Systèmes embarqués

J.-M Friedt, SENSeOR

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

16 octobre 2016

Les systèmes embarqués

- L'embarqué : un domaine en plein essor, probablement l'avenir de l'informatique ("mobile", "autonome", "portable"...).
- Caractérsitiques ? Compacité et consommation.

De la machine à laver au routeur ethernet en passant par le téléphone portable et l'appareil photo numérique.

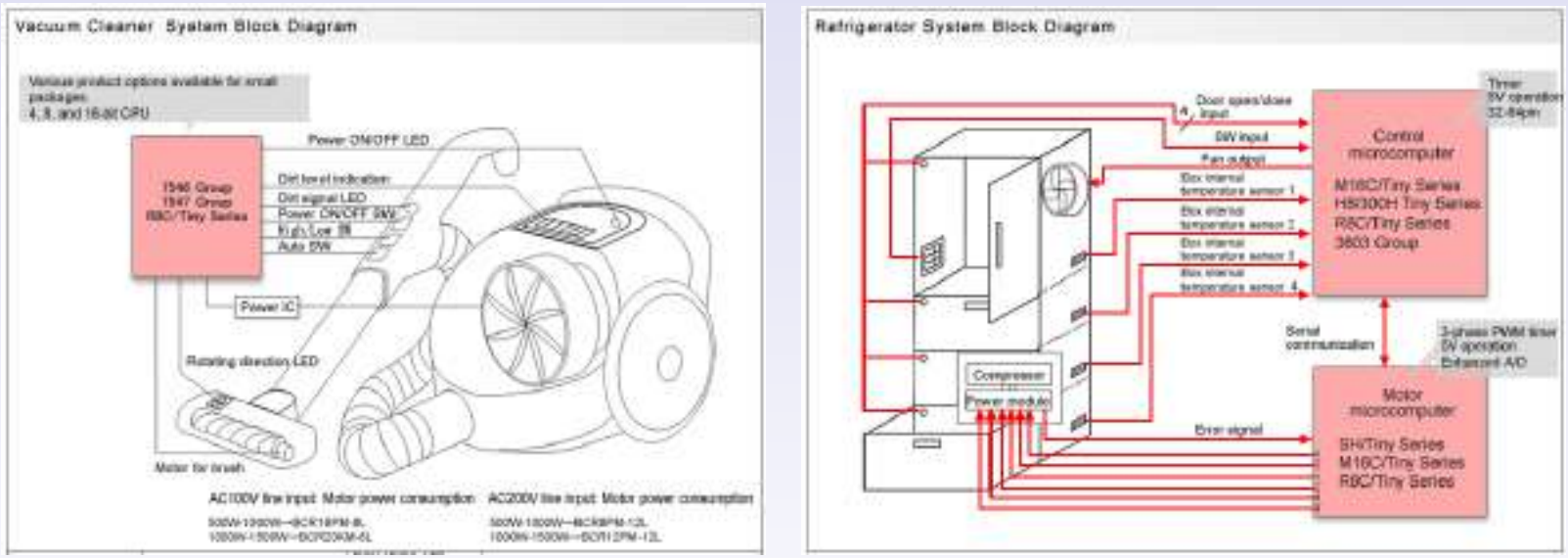


FIGURE – Quelques exemples de systèmes embarqués (Renesas), et choix des composants numériques associés.

Les systèmes embarqués

- L'embarqué : un domaine en plein essor, probablement l'avenir de l'informatique ("mobile", "autonome", "portable"...).
- Caractérsitiques ? Compacité et consommation.

De la machine à laver au routeur ethernet en passant par le téléphone portable et l'appareil photo numérique.

FIGURE – Quelques exemples de systèmes embarqués (Renesas), et choix des composants numériques associés.

Les systèmes embarqués

Avoir des outils (matériels et logiciels) adaptés à son application.
Un système embarqué est un *ensemble*

- des éléments de calcul numériques (processeurs 8 à 32 bits, de quelques octets de RAM pour l'ATtiny à des MB pour des systèmes linux embarqués)
- du logiciel gérant au mieux les systèmes numériques et périphériques (travail sur interruption avec mode de veille profonde, watchdog)
- une gestion efficace de l'énergie (minimiser le nombre de tensions, favoriser les convertisseurs DC-DC au lieu des régulateurs linéaires, couper les périphériques inutiles)

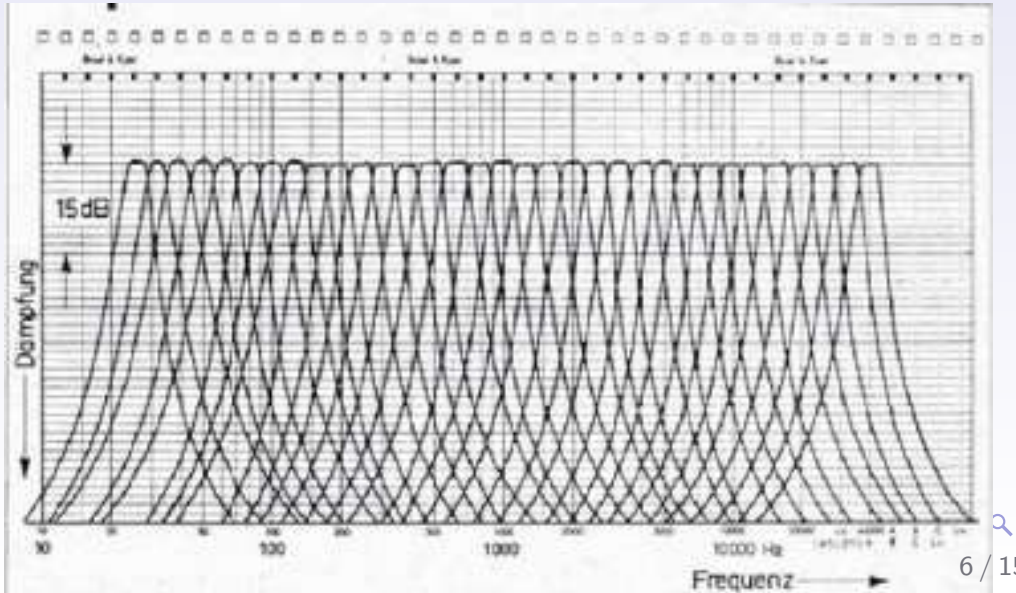
Objectif de cette présentation

- Justifier l'utilisation des systèmes numériques dans les applications embarquées
- proposer des solutions à divers problèmes de complexité variable, de l'automate à la console de jeu
- proposer des *méthodes* d'analyse du problème et des *outils génériques libres* pour le résoudre
- des travaux pratiques pour mettre en œuvre les connaissances acquises
- Le logiciel libre dans l'embarqué : une solution fonctionnelle (`http://2008.rmll.info/-Embarque-.html` à `https://2015.rmll.info/systemes-embarques?lang=fr`) particulièrement adaptée lors de l'apprentissage (disponibilité des codes sources).
- Revues périodiques des systèmes embarqués sous GNU/Linux (`http://www.ed-diamond.com/`), `http://linuxdevices.com/`

Pourquoi le numérique ?

Exemple de la caractérisation de la fonction de transfert d'un dispositif radiofréquence

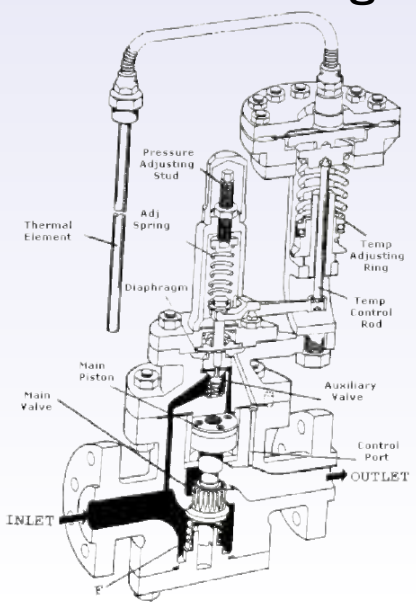
- une source de signaux, une mise en forme des signaux, une analyse des signaux transmis, et une communication avec l'utilisateur
- tout ceci peut se faire de façon analogique : source de bruit et banc de filtres
- pourquoi l'obsession pour le numérique ?
  - **stabilité** (les algorithmes n'évoluent pas dans le temps)
  - **flexibilité** (adaptation de l'algorithme en fonction de conditions de mesure, adaptation des paramètres de mesure)
  - **reconfigurabilité** (un seul circuit, plusieurs algorithmes)



Pourquoi le numérique ?

Exemple de la caractérisation de la fonction de transfert d'un dispositif radiofréquence

- une source de signaux, une mise en forme des signaux, une analyse des signaux transmis, et une communication avec l'utilisateur
- tout ceci peut se faire de façon analogique : source de bruit et banc de filtres
- pourquoi l'obsession pour le numérique ?
  - **stabilité** (les algorithmes n'évoluent pas dans le temps)
  - **flexibilité** (adaptation de l'algorithme en fonction de conditions de mesure, adaptation des paramètres de mesure)
  - **reconfigurabilité** (un seul circuit, plusieurs algorithmes)



←

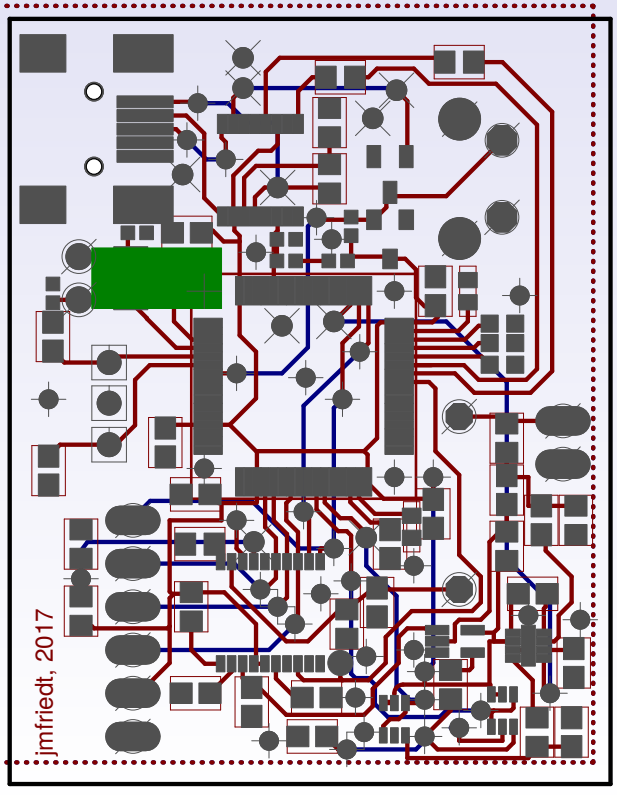
contrôleur en pression et température (`hawsepipe.net`)

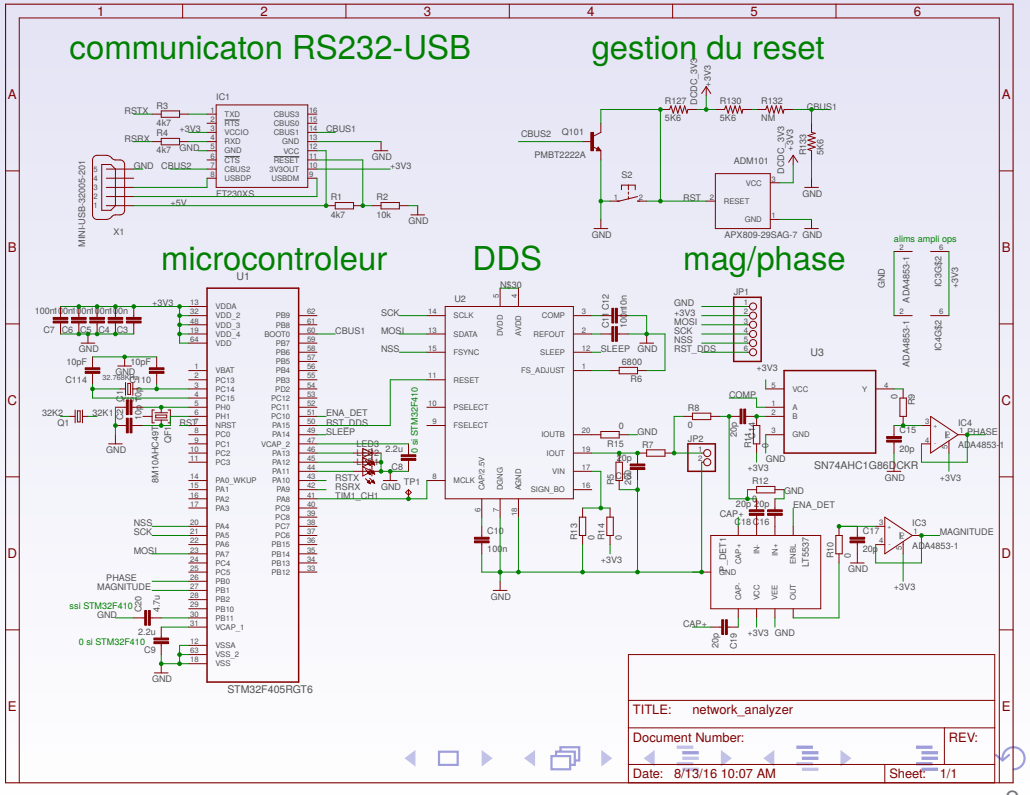
V2 →

Pourquoi le numérique ?

Exemple de la caractérisation de la fonction de transfert d'un dispositif radiofréquence

- une source de signaux, une mise en forme des signaux, une analyse des signaux transmis, et une communication avec l'utilisateur
- tout ceci peut se faire de façon analogique : source de bruit et banc de filtres
- pourquoi l'obsession pour le numérique ?





Choix des outils

Diversité des outils et plate-formes disponibles :

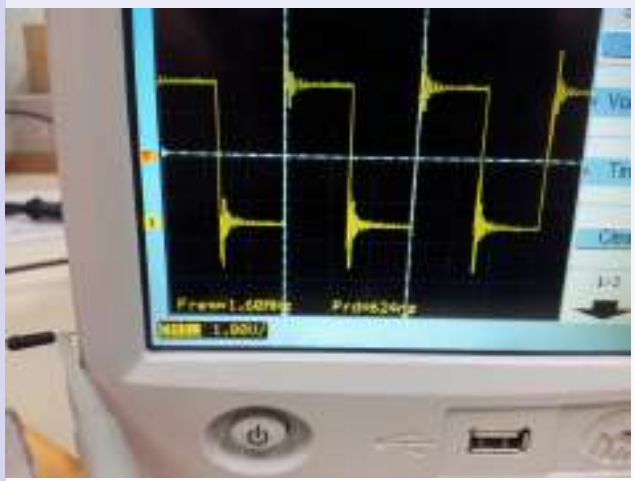
- focalisation sur les systèmes numériques, répondant au mieux aux besoins de maintenance, souplesse, diversité des applications... mais le numérique reste avant tout de l'analogique (consommation électrique, afficheurs, stockage en mémoire...)
- fabriquants : Motorola (Freescale), Intel, Hitachi (Renesas), Microchip, Atmel, Philips (NXP), Texas Instruments, Analog Devices...
- deux grandes stratégies : des architectures propriétaires à un constructeur (Hitachi, Freescale) ou des architectures utilisées par plusieurs constructeurs (Intel 8051, ARM)

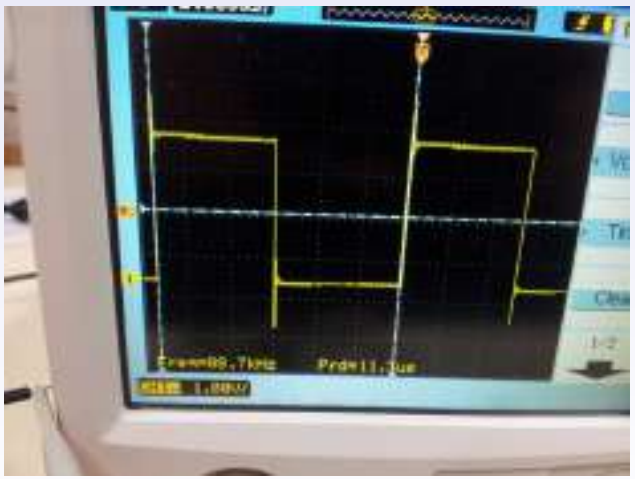
Pourquoi maîtriser ses outils

L'exemple d'Arduino : une bibliothèque qui vise, en faisant abstraction des détails du matériel, à rendre le code portable, au détriment des performances.

```
// Alternatively Blinking LED1 and LED2 for OLIMEXINO-32u4
// two general purpose LEDs at ARDUINO pins 7 and 9
int led1 = 7;

// the setup routine runs once when you press reset:
void setup() {
// initialize the digital pins as an outputs.
  pinMode(led1, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led1, HIGH);   // turn the LED on
  digitalWrite(led1, LOW);
}
```

```
#include <avr/io.h> //E/S ex PORTB
#define F_CPU 16000000UL

int main(void){
 DDRE |=1<<PORTE6;
 PORTE &= ~1<<PORTE6;

 while (1){
   PORTE=0x00;
   PORTE=PORTE6;
 }
 return 0;
}
```



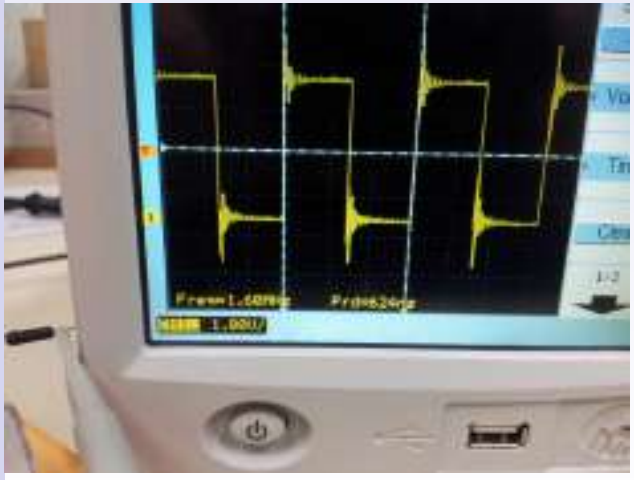


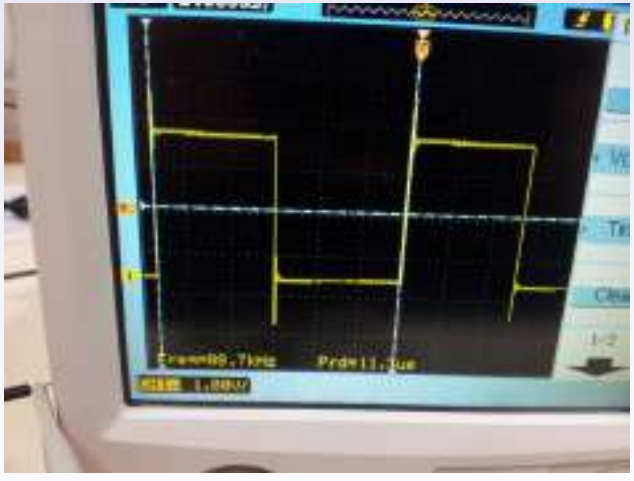
Pourquoi maîtriser ses outils

L'exemple d'Arduino : une bibliothèque qui vise, en faisant abstraction des détails du matériel, à rendre le code portable, au détriment des performances.

```
// Alternatively Blinking LED1 and LED2 for OLIMEXINO-32u4
// two general purpose LEDs at ARDUINO pins 7 and 9
int led1 = 7;

// the setup routine runs once when you press reset:
void setup() {
// initialize the digital pins as an outputs.
  pinMode(led1, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led1, HIGH);   // turn the LED on
  digitalWrite(led1, LOW);
}
```

```
main:
.L__stack_usage = 0
sbi 0xd,6
in r24,0xe
andi r24,lo8(-128)
out 0xe,r24
ldi r24,lo8(6)
.L2:
out 0xe,__zero_reg__
out 0xe,r24
rjmp .L2
```





Optimisation des ressources

On va constamment osciller entre logiciel et matériel. Le logiciel et le matériel sont équivalents (exemple du NOT) [1, p.11]. (Fig. 2).

