t \Rightarrow ds = d\tau$ et

$$\begin{array}{l} \Leftrightarrow TF(conv(x,y)) = \iint x(t)y(s)dt\exp(j2\pi f(t+s))ds \\ \Leftrightarrow TF(conv(x,y)) = \int x(t)\exp(j2\pi ft)dt \times \int y(s)\exp(j2\pi fs)ds \\ \Leftrightarrow TF(conv(x,y)) = TF(x)\cdot TF(y) \end{array}$$