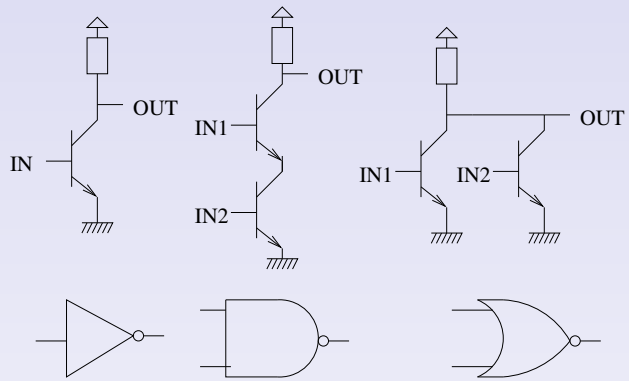


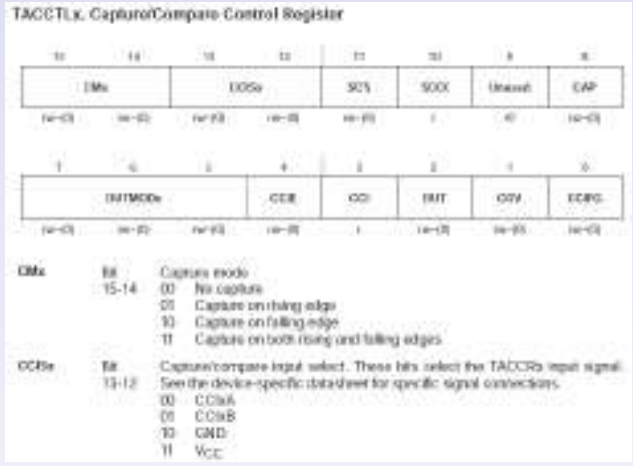
FIGURE – Quelque circuits de base

Cependant, le logiciel est reconfigurable, le matériel nécessite une refonte des circuits ⇒ accent sur le logiciel

Format des données

- Un système numérique vise à traiter des données qu'il faut donc représenter
- La base correspond aux nombres d'états possibles. Électronique numérique : deux états = binaire

Long et fastidieux à lire, ne s'utilise que pour décrire des registres dont chaque bit est identifié individuellement :



- Représentation hexadécimale des données : 0x00-0xFF, chaque lettre correspond à un quartet (*nibble*), compris entre 0 et 15.
- 8 bits $\in$ 0..255, 10 bits $\in$ 1024, 12 bits $\in$ 4096, 16 bits $\in$ 65536

Exemple : $0d42 = 0b00101010 = 0x2A = 2^5 + 2^3 + 2^1 = 2 \times 16 + 10$

Comprendre le fonctionnement d'un processeur

- Problèmes : comment encoder des nombres négatifs ?
- On pourrait ajouter un 1 sur le bit de poids le plus fort, mais n'est pas compatible avec la somme et 0 a deux représentations (problème lors de la comparaison).
- Solution : le complément à 2. On NOT tous les bits et on ajoute 1.

Exemple : la somme : 5-4= 0b0101+(NOT(0b0100)+1)= 0b0101+0b1100= 0b0001

Exemple : la multiplication : $(-2) \times 3$=(NOT(0b0010)+1)×0b0011=0b1110×0b0011=0b$\not{0}$ $\not{1}$ $\not{0}$1010 et NOT(0b1010)+1=0b0101+1=0b0110=6

Exercices : 2×3, $(-2) \times 3$, $5 \times (13)$, $5 \times (-13)$

Passer au bon nombre de bits dans le résultat *avant* de faire le produit

La multiplication est lourde, à éviter (×10=×8+×2 et décalages)

Encodages particuliers

- le BCD (*Binary Coded Decimal*) code sur 4 bits un nombre entre 0 et 9 : afficheurs 7 segments, RTC (affichage).
  09+1=10 et non 0A en BCD.
- ASCII : correspondance entre caractères affichables et un octet (0..255)

```
ASCII(7)                    Linux Programmer's Manual                    ASCII(7)

NAME
       ascii - the ASCII character set encoded in octal, decimal, and hexadec-
       imal

          2 3 4 5 6 7        30 40 50 60 70 80 90 100 110 120
         -------------      --------------------------------
       0:   0 @ P ` p     0:    (  2  <  F  P  Z  d   n   x
       1: ! 1 A Q a q     1:    )  3  =  G  Q  [  e   o   y
       2: " 2 B R b r     2:    *  4  >  H  R  \  f   p   z
       3: # 3 C S c s     3: !  +  5  ?  I  S  ]  g   q   {
       4: $ 4 D T d t     4: "  ,  6  @  J  T  ^  h   r   |
       5: % 5 E U e u     5: #  -  7  A  K  U  _  i   s   }
       6: & 6 F V f v     6: $  .  8  B  L  V  `  j   t   ~
       7:   7 G W g w     7: %  /  9  C  M  W  a  k   u  DEL
       8: ( 8 H X h x     8: &  0  :  D  N  X  b  l   v
       9: ) 9 I Y i y     9: '  1  ;  E  O  Y  c  m   w
       A: * : J Z j z
       B: + ; K [ k {
       C: , < L \ l |
       D: - = M ] m }
       E: . > N ^ n ~
       F: / ? O _ o DEL
```

Ne jamais utiliser d'accent et autres caractères non-ASCII

Masquages

- Probablement l'opération la plus courante sur un microprocesseur
- Les masques : OR pour conserver les 1, AND pour conserver les 0, XOR pour inverser [2]. Différence entre char et caractère ASCII.
- Tables de logique : 1 OR x=1, 0 AND x=0, x XOR x=0 et x XOR x#=1
- historiquement, on remplaçait `registre=0` par `xor registre,registre` (plus rapide/petit [1] [3, p.60])

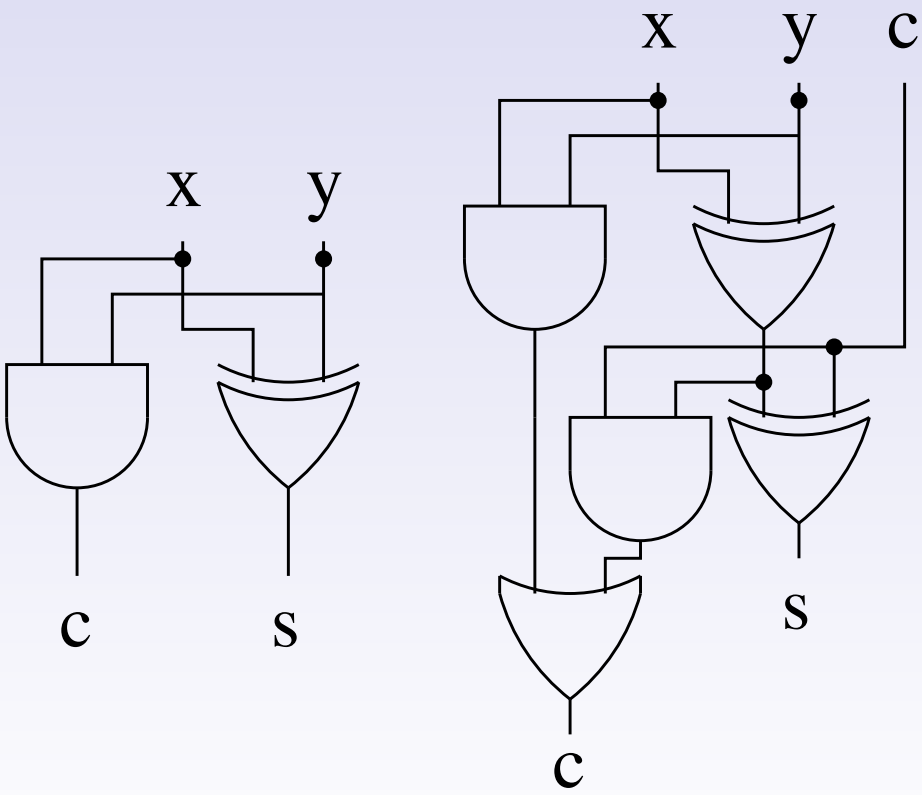
OU sur des bits distincts revient à une addition :
0x10 | 0x02=0x10+0x02=16+2.

Exemples : conserver le quartet de poids fort ? mettre à 1 les deux premiers bits ? Inverser le bit de poids le plus fort ?

1. architecture 8085 : XRA occupe 1 octet et prend 4 cycles, LDA occupe 3 octets et prend 13 cycles

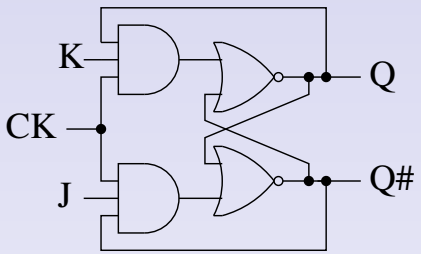
Opération arithmétique = logique

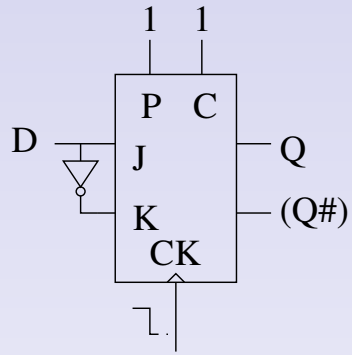
- Opérations sur les données : exemple de l'additioneur.
- Toutes ces opérations sont *asynchrones* : absence d'horloge
- La solution la plus simple n'est pas forcément à négliger.



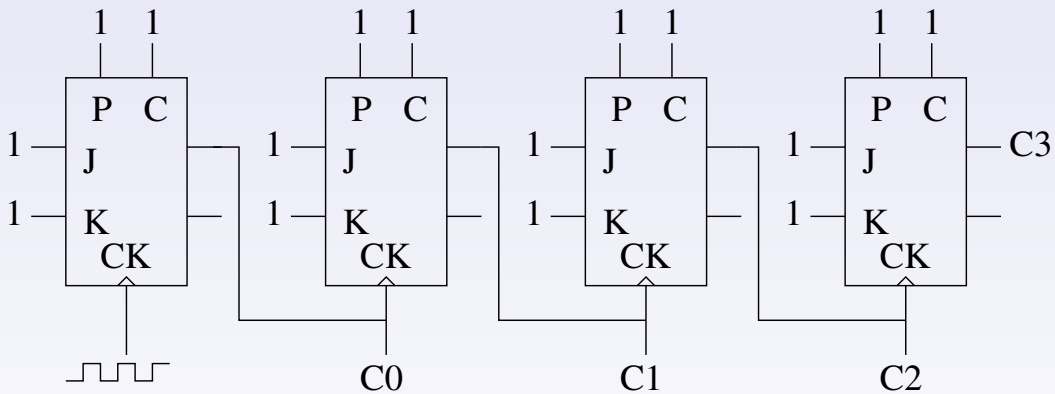
Mémoriser un état

Le rôle des bascules (74xx73) : bascule D et bascule J-K (J=Set, K=Reset, P=Preset#, C=Clear#) [2]





J	K	Q	Q'
0	0	Q	Q
1	0	Q	1
0	1	Q	0
1	1	Q	!Q



Toute opération de mémorisation doit être synchrone (sortie de la boucle de rétroaction de son état de mémorisation)

2. http ://computer.howstuffworks.com/boolean5.htm

Quelques composants fondamentaux

Comment effectuer ces opérations ?

- Fonctions logiques de base
- le demultiplexeur (74138) : N bits en entrée active 2^N sorties
- le comparateur (74688) : sortie passe à 0 si les 8 bits sont égaux
- les portes logiques : 7400=NAND, 7402=NOR, 7404=NOT, 7408=AND, 7432=OU ...
- Le latch (74573 niveau, 74574 front) est un composant essentiel : il retient sur sa sortie la valeur en entrée lors d'un front d'horloge. Éventuellement, il peut passer sa sortie en *haute impédance (hi Z)*. Élément essentiel lors du partage de ressources (un seul périphérique parle sur un bus à un instant donné, sinon conflit !).
- la sortie Open Collector

Rappels de C : structure d'un programme

1. par convention, l'exécution d'un programme débute *toujours* par la fonction `main()`
2. un programme s'exécute séquentiellement (à l'exception des interruption et programmes multithreadés)
3. les types[3] de variables simples (`type nom;`) :
   - `(unsigned) char` : 8 bits
   - `(unsigned) short` : 16 bits
   - `(unsigned) int` : 32 bits

     `http://kbase.redhat.com/faq/FAQ_52_4501.shtm`
   - `(unsigned long long)` : 64 bits
   - les nombres à virgule flottante simple précision `float` et double précision `double`
4. opération arithmétiques et logiques : `+`, `-`, `*`, `/`, `%` puis `|`, `&`, `~`, `^` `>>n` divise par 2^n (décalage à droite), `<<n` multiplie par 2^n (décalage à gauche)
5. cast : `(float)3=3.`, (type)variable pour convertir variable en type
6. tableau : `type var[taille]`, commence à 0 et finit à `taille-1`

3. Atmel AVR4027, *Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers*, Rev. 8453A-AVR-11/11

Rappels de C : structure d'un programme

- les types de variables composés : `struct nom prototype`

```
int main()
{
 struct complexe1 {int reel;int imaginaire;};
 struct complexe1 toto1;

 typedef int jmf;
 struct complexe2 {jmf reel;jmf imaginaire;};
 struct complexe2 toto2;

 toto1.reel=1; toto1.imaginaire=3;
 toto2.reel=2; toto2.imaginaire=4;
 printf("%d %d\r\n",toto1.reel,toto1.imaginaire);
 printf("%d %d\r\n",toto2.reel,toto2.imaginaire);
}
```

- affichage : `printf(prototype,arguments);` avec prototype
  - %d pour entier en décimal, %f pour flottant, %x pour entier en hexa
  - \r = retour chariot, \n nouvelle ligne, \t tabulation ...

Rappels de C : contrôle du flux

- `if (condition) {...}` ($<$, $>$, $<=$, $>=$, $==$; $!=$)
  ne pas confondre la comparaison $==$ avec l'assignation $=$
  tests OU `||`, ET `&&`
  booléen : 0=FAUX, !0=VRAI $\Rightarrow$ `while (1) ...`
- `for (debut;fin;increment) {...}`
- `while (condition) {}` et `do {...} while (condition);`
- ```
  switch (variable)
    {case (val1):
     case (val2): ...; break;
     case (val3): ...; break;
     default: ... break;
    }
  ```
- ```
  label: instruction ...
         goto label;
  ```
  équivalent C du jump de l'assembleur

Rappels de C : fichiers d'entête

Passage de paramètres :

```
int fonction (int a,int b)
{return(a*b);}
```

passage des valeurs ⇒ `a` et `b` ne peuvent pas être modifiées dans `fonction`.

Déclarer le prototype des fonctions définies dans d'autres parties du code (compilation séparée, bibliothèques)

- extension .h
- définition des types des fonctions

  ```
  int fonction (int ,int );
  ```

- déclarer les structures de données
- déclarer les constantes `#define constante valeur` (pas de ";" à la fin de la déclaration)
- deux bibliothèques standard : `stdio` et `stdlib`
- **JAMAIS DE CODE SOURCE DANS LES FICHIERS D'ENTÊTE**

Rappels de C : fichiers d'entête

```
int f(int a,int b)
{return(a*b);}

int main()
{int aa=2,bb=4;
 printf("%d\r\n",f(aa,bb));}
```

fichier `fctn.h`

```
int f(int ,int );
```

fichier `fctn.c`

```
int f(int a,int b)
{return(a*b);}
```

fichier `prog.c`

```
#include "fctn.h"

int main()
{int aa=2,bb=4;
 printf("%d\r\n",f(aa,bb));}
```

Objectif : séparer les fichiers des sources par fonctionnalités afin de **réutiliser** le code.

Rappels de C : utilisation de bibliothèques

- une bibliothèque implicite : `libc`
- le fichier d'entête définit les prototypes des fonctions fournies par la bibliothèque : `#include <math.h>`
- les dépendances des appels aux fonctions de la bibliothèque sont résolues lors de l'édition de liens (cf plus loin)
- les dépendances sont fournies dans les pages d'aide (`manpages`) :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main() {printf("%f",sqrt(3.));}
```

```
SQRT(3)                    Linux Programmer's Manual                    SQRT(3)
NAME
       sqrt, sqrtf, sqrtl - square root function
SYNOPSIS
       #include <math.h>

       double sqrt(double x);
       float sqrtf(float x);
       long double sqrtl(long double x);
```

- ATTENTION : calculer `sqrt(i*i+q*q)` prend 10 fois plus de temps que `i*i+q*q`

```
int main() {float i=1.,q=3.,m; int k;
            for (k=0;k<100000000;k++) m=(i*i+q*q); }
```

nécessite 3,4 s avec `sqrt` et 0,39 s sans (processeur avec FPU).

Rappels de C : les pointeurs

- Richesse (et faiblesse pour ses détracteurs) du langage pour permettre d'accéder au plus près de la mémoire.
- soit une variable `toto` : son emplacement mémoire est défini par `&toto`
- le C ne connaît la notion de tableau que sous forme d'une zone mémoire accessible par son adresse (pointeur) :
  `int tableau[255];` alloue 1020 octets à partir de l'adresse `tableau`
- allocation dynamique de mémoire : on définit un *pointeur*, sans définir la taille de la structure associée. *memory alloc* (`malloc()`) allouera la mémoire et la façon d'en interpréter le contenu
  `int *t;t=(int*)malloc(255*sizeof(int));` alloue l'espace pour 255 entiers
- accéder au contenu :`t[n]=t+n*sizeof(int);`

```
char c[4]={0x12,0x34,0x45,0x56};
short *s;
int *i;
s=(short*)c; printf("%x",s[1]);  // 5645
i=(int*)c;   printf("%x",i[0]);  // 56453412
```

- un tableau bidirectionnel est une concaténation des lignes

Rappels de C : les pointeurs

- passage de paramètre par pointeur : on ne passe pas la valeur mais l'emplacement mémoire ⇒ possibilité de modifier le contenu

```
void fonction(int *a)
{*a=*a+1;}

int main()
{int aa=1;
 fonction(&aa);printf("%d\r\n",aa);
}
```

renvoie 2

- subtilité pour les structures : "." devient ->

```
struct complexe1 {int reel;int imaginaire;};

void fonction(struct complexe1 *a)
{a->reel=a->reel+1;}

int main()
{struct complexe1 toto1;
 toto1.reel=1;
 fonction(&toto1);
 printf("%d\n",toto1.reel);
}
```

Rappels de C : utilisation de bibliothèques

Accès aux fichiers : `stdio`

Deux accès : formaté ou binaire

1. accès formaté au moyen de `fprintf()` et `fscanf()` :

```
FILE *f;
f=fopen("mon_fichier.txt","w");
fprintf(f,"%d %x %f",1,10,3.14);
fflush(f);
fclose(f);
```

2. accès direct au moyen de `write()` et `read()` :

```
int f,tableau[nombre];
f=open("mon_fichier.bin",O_RDWR);
write(f,tableau,nombre*sizeof(int));
close(f);
```

sous unix, tout périphérique (à l'exception des interfaces réseau/socket) est fichier $\Rightarrow$ `open("/dev/ttyS0",O_RDWR ...)`

Rappels de C : chaînes de caractères

- il n'existe pas de chaîne de caractères en C : tableau de caractères terminant par 0,
- fonctions associées à ces structures de données : `sscanf()` et `sprintf()`
- conversion chaîne-valeur : `atof()`, `atoi()` ...

```
char s[18]={'H','e','l','l','o','\0'};
printf("%s\r\n",s);
sprintf(s,"Bonjour %f",3.14159);
printf("%s\r\n",s);
```

ATTENTION aux trous de sécurité (que se passe-t-il si on autorise `sscanf` à recevoir plus de caractères que la taille de `s` ?)

Preprocesseur → assembleur → linker

Nécessité de comprendre le C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct matrice {int reel[1024*1024];} toto1;

void fonction(struct matrice *a) {a->reel[1]=a->reel[1]+1;} // ici 2..1001
void fonction2(struct matrice a) {a.reel[1]=a.reel[1]+1;}   // ici 2

int main()
{time_t t1,t2,t3;
 int k;
 toto1.reel[1]=1;                                          // ici 1
 time(&t1);
 for (k=0;k<1000;k++) fonction(&toto1);
 time(&t2);
 for (k=0;k<1000;k++) fonction2(toto1);
 time(&t3);
 printf("t1=%d t2=%d\r\n",t2-t1,t3-t2);                    // t1=0 t2=16
}
```

```
TIME(2)                    Linux Programmers Manual                    TIME(2)
NAME
       time - get time in seconds
SYNOPSIS
       #include <time.h>
       time_t time(time_t *t);
```

Les bus : un ensemble de signaux

- Notion de bus,
- exemple de décodage d'adresse pour accéder aux divers périphériques sur un même bus.
- Notion de banc mémoire (aujourd'hui automatisé par la MMU qui traduit adresse virtuelle en adresse physique, cache et gestion des priorités sur le bus).
- Bus de données, bus d'adresses et bus de contrôle.

Le bus de données fait transiter les informations (quoi), le bus d'adresse fait transiter les destinations (où), le bus de contrôle définit l'action en cours (comment : lecture ou écriture, 8 ou 16 bits ...).

Exemple : le PC104/ISA

Le bus prend son origine *dans* le processeur mais est accessible à l'extérieur.

Exemple du bus PC104/ISA [4] :

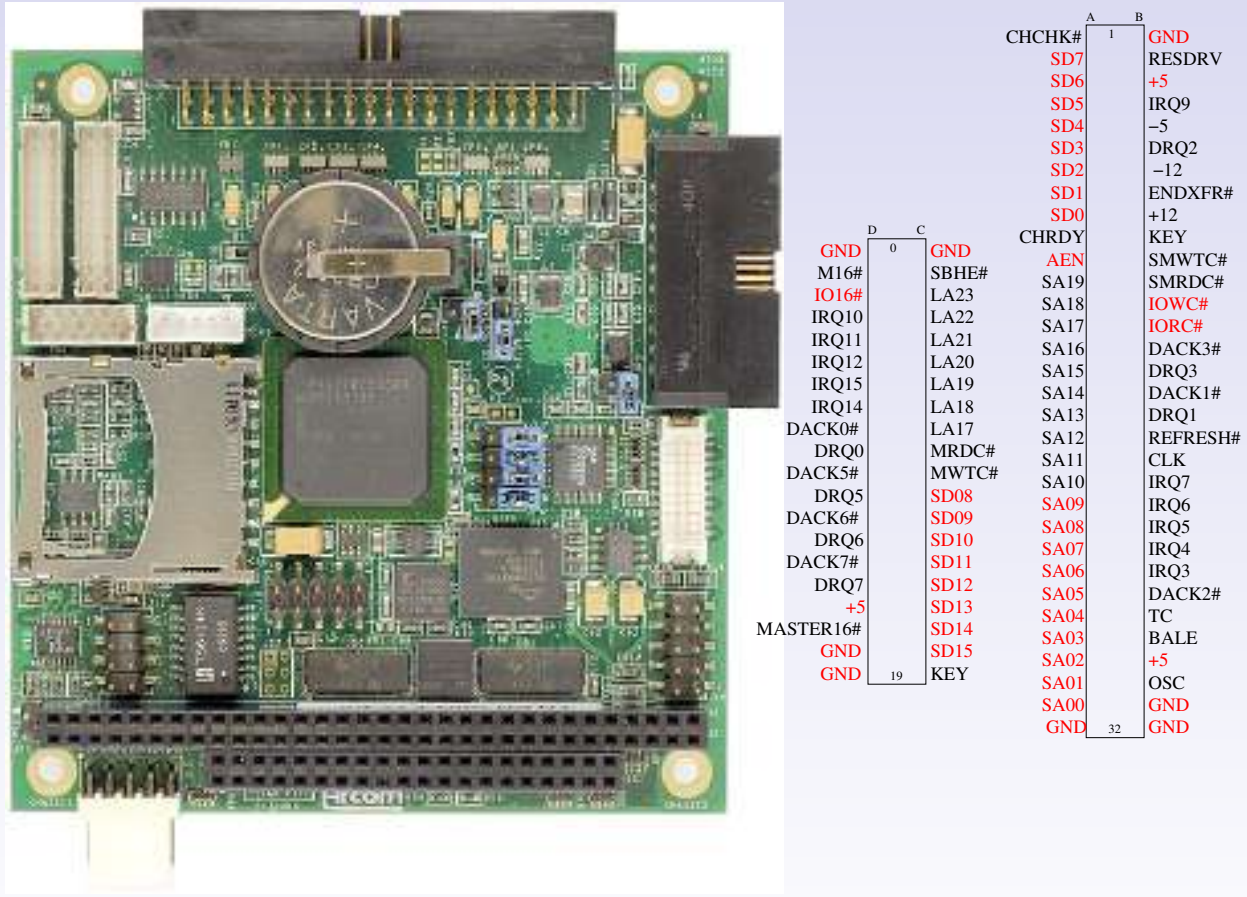


FIGURE – Bus PC104 (photo http://www.arcom.com/pc104-xscale-titan.htm)

Exploitation d'un bus

Tout composant possède un signal d'activation, généralement sur niveau bas : CS#[4].

FIGURE – Exemple de cartes d'acquisitions sur bus PC104 (Advantech PCM-4852)

L'objectif est de

- décoder l'adresse fournie sur le bus d'adresse (à qui parle-t-on ?)
- décoder l'action sur le bus de contrôle (écriture, lecture, accès mémoire ou port d'entrée sortie, transaction 8 ou 16 bits ?)
- répondre à cette action dans les délais (trop tard et on crée un conflit)

4. tout symbole suivi de # ou surligné est actif au niveau bas

Relation matériel-logiciel : ISA

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>

int main(int argc,char **argv)
{unsigned char val;
 ioperm(0x378,3,1);     // autorisation d'acces
  if (argc==1) {val=0xff;printf("%s val\n",argv[0]);}
     else val=atoi(argv[1]);
  outb(val,0x378);sleep(1);
}
```

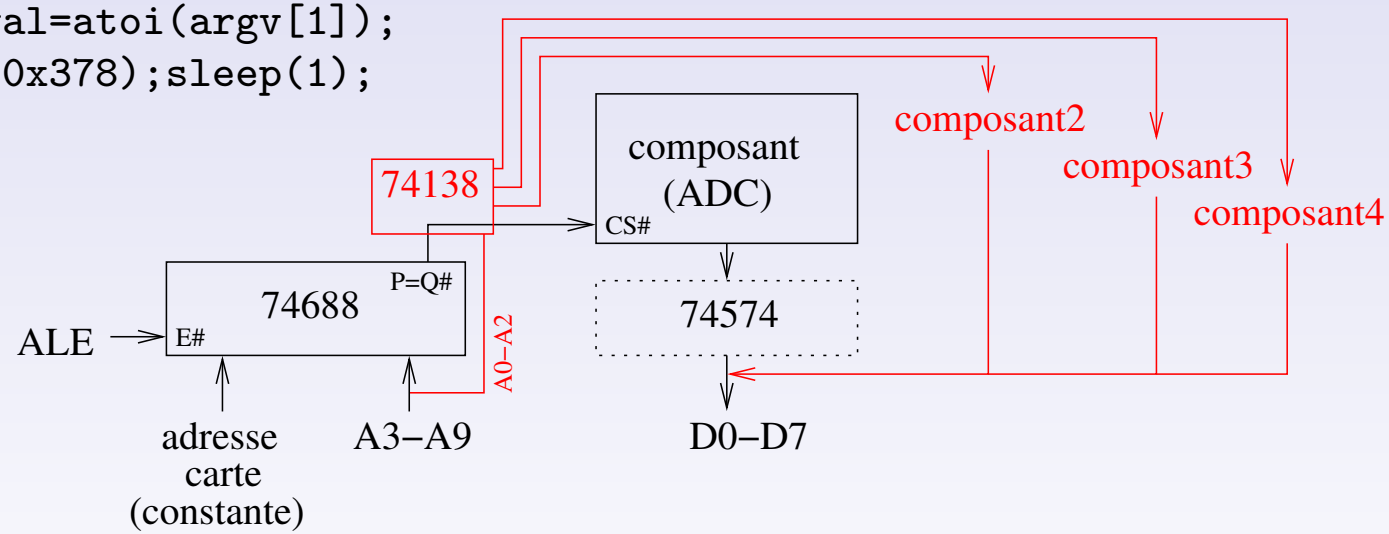
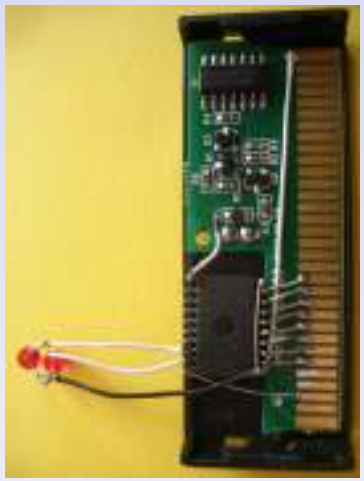


Schéma générique de connection d'un composant à un bus

Différence entre Motorola et Intel : dans le premier cas tout est mémoire, dans le second cas distinction entre `in`/`out` et `mov` [5].

Relation matériel-logiciel : NDS

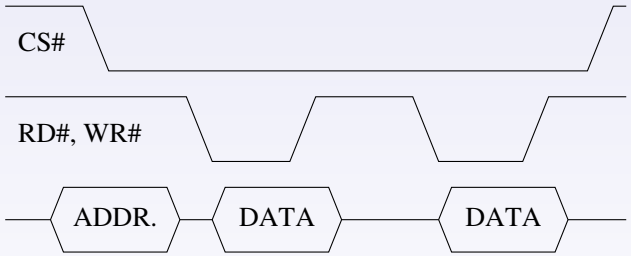
La plage d'adresses permettant l'accès à un périphérique est définie par le concepteur et documentée [5].



Bus = ensemble de broches

Adresse définit quel composant/fonction est actif
Ici, latch ou acquisition

Pour écrire la valeur 2=0b0000000000000010 sur le bus de données :
`*(unsigned short*)0x8000000=2`



5. http://nocash.emubase.de/gbatek.htm#dsmemorymaps

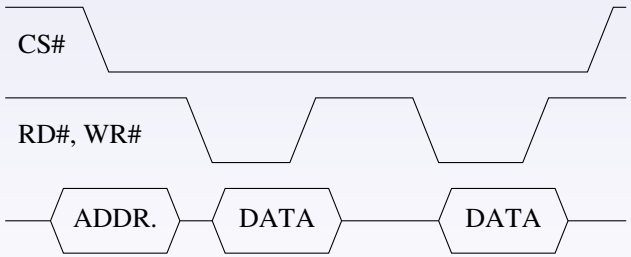
Relation matériel-logiciel : NDS

La plage d'adresses permettant l'accès à un périphérique est définie par le concepteur et documentée [5].

Bus = ensemble de broches

Adresse définit quel composant/fonction est actif
Ici, latch ou acquisition

Pour écrire la valeur 2=0b0000000000000010 sur le bus de données :
`*(unsigned short*)0x8000000=2`



5. http://nocash.emubase.de/gbatek.htm#dsmemorymaps

Consommation et puissance de calcul ...

- En technologie CMOS, les *transitions* d'état induisent la consommation de courant (vitesse×puissance≃1 pJ) [3] ⇒ plus il y a de périphériques et plus ils sont rapides, plus on consomme.
- Il faut donc choisir le processeur approprié pour son application [6].

6. $E = \sum_N 1/2 \times C \times V^2 \times num\_transitions$: la puissance baisse avec le carré de la tension ⇒ choisir des technologies à faible tension de fonctionnement

Conclusion

- **Bien maîtriser ses outils pour choisir le mieux approprié**, compromis entre coût/ puissance/ consommation/ logiciels de développement/disponibilité.
- compromis compacité/consommation/souplesse
- le numérique est il la meilleure solution par rapport à l'analogique ?
- le programmable est-il la meilleure solution par rapport aux portes logiques ? (robustesse aux environnements durs)

Architecture d'un processeur

- Différence entre microcontrôleur et microprocesseur : le microcontrôleur est un cœur de calcul entouré de périphériques (GPIO, timer, ADC/DAC, mémoire volatile ou non).
- Le bootloader : un petit bout de programme en ROM qui s'exécute sous certaines conditions pour transférer un programme depuis un PC vers le système embarqué (ISP : In System Programming)

Briques de base : le microcontrôleur

- le cœur du processeur (unité arithmétique et logique, décodeur d'instructions) est entouré de
- registres, zones mémoire pouvant contenir peu d'information, mais très rapidement accessible.
- À côté du cœur, de la mémoire volatile (RAM) et non volatile (ROM, flash ...)
- une partie de ces mémoires contient le programme qui s'exécute (Progam Counter), la RAM emmagasine les variables modifiable et...
- la pile, sorte de papier brouillon du processeur pour se rappeler

Dans l'ordre, on préférera accéder aux registres (souvent 1, 2 mais parfois 16 ou 256), puis la pile (stack pointer).

Au commencement...

- 8051, 68HC05/HC08/HC11, PIC : des automates avec un peu de logique
- Exemple d'application 8 bits : Gameboy (Z80), Apple][(6502), ...
- Coût réduit, simplicité de mise en œuvre (pas de pagination de la mémoire, quelques mnémoniques simples, peu d'interruptions et de périphériques), encombrement réduit (SO8 !).
- Une solution attractive : ATtiny qui est un "convertisseur intelligent", compacte et simple.

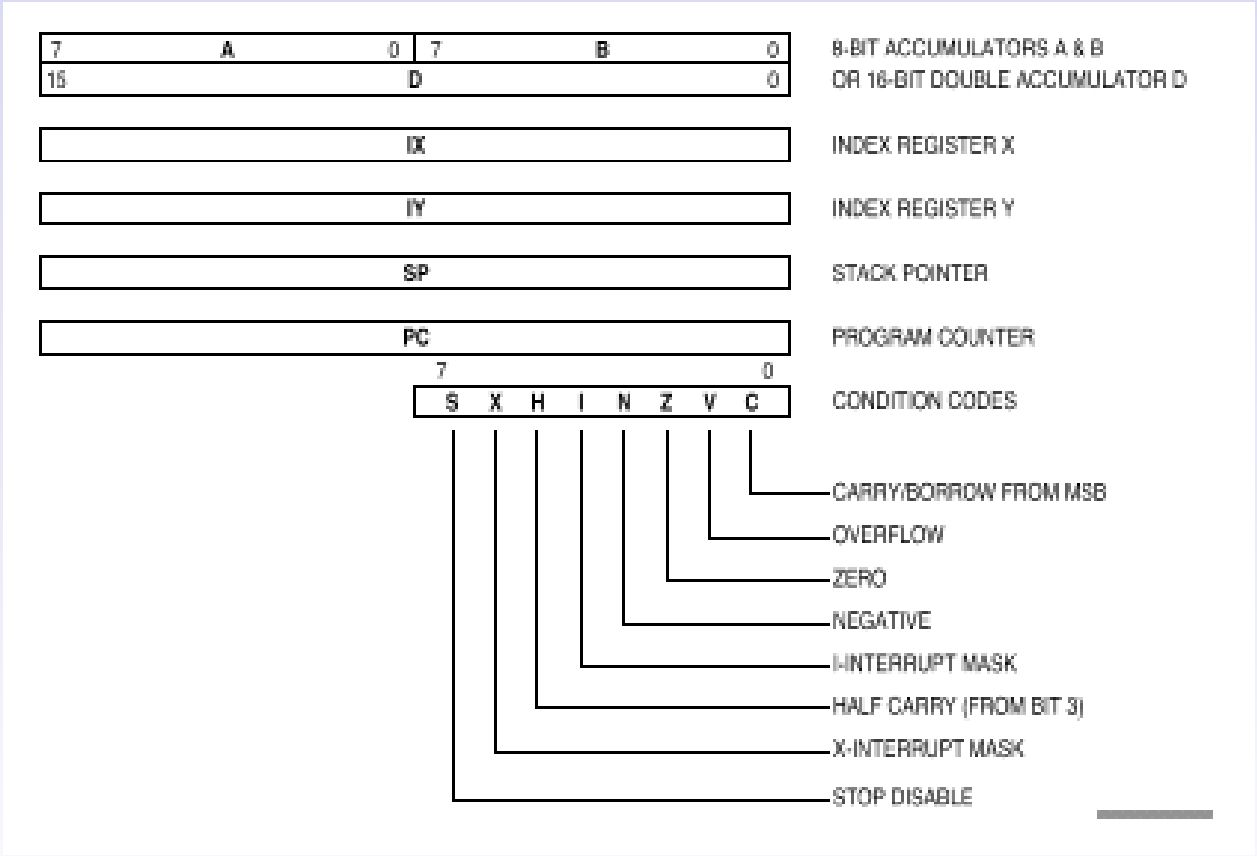
Généralités sur les cœurs de processeurs

- Deux grandes architectures : Harvard et Von Neuman (séparation de la plage données et opcodes pour 1 instruction par cycle).[7]
- Peu de place mémoire ⇒ programmation en assembleur.
- Outils génériques supportant de nombreuses architectures : `asl` et `asxxxx`[8]. On n'apprend qu'une syntaxe pour tous les microcontrôleurs.
- ISP (In System Programming) : bootloader permettant de charger en mémoire non-volatile un programme et de l'exécuter, par port RS232 ou JTAG[9]

7. Fetch-Decode-Execute en 1 cycle sur Harvard, plusieurs cycles sur Von Neuman
8. shop-pdp.kent.edu/ashtml/asxxxx.htm
9. Le bus JTAG : un outil standard pour le debuggage et la programmation.

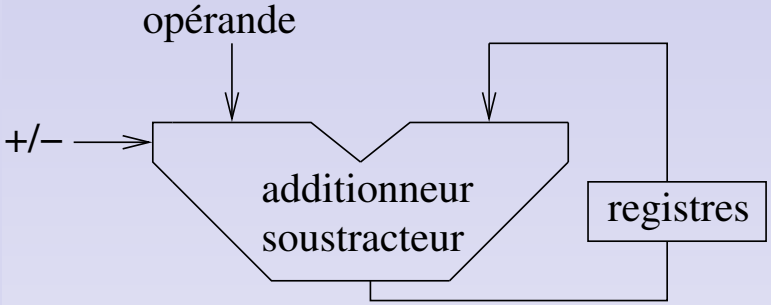
Architecture d'un processeur : les registres

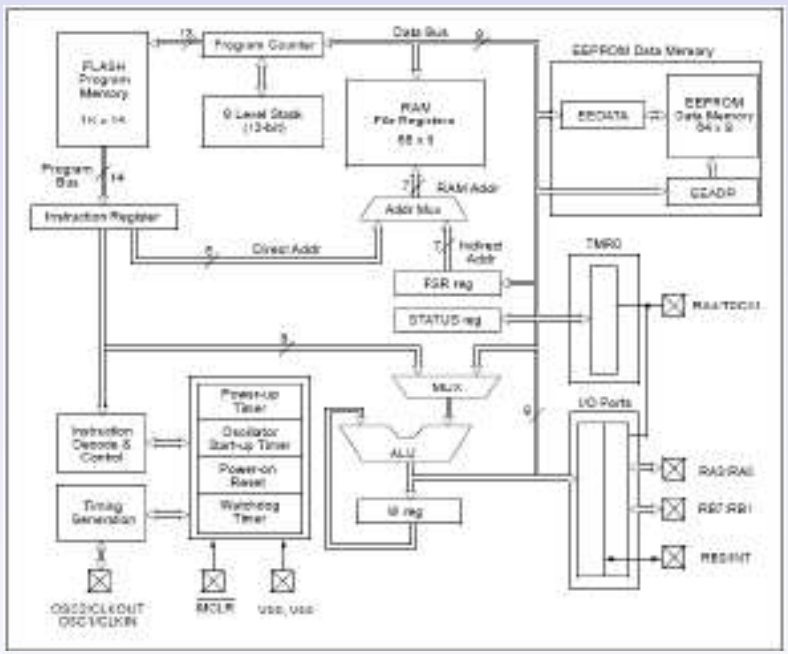
Les drapeaux (flags) : dépassement de capacité, retenue, négatif et zero.



Noter qu'une opération résultant en 0 génère un flag (Z) qui peut rapidement être testé.

Architecture d'un processeur : ALU





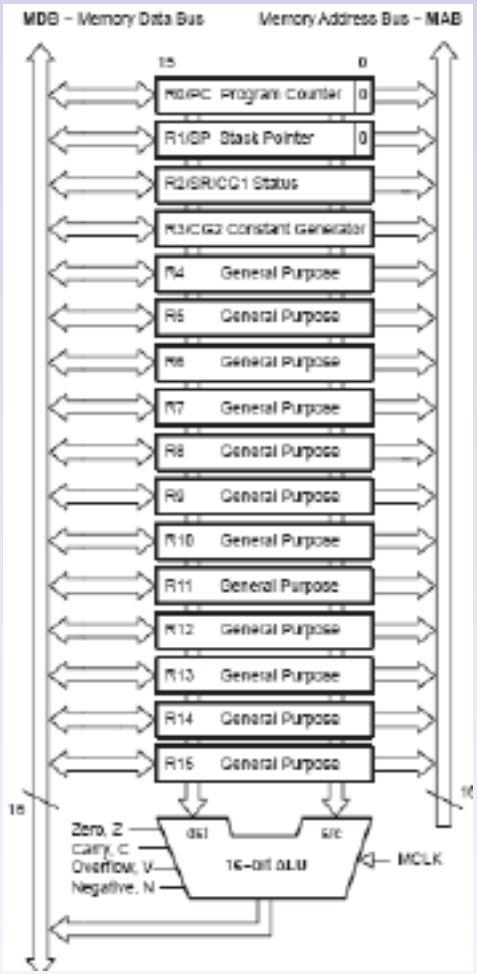
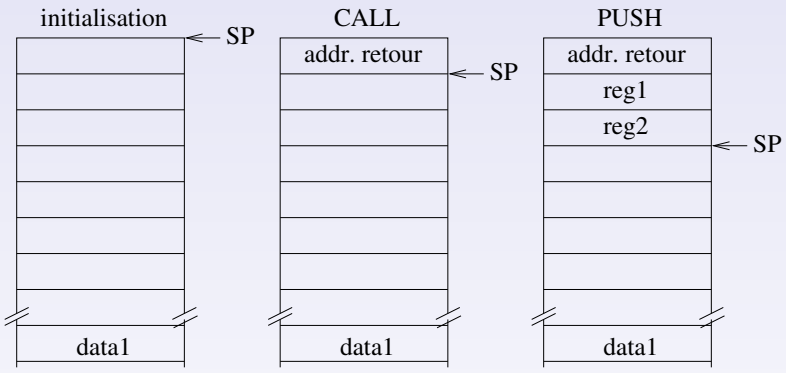


FIGURE – Unité arithmétique et logique : une multitude de registres remplace l'unique accumulateur en sortie de l'ALU (mémorisation des résultats intermédiaires). Milieu : architecture du Microchip PIC 16F84. Droite : architecture de l'ALU du MSP430.

Le program counter

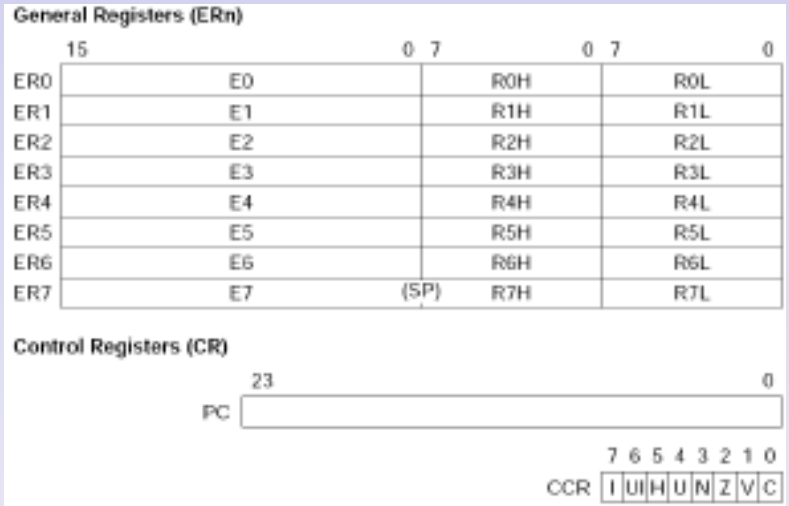
Où en est l'exécution [6].
Le PC en soi est caché, mais comprendre son utilisation évite bien des déboires :

- un saut (`jmp`) correspond à stocker une nouvelle valeur absolue dans le PC (PC=xxxx)
- un saut relatif (`bra`) correspond à une opération sur PC (PC+=xx)
- un appel de procédure (`call`) correspond à *empiler* PC, effectuer les opérations de la procédure, et revenir (`ret`) en dépilant le PC.
- PC est initialisé au Reset
- la pile sert à conserver temporairement des variables (`push`, `pop`) et lors de l'appel d'interruptions.



Manipuler la pile sans la remettre en état dans une procédure est une garantie de plantage

Premiers pas en assembleur



Liste des registres et drapeaux d'un microcontrôleur 32 bits (Renesas H8/3048) [10]

- Tous les assembleurs se ressemblent et fournissent les mêmes fonctionnalités
  - déplacer des valeurs
  - opérer sur des registres et opérations entre registres
  - sauter dans le programme
- un langage assembleur est un ensemble de mnémoniques représentant des opcodes
- un programme assembleur convertit les mnémoniques en opcodes + arguments (bijection)

10. www.datasheetcatalog.org/datasheets/restul/499485_DS.pdf

Les opcodes

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	NOP	AJMP	LJMP	RR	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC
0x10	JBC	ACALL	LCALL	RRC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC
0x20	JB	AJMP	RET	RL	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD
0x30	JNB	ACALL	RETI	RLC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC
0x40	JC	AJMP	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL
0x50	JNC	ACALL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL
0x60	JZ	AJMP	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL
0x70	JNZ	ACALL	ORL	JMP	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0x80	SJMP	AJMP	ANL	MOVC	DIV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0x90	MOV	ACALL	MOV	MOVC	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB
0xa0	ORL	AJMP	MOV	INC	MUL	?	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0xb0	ANL	ACALL	CPL	CPL	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE
0xc0	PUSH	AJMP	CLR	CLR	SWAP	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH
0xd0	POP	ACALL	SETB	SETB	DA	DJNZ	XCHD	XCHD	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ
0xe0	MOVX	AJMP	MOVX	MOVX	CLR	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0xf0	MOVX	ACALL	MOVX	MOVX	CPL	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV

Architecture CISC (*Complex instruction set computer*)
`http://www.win.tue.nl/~aeb/comp/8051/set8051.html`
ALU : un démultiplexeur qui convertit une valeur d'opcode en action sur les registres en entrée [7].

Les instructions (MSP430)

- `RRC` Rotate right through carry
- `SWPB` Swap bytes
- `RRA` Rotate right arithmetic
- `SXT` Sign extend byte to word
- `PUSH` Push value onto stack
- `CALL` Subroutine call ; push PC and move source to PC
- `RETI` Return from interrupt ; pop SR then pop PC
- `JNE/JNZ` Jump if not equal/zero
- `JEQ/JZ` Jump if equal/zero
- `JNC/JLO` Jump if no carry/lower
- `JC/JHS` Jump if carry/higher or same
- `JN` Jump if negative
- `JGE` Jump if greater or equal
- `JL` Jump if less
- `JMP` Jump (unconditionally)
- `MOV` Move source to destination
- `ADD` Add source to destination
- `ADDC` Add source and carry to destination
- `SUBC` Subtract source from destination (with carry)
- `SUB` Subtract source from destination
- `CMP` Compare (pretend to subtract) source from destination
- `DADD` Decimal add source to destination (with carry)
- `BIT` Test bits of source AND destination
- `BIC` Bit clear (dest &= $\sim$src)
- `BIS` Bit set (logical OR)
- `XOR` Exclusive or source with destination
- `AND` Logical AND source with destination (dest &= src)

Architecture RISC (*Reduced instruction set computer*)

Un exemple concret

```
[...]
Mainloop:       xor.b   #0xff,&P1OUT    ; clignote
                call    #delai          ; attend
                JMP     Mainloop        ; Endless Loop

delai:          mov.w   #0x4fff,tmp
attend:         dec.w   tmp
                jnz     attend
                ret
```

Faire clignoter une diode en inversant périodiquement la polarité du port.

```
00001160 <Mainloop>:
    1160: f2 e0 ff 00  xor.b #255,&0x0021 ;#0x00ff
    1164: 21 00
    1166: b0 12 6c 11  call #4460 ;#0x116c
    116a: fa 3f        jmp $-10       ;abs 0x1160

0000116c <delai>:
    116c: 37 40 ff 4f  mov #20479,r7 ;#0x4fff

00001170 <attend>:
    1170: 17 83        dec r7 ;
    1172: fe 23        jnz $-2        ;abs 0x1170
    1174: 30 41        ret
```

Les interruptions

- Une façon de rompre l'exécution séquentielle du programme : les interruptions.
- Un évènement interrompt l'exécution du programme, le fait passer dans une fonction appropriée (*courte*), et revient à la tâche principale : la table des vecteurs.

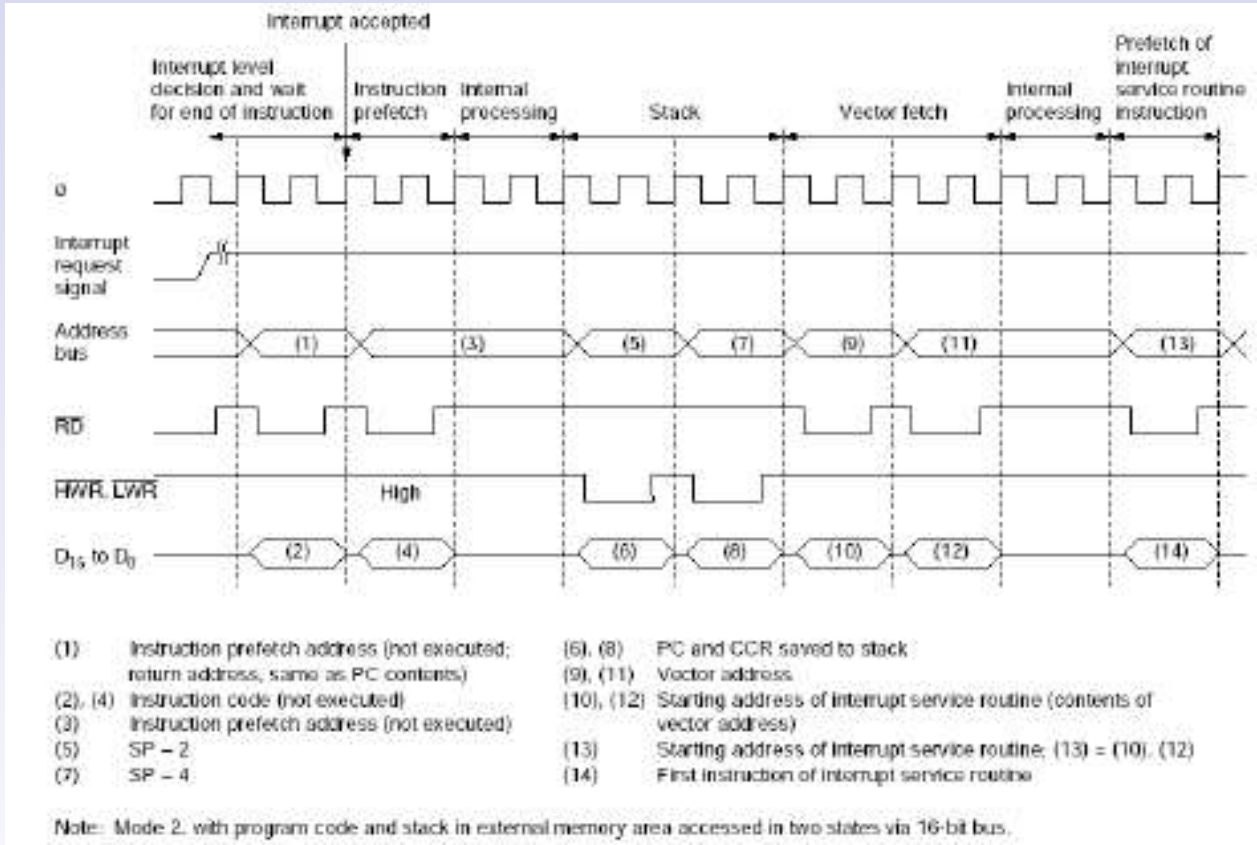
```
interrupt(TIMERA0_VECTOR ) wakeup Vector_TIMERA0(void)
{CCR0=32768/2;  // relancer le timer
 secondes++;    // variable globale
}
interrupt(TIMERB0_VECTOR ) /*enablenested*/ Vector_TIMERB0(void)
{TBCCR0=200;
 P1OUT |= 0x40;
}

int main()
{[...]
 eint();
 while (1) {LPM1;action();}
}
```

- Divers périphériques sont associés à des interruptions : la plupart des périphériques matériels (ports de communication, conversions), timer (évènement périodique), broches associées à une impulsion ...
- L'interruption timer permet de séquencer divers évènements avec des intervalles de temps prédéfinis [11], avec latence bornée.

```
void IRQ_Handler (void) {
  if (IRQSIG & GP_TIMER_BIT)  { /* Timer0 Interrupt          */
     ticks++;T1CLRI = 1;        /* Clear Timer 0 interrupt   */
  }
  if (IRQSIG & RTOS_TIMER_BIT)  {       // Timer0 Interrupt
     T0CLRI=0;
     switch (g2)
     ...
  }
}

int main(void)
{
 IRQ = IRQ_Handler;
 ...
}
```

11. multitâche préemptif, opposé au multitâche coopératif où chaque processus achevé donne la main au suivant

Timing de l'interruption : le H8/3048



Datasheet du H8/3048 : `http://www.datasheetcatalog.org/datasheets/restul/499485_DS.pdf`

Le watchdog

- Le watchdog : reset le processeur si une condition n'est pas vérifiée au bout d'un temps prédéfini.
- Associé aux timers : si le timer atteint la limite, le watchdog est déclenché
- Un élément dans la boucle ré-initialise le timer et garantit ainsi le bon fonctionnement du programme
- Évite un plantage à long terme du système informatique qui mettrait en péril l'instrument contrôlé.

Une interruption particulière : le RESET. Une zone mémoire *doit* exister pour cette destination.

Analyse du programme assembleur

Avantages :

- on sait exactement quel emplacement mémoire/registre est utilisé à quel moment
- la durée d'exécution peut être calculée au cycle près

MAIS

- code plus difficile à lire (la moindre opération prend beaucoup de lignes)
- code plus difficile à réutiliser et partager
- code plus difficile à faire évoluer dans le temps

Pourquoi ne pas utiliser la RAM : problème de relocalisation du code.

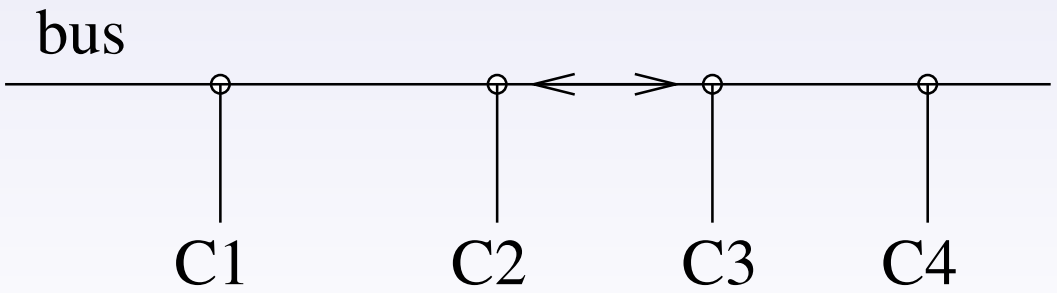
Les ports numériques

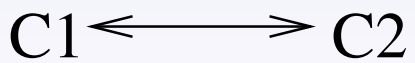
- Un latch, aussi appelé GPIO.
- Fonction souvent partagée avec une autre fonctionnalité (à sélectionner à l'initialisation)
- un port aime généralement *absorber* du courant
- un port est bidirectionnel : sa direction est définie par le programmeur
- un port en entrée *doit* avoir un état connu. Un interrupteur ne reste pas flottant : résistances de tirage.

Les bus de communications

Communication *sérielle* des données : plus rapide et moins coûteux (SATA v.s PATA)

- Notions de bus synchrones (partage de l'horloge) et asynchrone (chaque interlocuteur a sa propre horloge).
- synchrone est généralement plus rapide mais nécessite un signal additionnel pour l'horloge (sauf codage Manchester – ethernet)
- asynchrone nécessite l'accord préalable de tous les interlocuteurs sur le débit de communication (*baudrate*)
- très utilisé comme bus local entre un microcontrôleur et ses périphériques (peu de fils = réduction des coûts de câblage)





Les protocoles synchrones

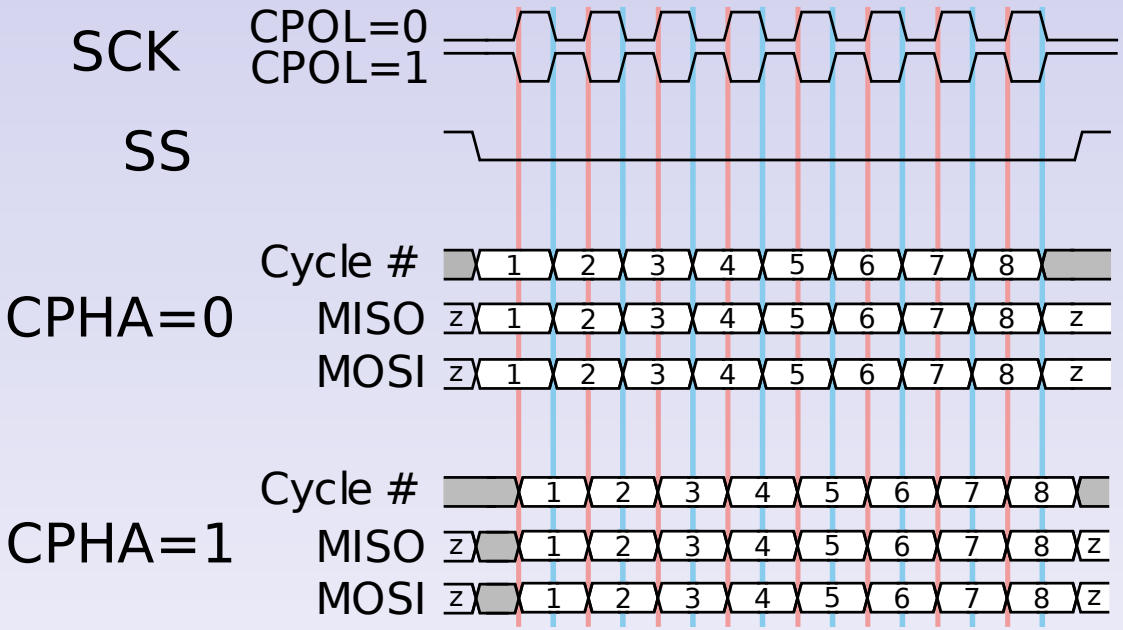


FIGURE – Chronogramme SPI (`http://en.wikipedia.org/wiki/Image:SPI_timing_diagram.svg`). Protocole synchrone : l'horloge est explicitement distribuée sur le bus.

- bus asymétrique : maître $\neq$ esclave, CS# désigne le(s) destinataire(s)
- version 2 fils d'un protocole synchrone : I^2C (bus de données est bidirectionnel).

Les protocoles asynchrones

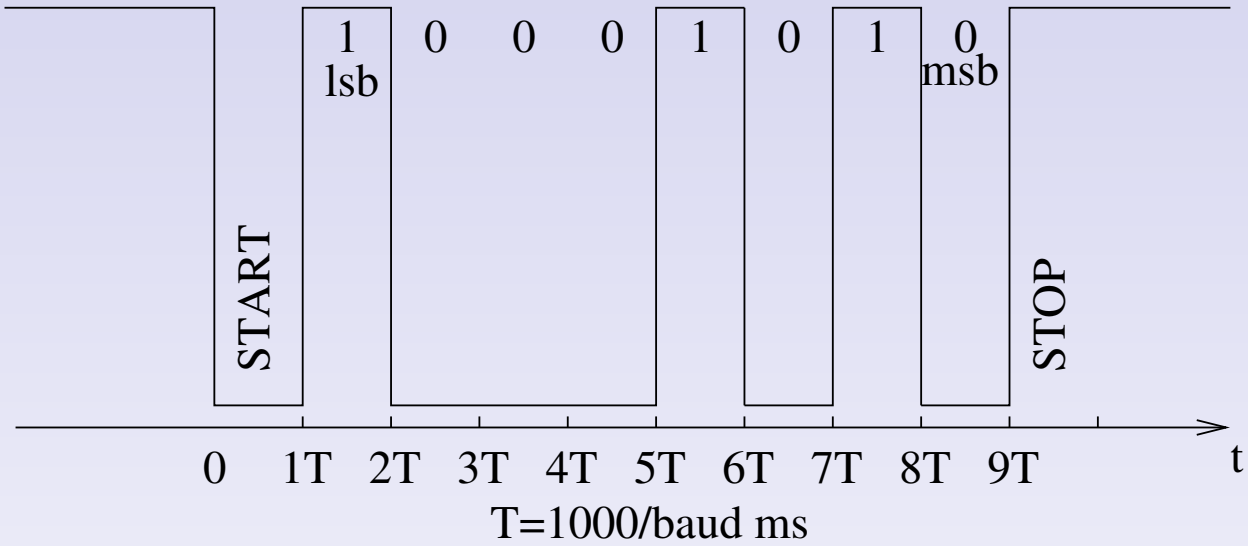


FIGURE – Chronogramme RS232. Pour 1200 bauds, $T = 833$ μs. En sortie du MAX232, les niveaux sont symétriques (± 12 V) *et inversés*. Protocole asynchrone : T doit être connu par les deux interlocuteurs.

RS232 : liaison point à point (2 interlocuteurs)

Stablité des références de temps

Quelle est la stabilite d'une montre ?

- une année contient $\simeq \pi \times 10^7$ secondes
- 1 minute/an=2.10^{-6} (2 ppm)

Quelle est la sensibilité en température d'un diapason à quartz ? en Si ?
$\Rightarrow$ le débit de communication est limité par la stabilité des horloges des deux interlocuteurs

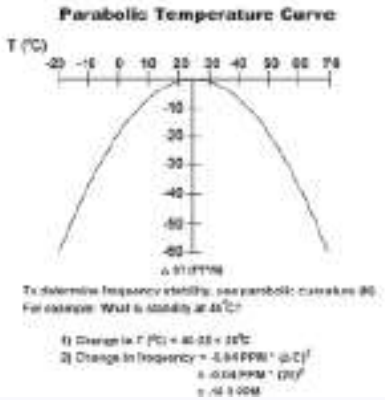
• PART NUMBER Learn More - Internet Required

Part Number	Model Number	Frequency Stability	Operating Temperature	Frequency
414LF-Frequency-xxxxx	FSRLF	See table	-40 °C ~ +85 °C	32.768 kHz

• STANDARD SPECIFICATIONS

PARAMETERS	MAX (unless otherwise noted)
Frequency	32.768 kHz
Frequency Tolerance @ 25°C	±20 PPM
Frequency Stability Temperature Coefficient	-0.04 PPM / (Δ°C)²
Temperature Range	
Turnover (TO)	+20°C ~ +30°C
Operating (TOPR)	-40°C ~ +85°C
Storage (TSTG)	-55°C ~ +125°C
Equivalent Series Resistance (RS)	50 kΩ
Load Capacitance (CL)	12.5 pF (Standard) 6 pF (Optional)
Insulation Resistance @ 100VDC	500 MΩ Min
Drive Level	1.0 µW
Aging per year	±3 PPM
Termination Finish	100% Sn

All specifications subject to change without notice.



a

a. www.foxonline.com

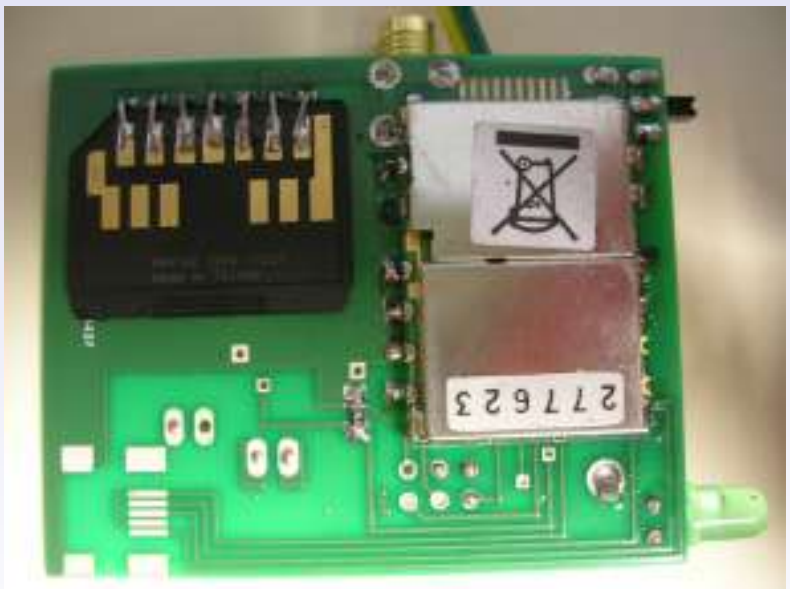
Cas du bus CAN

- Bus garantissant les transactions, utilisé notamment dans l'automobile et communications industrielles
- un identifiant (11 bits) suivi de données (8 octets), contenant un CRC
- contrôleur dédié gère les transactions : par exemple MCP2515 (SPI↔CAN)
- 2 fils pour des signaux différentiels robustes aux perturbations électromagnétiques
- asynchrone ⇒ tous les interlocuteurs se mettent d'accord au préalable sur le débit
- bus symétrique dans lequel l'identifiant désigne le(s) destinataire(s) du message

Exemple pratique

- un microcontrôleur 8 bits, cœur de 8051
- un récepteur GPS OEM (RS232, asynchrone)
- une carte de stockage de masse non volatile SD (SPI, synhrone)
- un convertisseur USB-RS232 (FT232RL)

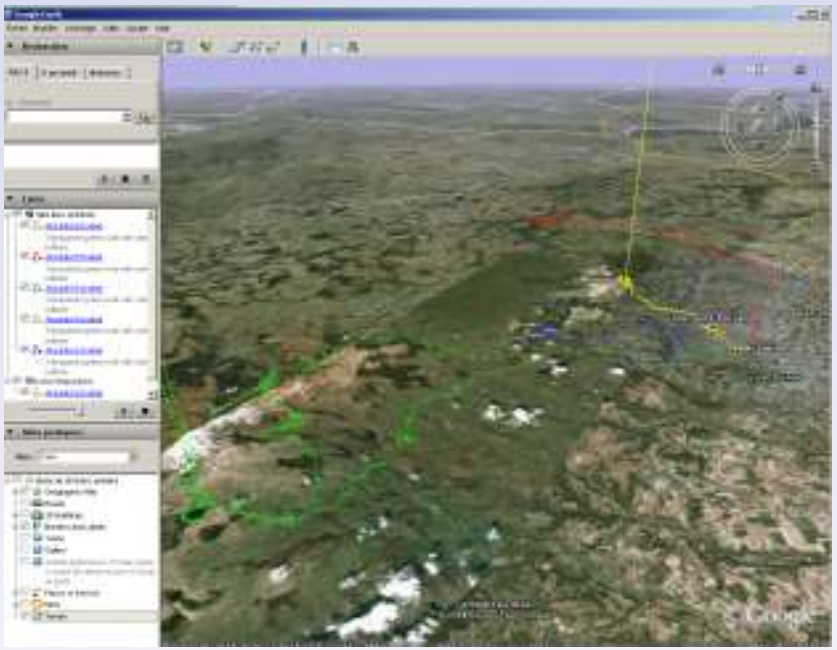
$\Rightarrow$ enregistrement de la position du récepteur toutes les secondes, avec une précision court terme du mètre, long terme de 7-10 m.
Autonomie : 20 h sur 4 piles NiMH, limité par le récepteur GPS [12]

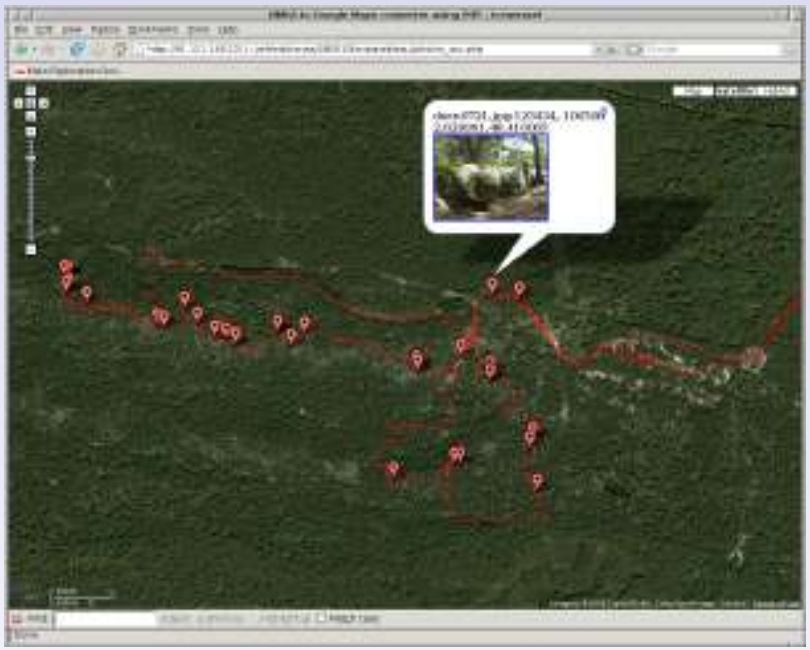


12. J.-M Friedt, É. Carry, *Enregistrement de trames GPS – développement sur microcontroleur 8051/8052 sous GNU/Linux*, GNU/Linux Magazine France **81** (Février 2006)

Exemple pratique

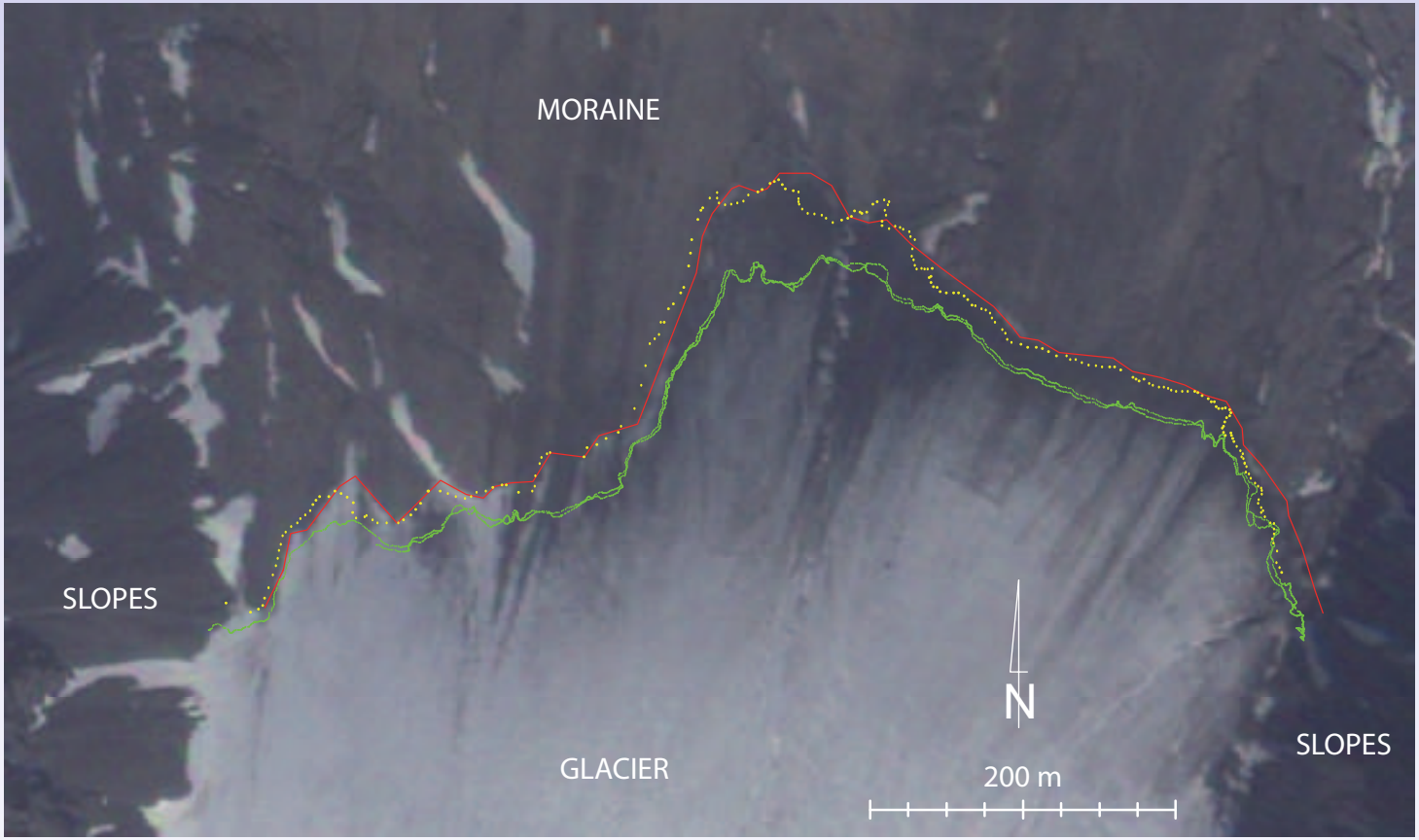
Exploitation des données





Si un système embarqué a pour vocation d'être utilisé, il *faut* une interface utilisateur.

Exemple pratique



Mesure de front de glacier, Sept. 2008 & Sept. 2011 [13]

13. É. Bernard, J.-M Friedt, A. Saintenoy, F. Toll, M. Griselin & C. Marlin, *Where does a glacier end ? GPR measurements to identify the limits between the slopes and the real glacier area*, Int. J. of Applied Earth Observation and Geoinformation (2012)

Les bus de communications

- Intermédiaire entre le protocole synchrone et asynchrone : codage de Manchester contenant beaucoup de transitions, et permettant ainsi de rapidement synchroniser les horloges des deux interlocuteurs (ethernet, radio tel que RFID).
- Un bus est soit disponible sous forme matérielle (UART), sinon on peut l'émuler de façon logicielle (problème de l'interruption à intervalles de temps régulier pour générer le bon baudrate).

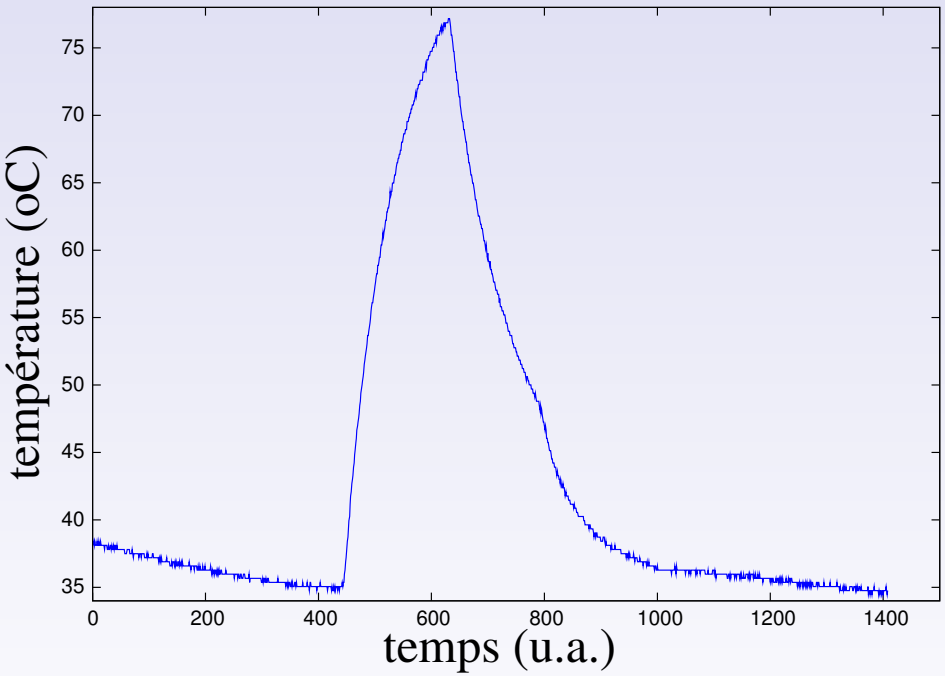
Conclusion :

- le protocole synchrone tend à être plus rapide et porter plus loin, mais nécessite plus de signaux (SPI nécessite 3 fils – MOSI, MISO et CK – là où un protocole asynchrone n'en nécessite que deux).
- I^2C utilise un bus de données bidirectionnel et un bit pour dire si on écrit ou on lit (idem pour PS2).

Conversion analogique-numérique

Probablement la partie la plus "amusante" des microcontrôleurs, interaction avec l'environnement :

- résolution : quelle est la plus petite variation mesurable (8-24 bits) ?
- stabilité : quelle référence (comment mesurer la tension qui nous alimente ?)
- rapidité (plus on est long, plus on consomme)
- antialiasing
- $V = V_{ref} \times \frac{val_{bits}}{max_{bits}}$
- 8 bits=48 dB
  10 bits=60 dB
  16 bits=96 dB
- **attention** à l'impédance d'entrée d'un ADC (2-10 kΩ) ⇒ mise en forme (ampli-op) du signal analogique (sortie basse impédance, Nyquist, exploitation de la dynamique de l'ADC par gain et offset). Pont diviseur ?



Les compteurs

- plusieurs compteurs, généralement avec plusieurs sources d'horloge avec des gammes de fréquences différentes
- utilisées dans le logiciel (séquencer), en entrée (Input Capture : mesure d'intervalles de temps) ou en sortie (PWM)
- de nombreux capteurs fournissent un signal modulé en largeur d'impulsion (oscillateur)

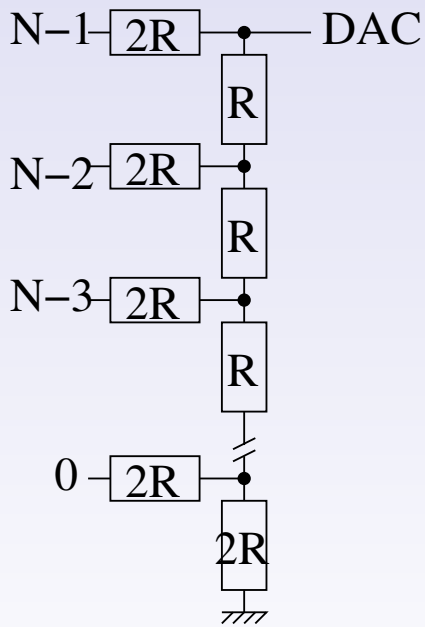
```
volatile int global=0,g2=0;

void IRQ_Handler (void) {
  if (IRQSIG & RTOS_TIMER_BIT)  {       // Timer0 Interrupt
        T0CLRI=0;
         switch (g2) {
          case 10: {programme(0);g2--;break;} // START
          case 9:
          case 8:
          case 7:
          case 6:
          case 5:
          case 4:
          case 3:
          case 2: {programme(global&0x01);global=global>>1;g2--;break;}
          case 1: {programme(1);g2--;break;}
          default: break;
} } }

void InitTimer(void){
  T0LD=34588;                                          // 41.78 MHz/34588=1208 bps
  T0CON=0xC0; IRQEN = GP_TIMER_BIT | RTOS_TIMER_BIT;  /* Configure Timer 0 */
}

void WriteChar(unsigned char ch)
{g2 = 10;global = ch;}
```

Conversion numérique-analogique

Faire interagir notre système numérique avec le monde extérieur (XXX contrôlé en tension, XXX=source de courant, oscillateur, ...), rétroaction.

- le plus stupide : le réseau de résistances
- indépendant des valeurs de résistances : R-2R
- PWM et filtre passe bas
- impédance de sortie

Accessoirement, la PWM pour commander un servo.



Estimation de l'autonomie d'un circuit (1)

- batteries de capacité données par le constructeur : par exemple 2000 mA.h
- consommation du système embarqué : par exemple 10 mA (attention, une LED = 20 mA !)
- autonomie [h]=(capacité [mA.h])/(courant consommé [mA]). Ici, l'autonomie théorique est de 2000/10=200 h. Cette autonomie est à réduire de la tension minimum nécessaire au bon fonctionnement du régulateur de tension, de la température et de l'usure de l'accumulateur.

Si le courant alterne entre plusieurs états, on prendra la valeur moyenne. Exemple : un système consomme 1 mA pendant 59 secondes et 99 mA pendant 1 seconde. Quelle est l'autonomie du système alimenté sur une pile de capacité 1 A.h ?
La consommation moyenne est $\frac{1\times 59+99\times 1}{60} = 2,6$ mA. L'autonomie est $1000/2,63 = 380$ h.

Estimation de l'autonomie d'un circuit (2)

- Un convertisseur DC-DC (travail à puissance constante, élévation de tension et moins de dissipation thermique – Maxim Switch Mode DC-DC Power Supplies tel que MAX1835 ou MAX1674) aura un meilleur rendement et fera fonctionner le système plus longtemps qu'un régulateur linéaire (zener, travail à courant constant : 78xx et successeurs, LDO), mais est source de bruit.

Généralités sur les microcontrôleurs 16 bits

- TI MSP430, Freescale HC12, Renesas H8 [14].
- Commence à fournir une puissance plus intéressant et notamment de travailler sur des mots.
- Plus que 8 bits $\Rightarrow$ travail sur des types de données de plusieurs octets (short, int, long long). Problème d'endianness lors de données sur plusieurs octets.
- les instructions se déclinent en action sur 8 ou 16 bits
- Exploitation des capacités par un langage de haut niveau (C)

14. Debian : `gcc-h8300-hms`

Programmation en C

Compilateur C libre : `mspgcc` pour Windows et GNU/Linux[15], documenté à `http://mspgcc.sourceforge.net/manual/index.html`
De façon générale, *tous* nos travaux se font avec `gcc`, le GNU Compiler Collection :

- un ensemble cohérent d'outils
- disponible sous forme de code source $\Rightarrow$ portable sur toute plateforme, adaptable à ses besoins, pérenne
- redistribuable librement
- compilateur C mais aussi Fortran, Ada, Java.
- l'utilisateur détient le contrôle sur toutes les commandes
- frontend pour appeler le préprocesseur (`cpp`, `gcc -E`), le compilateur (`cc1`, `gcc -s`), l'assembleur (`as`, `gcc -c`) et le linker (`ld`)
- séquence de compilation se découvre par `gcc -v`

15. `http://mspgcc.sourceforge.net/`

Séquence de compilation

```
#include <math.h>

#define i_init 3

int main()
{volatile int i=i_init;
 i=i+1;
}
```

gcc -E

```
...
extern double acos (double __x) __attribute__ ((__nothrow__ , __leaf__)); extern double →
      ↪__acos (double __x) __attribute__ ((__nothrow__ , __leaf__));

extern double asin (double __x) __attribute__ ((__nothrow__ , __leaf__)); extern double →
      ↪__asin (double __x) __attribute__ ((__nothrow__ , __leaf__));

extern double atan (double __x) __attribute__ ((__nothrow__ , __leaf__)); extern double →
      ↪__atan (double __x) __attribute__ ((__nothrow__ , __leaf__));
...
struct exception

  {
    int type;
    char *name;
    double arg1;
    double arg2;
    double retval;
  };
...
int main()
{volatile int i=3;
 i=i+1;
}
```

gcc -S

```
        .file   "t.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    $3, -4(%ebp)
        movl    -4(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -4(%ebp)
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 4.9.1-4) 4.9.1"
        .section        .note.GNU-stack,"",@progbits
```

objdump -dSt

```
08048274 <_init>:
 8048274:       53                      push   %ebx
 8048275:       83 ec 08                sub    $0x8,%esp
 8048278:       e8 83 00 00 00          call   8048300 <__x86.get_pc_thunk.bx>
 804827d:       81 c3 d7 13 00 00       add    $0x13d7,%ebx
 8048283:       8b 83 fc ff ff ff       mov    -0x4(%ebx),%eax
 8048289:       85 c0                   test   %eax,%eax
 804828b:       74 05                   je     8048292 <_init+0x1e>
 804828d:       e8 1e 00 00 00          call   80482b0 <__gmon_start__@plt>
 8048292:       83 c4 08                add    $0x8,%esp
 8048295:       5b                      pop    %ebx
 8048296:       c3                      ret
...
int main()
{volatile int i=i_init;
 80483cb:       55                      push   %ebp
 80483cc:       89 e5                   mov    %esp,%ebp
 80483ce:       83 ec 10                sub    $0x10,%esp
 80483d1:       c7 45 fc 03 00 00 00    movl   $0x3,-0x4(%ebp)
 i=i+1;
 80483d8:       8b 45 fc                mov    -0x4(%ebp),%eax
 80483db:       83 c0 01                add    $0x1,%eax
 80483de:       89 45 fc                mov    %eax,-0x4(%ebp)
 80483e1:       c9                      leave
 80483e2:       c3                      ret
}
```

Le problème ...

Comment représenter des entiers de plus d'un octet ?

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

int main()
{unsigned short court=0x1234;
 unsigned long  lon=0x12345678;
 unsigned char *c;
 printf("%x %x\n",court,lon);
 c=(unsigned char*)(&court);
 printf("%x %x\n",c[0],c[1]);                  // 2 octets
 c=(unsigned char*)(&lon);
 printf("%x %x %x %x\n",c[0],c[1],c[2],c[3]); // 4 octets
}
```

```
$ cat /proc/cpuinfo | grep name
model name  : Intel(R) Celeron(R) M processor      900MHz
$ ./integer
1234 12345678
34 12
78 56 34 12
```

```
# cat /proc/cpuinfo
CPU:            COLDFIRE(m5272)
Clocking:       65.4MHz
BogoMips:       43.62
# ./integer
1234 12345678
12 34
12 34 56 78
```

Le réseau est big-endian, ainsi que la JVM, SPARC, 68k et PPC. Intel x86, CRIS ETRAX et MSP430 sont little endian.

Les "solutions" ...

- Big endian : l'octet le plus significatif est placé à l'adresse la plus faible
- Little endian : l'octet de poids le plus faible est stocké à l'adresse la plus faible

$\Rightarrow$ big endian est le plus "lisible" (octet fort en premier)

Les types de données en C :

- (unsigned) char : 8 bits
- (unsigned) short : 16 bits
- (unsigned) int : 32 bits `http://kbase.redhat.com/faq/FAQ_52_4501.shtm`
- (unsigned long long : 64 bits

Éviter d'utiliser le long qui peut changer de taille (intel 32 ou 64 bits : `-m32` ou `-m64`)

```
int main()
{printf("%d %d %d ",sizeof(char),sizeof(short),sizeof(int));
 printf("%d %d\n",sizeof(long long),sizeof(int));
}

$ ./taille
1 2 4 8 4
```

Programmation en C

htonx et ntohx : sous linux : /usr/include/netinet/in.h

```
# if __BYTE_ORDER == __BIG_ENDIAN
/* The host byte order is the same as network byte order,
   so these functions are all just identity.  */
# define ntohl(x)       (x)
# define ntohs(x)       (x)
# define htonl(x)       (x)
# define htons(x)       (x)
# else
#  if __BYTE_ORDER == __LITTLE_ENDIAN
#   define ntohl(x)     __bswap_32 (x)
#   define ntohs(x)     __bswap_16 (x)
#   define htonl(x)     __bswap_32 (x)
#   define htons(x)     __bswap_16 (x)
#  endif
# endif
#endif
```

et dans /usr/include/bits/byteswap.h

```
#if defined __GNUC__ && __GNUC__ >= 2
# define __bswap_16(x) \
     (__extension__                                                   \
      ({ register unsigned short int __v, __x = (x);                  \
         if (__builtin_constant_p (__x))                              \
           __v = __bswap_constant_16 (__x);                           \
         else                                                         \
           __asm__ ("rorw $8, %w0"                                    \
                    : "=r" (__v) : "0" (__x) : "cc");                 \
           __v; }))
#else
/* This is better than nothing.  */
# define __bswap_16(x) \
     (__extension__                                                   \
      ({ register unsigned short int __x = (x); __bswap_constant_16 (__x); }))
#endif
```

Programmation en C

Le C sur microcontrôleur utilise la syntaxe du C mais des méthodes différentes. Nécessite une compréhension précise du fonctionnement du compilateur :

- utilisation courante des pointeurs puisqu'on maîtrise l'adresse à laquelle se trouve chaque variable
- pointeur et écriture dans un registre d'adresse définie en C (*interdit sur les architectures à MMU* : `*(type *)(adresse)=valeur;`
- passage de paramètres sur la pile de taille réduite
- optimisations abusives : `volatile`
- peu de mémoire $\Rightarrow$ en l'absence de contrôle de la pile, risque de dépassement $\rightarrow$ plantage !
- les arguments des fonctions se passent sur la pile ou par registre. Penser à passer un pointeur plutôt que la structure associée.
- des variables globales sont inévitables : échange d'information avec un gestionnaire d'interruptions.

volatile

Si on optimise sans volatile ... il ne reste rien !

Dans les deux cas : `msp430-gcc -O2 -mmcu=msp430x149 -D_GNU_ASSEMBLER_ -S`

```
int main()
{volatile unsigned short j=3,k;
 for (k=0;k<15;k++) {j+=k;}
}
```

```
        mov     #(__stack-4), r1
/* prologue end (size=2) */
        mov     #1lo(3), @r1
        mov     #1lo(0), 2(r1)
        cmp     #1lo(15), 2(r1)
        jhs     .L8
.L6:
        add     2(r1), @r1
        add     #1lo(1), 2(r1)
        cmp     #1lo(15), 2(r1)
        jlo     .L6
.L8:
```

```
int main()
{unsigned short j=3,k;
 for (k=0;k<15;k++) {j+=k;}
}
```

```
main:
        mov     #(__stack-0), r1
/* prologue end (size=2) */
/* epilogue: frame size=0 */
        br      #__stop_progExec__
```

De l'utilité de connaître l'assembleur

```
int main()
{volatile unsigned short j=3,k;
 for (k=1;k<15;k++) {j+=k;}
}
```

```
.L6:
        add     2(r1), @r1
        add     #llo(1), 2(r1)
        cmp     #llo(15), 2(r1)
        jlo     .L6
```

```
int main()
{volatile unsigned short j=3,k;
 for (k=15;k>0;k--) {j+=k;}
}
```

```
.L6:
        add     2(r1), @r1
        add     #llo(-1), 2(r1)
        jne     .L6
```

Comparaison de compilateurs

Dans tous les cas :

```
void delay (int length) {while (length >=0) length--;}
  // delay(5000)=1 ms
```

arm-thumb-elf-gcc (GCC) 4.0.0

```
196             delay:
197                     @ lr needed for prologue
198 0000 00E0           b       .L2
199             .L3:
200 0002 0138           sub     r0, r0, #1
201             .L2:
202 0004 0028           cmp     r0, #0
203 0006 FCDA           bge     .L3
204                     @ sp needed for prologue
205 0008 7047           bx      lr
207 000a 0000           .align  2
208                     .global jmf_putchar
209                     .code 16
210                     .thumb_func
```

Keil ARM Compiler V2.42

```
  47: void delay (int length) {       // delay(5000)=1 ms
00000000  B401  PUSH    {R0}
  48:     while (length >=0) length--;
00000002  E003  B       L_1  ; T=0x0000000C
00000004      L_3:
00000004  A800  ADD     R0,R13,#0x0
00000006  6801  LDR     R1,[R0,#0x0] ; length
00000008  3901  SUB     R1,#0x1
0000000A  6001  STR     R1,[R0,#0x0] ; length
0000000C      L_1:
0000000C  A800  ADD     R0,R13,#0x0
0000000E  6800  LDR     R0,[R0,#0x0] ; length
00000010  2800  CMP     R0,#0x0
00000012  DAF7  BGE     L_3  ; T=0x00000004
  49: }
00000014  B001  ADD     R13,#0x4
00000016  4770  BX      R14
00000018      ENDP ; 'delay?T'
```

Fonctionnalités égales mais temps d'éxécution différents

Déverminage de compilateur (AVR, gcc-avr 4.8.1)

sans volatile

```
[...]
char cmpt=5;

ISR(TIMER1_OVF_vect)
{ cmpt++;
  if(cmpt>99) cmpt=0;
  if (cmpt == 5) PORTC^=0X02;
}

int main(void)
{ TCCR1A=0x40;TCCR1B=0x4A;TCCR1C=0x80;
  TIMSK1=0x23;OCR1A=0;
  sei();
  DDRC = 0XFF ;
  while (1)
    if (cmpt == 5) PORTC^=0X01;
}
```

```
[...]
  22:coin.c        ****   TIMSK1=0X23;OCR1A=0;
 197 0012 83E2                ldi r24,lo8(35)
 198 0014 8093 6F00           sts 111,r24
 199 0018 1092 8900           sts 136+1,__zero_reg__
 200 001c 1092 8800           sts 136,__zero_reg__
  24:coin.c        ****   sei();
 205 0020 7894                sei
  26:coin.c        **** DDRD = 0X0F ;
 210 0022 8FE0                ldi r24,lo8(15)
 211 0024 8AB9                out 0xa,r24
  27:coin.c        **** DDRC = 0XFF ;
 214 0026 8FEF                ldi r24,lo8(-1)
 215 0028 87B9                out 0x7,r24
  30:coin.c        **** while (1)
  31:coin.c        ****         {
  32:coin.c        ****         if (cmpt == 5)
  33:coin.c        ****                 PORTC^=0X01;
 218 002a 91E0                ldi r25,lo8(1)
 219                      .L9:
  32:coin.c        ****         if (cmpt == 5)
 222 002c 8091 0000           lds r24,cmpt
 223 0030 8530                cpi r24,lo8(5)
 224 0032 01F0                breq .L7
 225                      .L10:
 226 0034 00C0                rjmp .L10
 227                      .L7:
 230 0036 88B1                in r24,0x8
 231 0038 8927                eor r24,r25
 232 003a 88B9                out 0x8,r24
 233 003c 00C0                rjmp .L9
```

Déverminage de compilateur (AVR, gcc-avr 4.8.1)

avec volatile

```
[...]
volatile char cmpt=5;

ISR(TIMER1_OVF_vect)
{ cmpt++;
  if(cmpt>99) cmpt=0;
  if (cmpt == 5) PORTC^=0X02;
}

int main(void)
{ TCCR1A=0x40;TCCR1B=0x4A;TCCR1C=0x80;
  TIMSK1=0x23;OCR1A=0;
  sei();
  DDRC = 0XFF ;
  while (1)
    if (cmpt == 5) PORTC^=0X01;
}
```

```
[...]
 22:coin.c        ****   TIMSK1=0X23;OCR1A=0;
194 0012 83E2                ldi r24,lo8(35)
195 0014 8093 6F00           sts 111,r24
196 0018 1092 8900           sts 136+1,__zero_reg__
197 001c 1092 8800           sts 136,__zero_reg__
 23:coin.c        ****
 24:coin.c        ****   sei();
202 0020 7894                sei
 26:coin.c        **** DDRD = 0X0F ;
207 0022 8FE0                ldi r24,lo8(15)
208 0024 8AB9                out 0xa,r24
 27:coin.c        **** DDRC = 0XFF ;
211 0026 8FEF                ldi r24,lo8(-1)
212 0028 87B9                out 0x7,r24
 30:coin.c        **** while (1)
 31:coin.c        ****         {
 32:coin.c        ****         if (cmpt == 5)
 33:coin.c        ****                 PORTC^=0X01;
215 002a 91E0                ldi r25,lo8(1)
216                      .L7:
 32:coin.c        ****         if (cmpt == 5)
219 002c 8091 0000           lds r24,cmpt
220 0030 8530                cpi r24,lo8(5)
221 0032 01F4                brne .L7
224 0034 88B1                in r24,0x8
225 0036 8927                eor r24,r25
226 0038 88B9                out 0x8,r24
227 003a 00C0                rjmp .L7
```

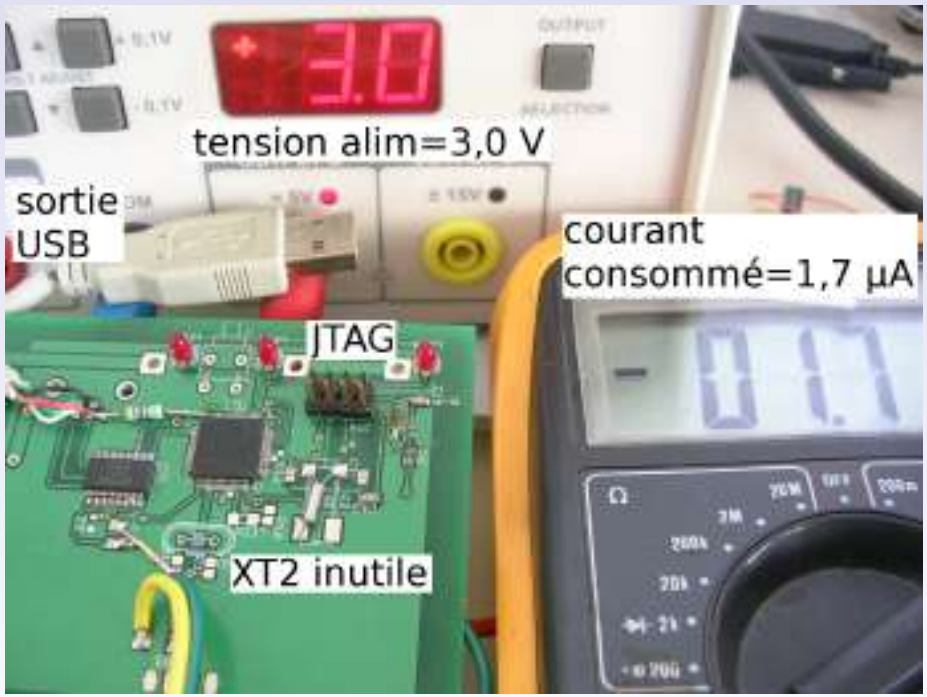
Le MSP430

Intérêt du MSP430 [9] : très faible consommation [16], divers modes de mise en veille, programmation sous gcc (programmation en C).

- 0.1 μA RAM retention
- 0.8 μA real-time clock mode
- 250 μA / MIPS active

Ce cœur met à disposition

- R1=pile (Stack Pointer)
- R4-R15 registres 16 bits
- deux oscillateurs
- 2 KB RAM (0x200-0x9ff)
- 60 KB flash



16. p.1-2 de MSP430x1xx Family Users Guide à http://www.ti.com/litv/pdf/slau049f

Le C n'est pas toujours bien

- l'ANSI-C ne fournit que peu de fonctionnalités
- le programmeur sur OS a souvent pris l'habitude de fonctionnalités additionnelles : utilisation de fonctions gourmandes telles que malloc et donc printf.
- un programme C cache beaucoup d'opérations au programmeur : qui initialise la pile ? qui initialise les ports ? `crt0.S`
- émulation logicielle du calcul flottant et fonctions mathématiques associées

Comment économiser de l'énergie

Dormir [17]

- les périphériques sont activées volontairement : n'utiliser que celles qui sont nécessaires
- le MSP430 contient 5 modes de veille, mais certains endorment le périphérique sensé nous réveiller
- en l'absence de mode de veille, couper l'alimentation et la rétablir quand nécessaire (horloge externe, DS1305 ou plus récent)
- retirer les options inutiles : LEDs, résistances de tirage faibles
- adapter son algorithme : compression puis transfert, stockage local, ...

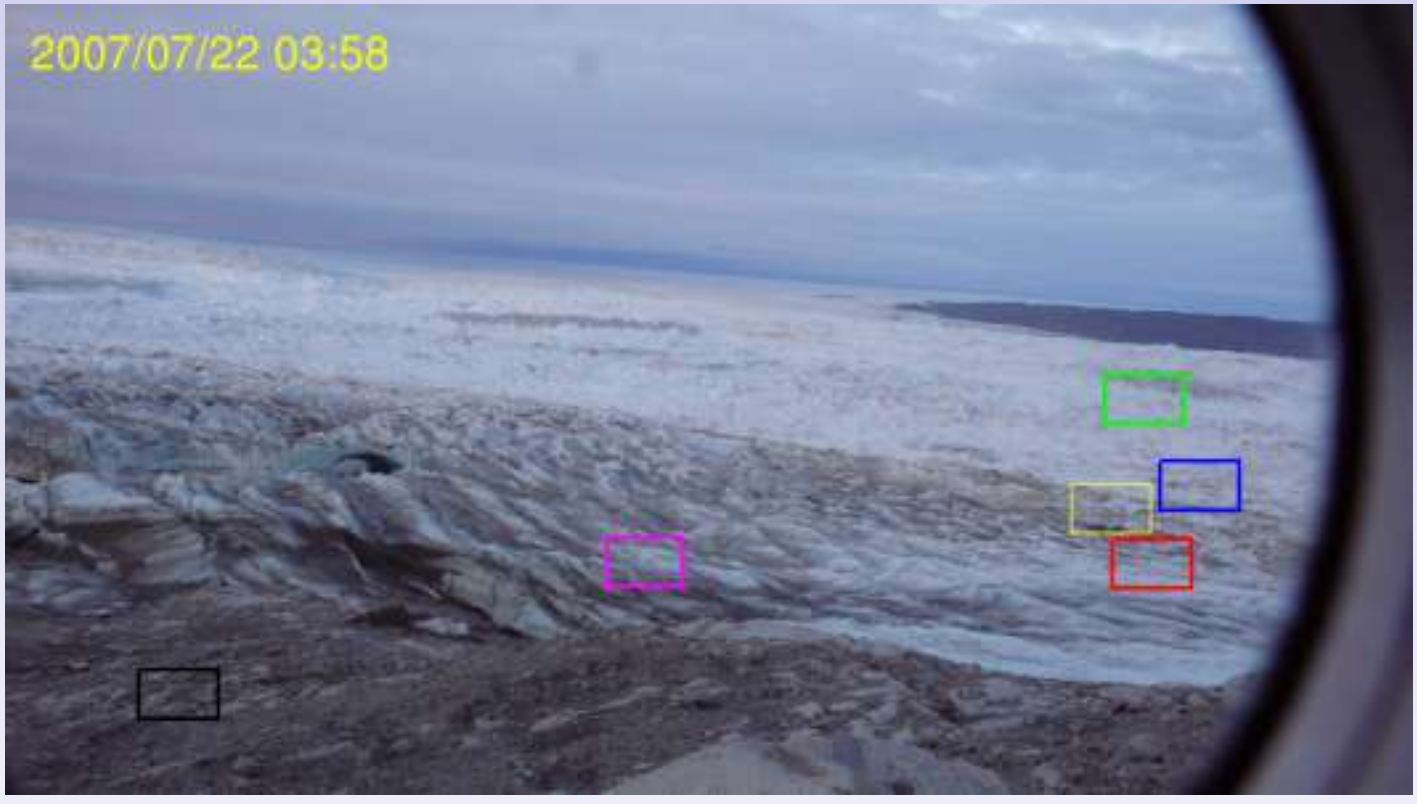
17. focus.ti.com.cn/cn/lit/an/slaa294a/slaa294a.pdf

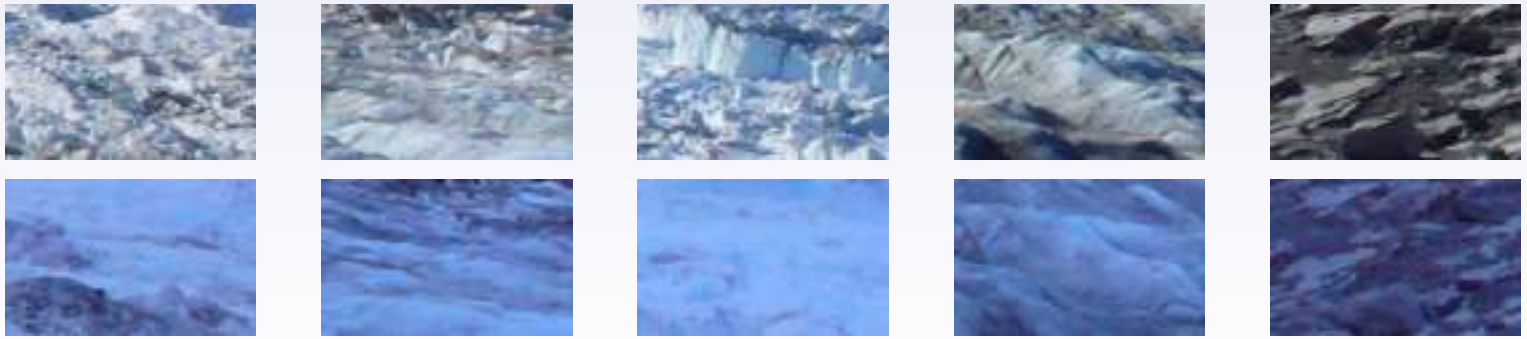
Exemple d'application embarquée

- un MSP430 se réveille toutes les secondes pour gérer une horloge temps réel logicielle [18]
- si l'alarme est atteinte, allumage de l'appareil photo et prise de vue
- test de la présence du câble USB pour la communication

18. http://www.moreauluc.com/

Exemple d'exploitation

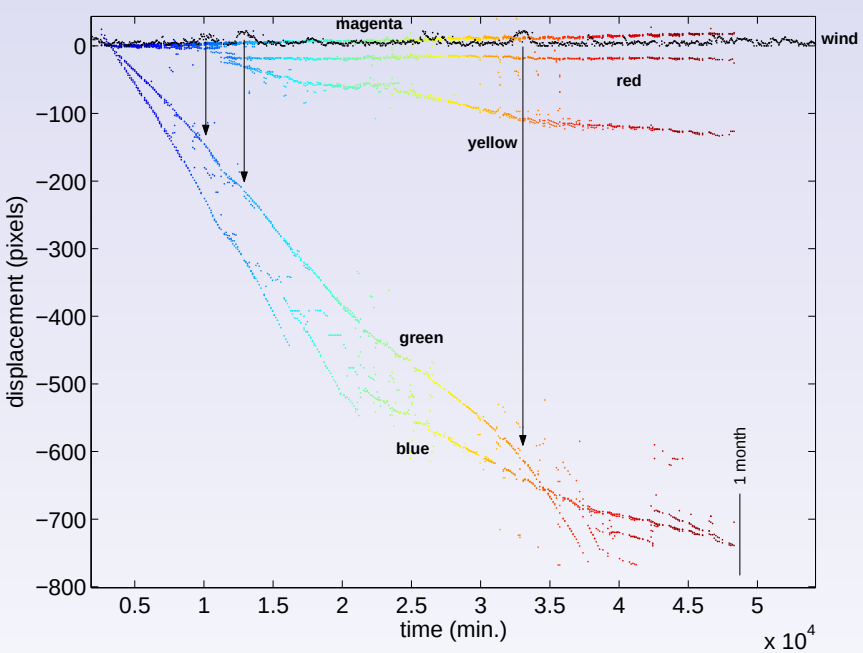


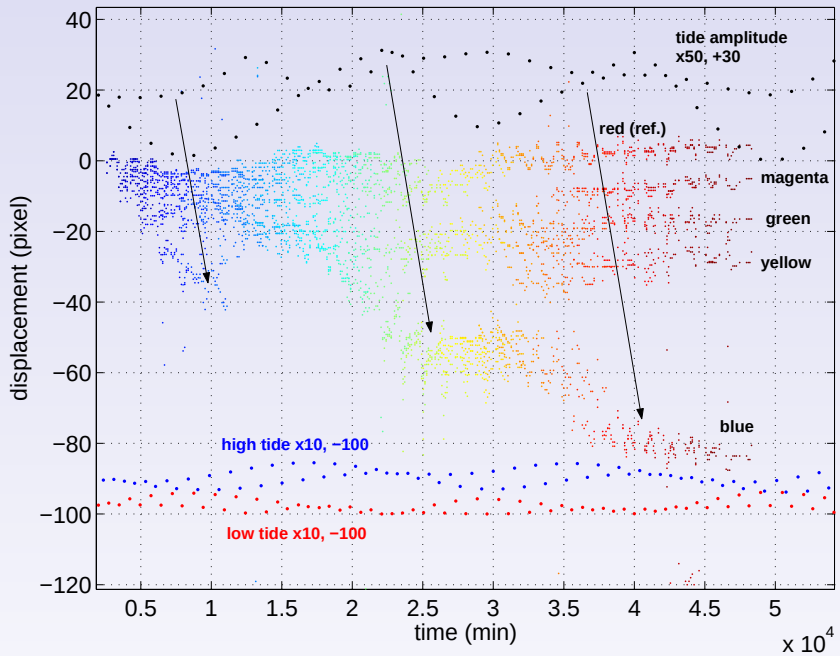


blue red green magenta black

Exemple

Déplacement moyen du flot d'icebergs, et oscillations verticales avec la marée [19]





19. Interface web à `xtide` à `http://tbone.biol.sc.edu/tide/sites_othernorth.html`

Exemple d'exploitation

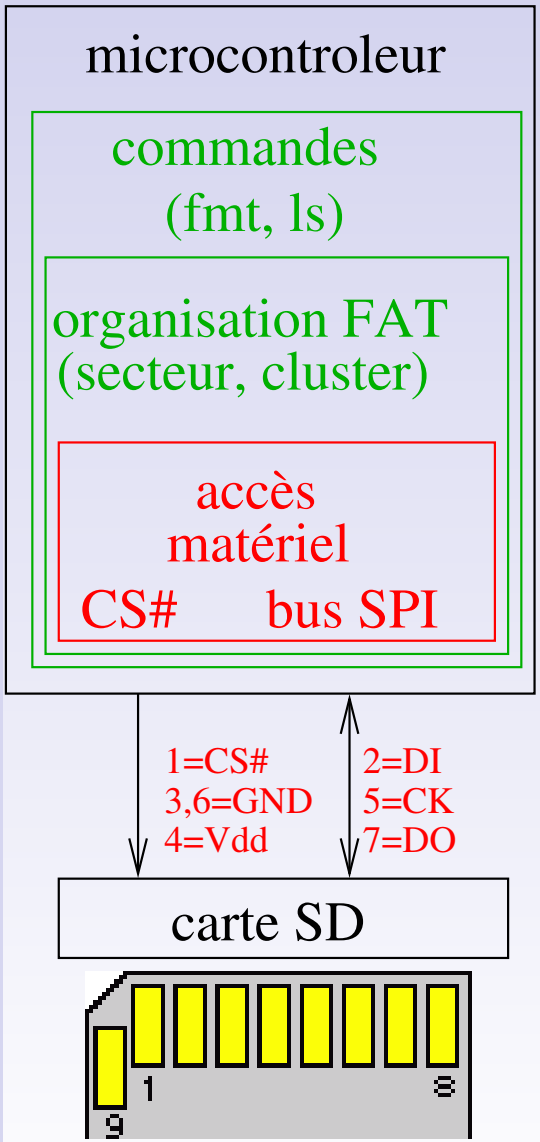
• D. Laffly, É. Bernard, M. Griselin, F. Tolle, J-M. Friedt, G. Martin & C. Marlin *High temporal resolution monitoring of snow cover using oblique view ground-based pictures*, Polar Record **48** (1), pp 11-16 (Jan. 2012)

• É. Bernard, J.-M Friedt, F. Tolle, M. Griselin, G. Martin, D. Laffly & C. Marlin, *Monitoring seasonal snow dynamics using ground based high resolution photography (Austre Lovénbreen, Svalbard, 79° N)*, ISPRS Journal of Photogrammetry and Remote Sensing (2012)

Portabilité du C : exemple du format FAT

- FAT (1980) : format de fichier utilisé par Microsoft sous DOS, resté un standard jusqu'à aujourd'hui sur tous les OS
- développé à l'époque des processeurs 8 et 16 bits $\Rightarrow$ ressources compatibles avec les microcontrôleurs actuels
- une fois un support de stockage de masse maîtrisé, permet de le rendre compatible avec un échange de données sur PC
- exemple concret : carte de stockage SD + format FAT
- deux approches : reprendre l'implémentation, ou exploiter une bibliothèque portable en C
- deux bibliothèques identifiées : EFSL (`http://sourceforge.net/projects/efsl/`) et FatFs (`http://elm-chan.org/fsw/ff/00index_e.html`)

Portabilité du C : FatFs



Modifier `Libraries/fat_sd/sd_spi_stm32.c` pour tenir compte du port SPI et CS#

```
#include "stm32f10x.h"
#include "platform_config.h"
#include "ff.h"
#include "term_io.h"  // xprintf
...
FATFS fs;
FIL f;
FRESULT res;
...
f_mount(0,&fs);
res=f_open(&f,"0:toto.txt",FA_READ | FA_OPEN_EXISTING);
if (res==0) {
  do {res=f_read(&f,buffer,1,&br);xprintf("%c",buffer[0]);}
  while (br>0);
  f_close(&f);
}
sprintf(buffer,"Hello World\r\n");
res=f_open(&f,"0:toto.txt",FA_WRITE | FA_OPEN_ALWAYS);
if (res==0) {f_lseek(&f,f.fsize);
             f_write(&f,buffer,13,&br);f_close(&f);
}
```

Test (STM32F103, SPI1, CS=PA4) :

```
Hello from a STM32 Demo-Application, M. Thomas
V1.2.1 7/2010
FatFs module test terminal
>fi
rc=0 FR_OK
>fl
----A 2010/10/31 17:57        17  TATA.TXT  tata.txt
----A 2010/10/31 20:30    246930  STM32_EF.TAR
----A 2010/10/31 20:25        19  TOTO.TXT  toto.txt
   3 File(s),     246966 bytes total
   0 Dir(s),  1019314176 bytes free
```

20

20. http://pinouts.ru/Memory/sdcard_pinout.shtml

Portabilité du C : EFSL

Occupation de RAM annoncée : 1500 octets
Exemple sur ARM7TDMI :

```
int main(void)
{ EmbeddedFile file;
  char chaine[42]="bonjour je suis une chaine de caracteres\r\n";
...
  if(efs_init(&efs,0)!=0){hang();}
  ls_openDir(&list ,&(efs.myFs ),"/" ) ;
  while(ls_getNext(&list)==0) list.currentEntry.FileName[11] = (char)0; jmf_prints(list.currentEntry.FileName);
  fs_umount(&efs.myFs);

  if(efs_init(&efs,0)!=0) jmf_prints("Could not open filesystem.\r\n");
  else
    {jmf_prints("Card init ok\r\n");
     if(file_fopen(&file,&efs.myFs,"TOTO.TXT",'r')!=0) jmf_prints("Could not open file.\r\n");
     else {jmf_prints("File found.\r\n");
           while ((e=file_read(&file,512,buf)))
              {for(l=0;l<e;l++)
               if (buf[l]=='\n') jmf_prints("\r\n"); else USART_SendData(USART1,buf[l]);
              }
           file_fclose(&file);
          }
    }
    jmf_prints("\r\n");
// il FAUT utiliser 'a' pour append, sinon erreur quand le fichier existe deja
// et qu'on utilise 'w'
    if (file_fopen(&file,&efs.myFs,"TOTO.TXT",'a')!=0) jmf_prints("Couldn't open file for appending\r\n");
    if (file_fopen(&file,&efs.myFs,"TOTO.TXT",'w')!=0) {jmf_prints("Couldn't open file for writing\r\n");
        return(-3);}

    file_write(&file,42,chaine);
    // 16 secondes pour 1000 et 10000 ecritures de 41 chars
    // 2'18" pour 100000*41 => 29 KB/s  (4.2 MB, no error)
    // 1'20" pour 50000*84 =>  52.5 KB/s (string length=84)
    file_fclose(&file);
    fs_umount(&efs.myFs);
    fin: goto fin;
```

Portabilité du C : EFSL

Sur microcontrôleur :

```
Welcome to minicom 2.3

OPTIONS: I18n
Compiled on Feb 24 2008, 16:35:15.
Port /dev/ttyUSB0

                Press CTRL-A Z for help on special keys

#
 Hello
ok
File list:
TATA    TXT
TOTO    TXT
finished browsing /
Card init ok
File found.
ceci est un test :
bonjour je suis une chaine de caracteres
bonjour je suis une chaine de caracteres
bonjour je suis une chaine de caracteres

Program completed, good bye
```

Sur PC :

```
eee:/home/jmfriedt/aduc/cortex/stm32_EFSL_demo# mount /dev/sdb /mnt/
eee:/home/jmfriedt/aduc/cortex/stm32_EFSL_demo# ls /mnt/
tata.txt  toto.txt
eee:/home/jmfriedt/aduc/cortex/stm32_EFSL_demo# cat /mnt/toto.txt
ceci est un test :
bonjour je suis une chaine de caracteres
bonjour je suis une chaine de caracteres
bonjour je suis une chaine de caracteres
bonjour je suis une chaine de caracteres
eee:/home/jmfriedt/aduc/cortex/stm32_EFSL_demo#
```

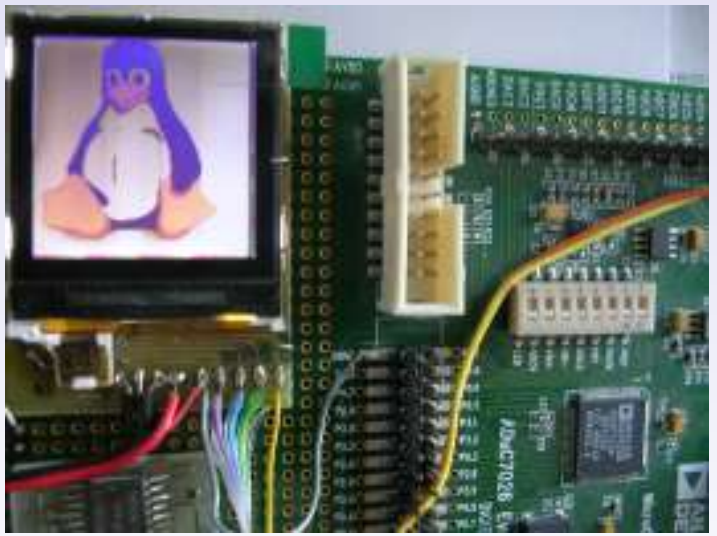
La crosscompilation

- Notion de cross-compilation et de toolchain : un compilateur pour une cible autre que le processeur sur lequel on travaille (généralement intel x86), ou un autre système d'exploitation (exemple : générer un exécutable windows sous GNU/Linux).
- Le trio `gcc`, `binutils` et `newlib`.
- Liste des cibles supportées par `gcc` (`http://www.gnu.org/software/gcc/backends.html`)

```
* Alpha
* ARM
* Atmel AVR
* Blackfin
* HC12
* H8/300
* IA-32 (x86)
* x86-64
* IA-64
* Motorola 68000
* MIPS
* PA-RISC
* PDP-11
* PowerPC
* R8C/M16C/M32C
* SPU
* System/390/zSeries
* SuperH
* SPARC
* VAX
```

Les outils pour crosscompiler

- Le trio `gcc`, `binutils` et `newlib`. Pour tous ces outils : `./configure --prefix=/usr/local/arm --target=arm-elf`
- Des outils sensés simplifier la tâche : buildroot.
- ARM7TDMI (le gameboy advance), MIPS (PSP, routeur WRT54G), CRIS (Fox), Coldfire (successeur au 68000), ARM9 (iMX de Freescale), FONera,
- Exemple d'écran graphique,
  masques pour fabriquer des couleurs sur 12 bits
  convertir PPM en ce format.



Exploitation de bibliothèques externes (`http://www.sparkfun.com/datasheets/LCD/Nokia_combined_LCD_driver_LPC2148.zip`)

newlib

Implémentation de `libc` répondant à toutes les exigences d'un OS
⇒ complet, mais complexe et lourd
Notion de *stubs* : colle entre `newlib` et nos programmes. Par exemple, envoyer `stdio` sur le port série[21].

```
int uart0_putc(int ch)
{
 while(!(0x020==(COMSTA0 & 0x020))) {}
 return (COMTX = ch);
}

_ssize_t _write_r (struct _reent *r,int file,const void *→
    ↪ptr, size_t len)
{ int i;
  const unsigned char *p;
  p = (const unsigned char*) ptr;
  for (i = 0; i < len; i++) {
      if (*p == '\n' ) uart0_putc('\r');
      uart0_putc(*p++);
  }
 return len;
}
```

21. http://wiki.osdev.org/Porting_Newlib

Exploitation d'une bibliothèque – `newlib`

`https://sourceware.org/newlib/libc.html#Stubs` : 17 stubs, colle entre `newlib` et nos programmes[22].

- **_exit** : quitter un programme sans fermer les fichiers ouverts
- environ :
- execve :
- fork :
- fstat :
- getpid :
- isatty :
- kill :
- link :
- lseek :
- open :
- **read** : lire depuis un fichier
- sbrk : *utilisé par* `malloc`
- stat :
- times :
- unlink :
- wait :
- **write** : écrire dans un fichier, incluant stdout

22. `http://wiki.osdev.org/Porting_Newlib`

newlib

L'utilisation de `printf` et du calcul flottant n'est pas à prendre à la légère :

Taille du fichier .hex intel :	avec stdio	thumb	80336
	sans stdio	thumb	4048
	avec stdio	arm	81641
	sans stdio	arm	4993

toujours en thumb :	sans stdio, avec une division flottante	12950
	sans stdio, avec atan sur flottant	17090
	avec stdio, avec une division flottante	80397
	avec stdio, avec atan sur flottant	80397

Thumb est un sous-ensemble d'instructions pour processeur ARM codé sur 16 bits au lieu de 32 bits [23]

23. J.-M Friedt, É. Carry, *Développement sur processeur à base de cœur ARM7 sous GNU/Linux*, GNU/Linux Magazine France **117** (Juin 2009), pp.40-59

Virgule fixe/virgule flottante

- Représentation d'un nombre en virgule flottante :
  - mantisse/exposant (mantisse $\in [0, 1-1[$), similaire à la notation scientifique $0,1234 \times 10^3$
  - supporte une large gamme de valeurs, au détriment de la précision (codage de la position de la virgule)
  - exposant 8 bits-mantisse 23 bits (float), 11-52 (double) qui code donc $2^{\pm 1024} \simeq 10^{\pm 308}$ sur 15 décimales ($2^{52} \simeq 4 \times 10^{15}$)
  - $\Rightarrow$ arithmétique complexe, addition passe par la dénormalisation pour avoir le même exposant (donc des 0 en début de mantisse du nombre le plus petit), multiplication aisée
  - implémentation matérielle sur processeurs haut de gamme (FPU)
- Représentation en virgule fixe : homothétie pour se ramener à des calculs sur des entiers.
- on note Qm.n pour représenter la partie entière sur m bits et la partie fractionnaire sur n bits [24] (notation en complément à deux, donc inclut un bit de signe)

24. Maxim, Application Note 3722 : Developing FFT Applications with Low-Power Microcontrollers (2006)

Opérations sur les flottants

- flottant = mantisse $\cdot 2^{exposant}$
- $\Rightarrow$ multiplication somme les exposants et multiplie les mantisses (problème de précision)
- $\Rightarrow$ sommer nécessite d'aligner les mantisses en ajustant les exposants
- en l'absence de FPU, opérations logicielles gourmandes en ressources

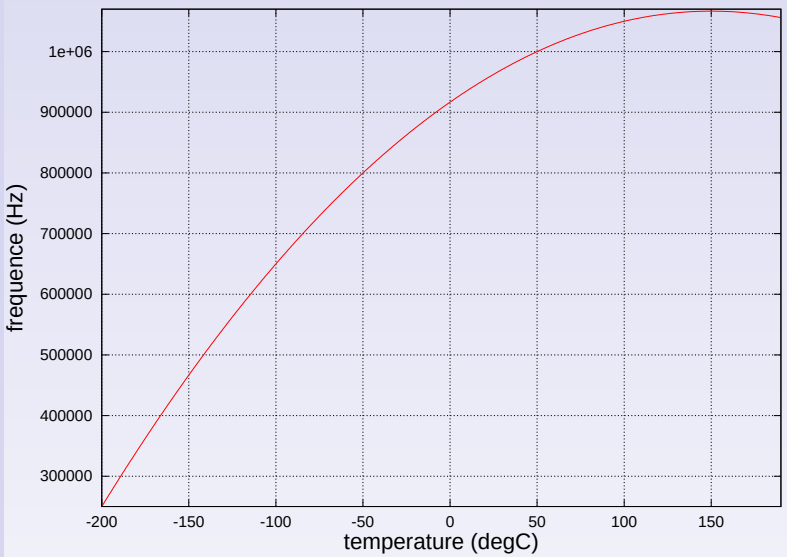
Exemple sur STM32F1 (pas de FPU)[25] :

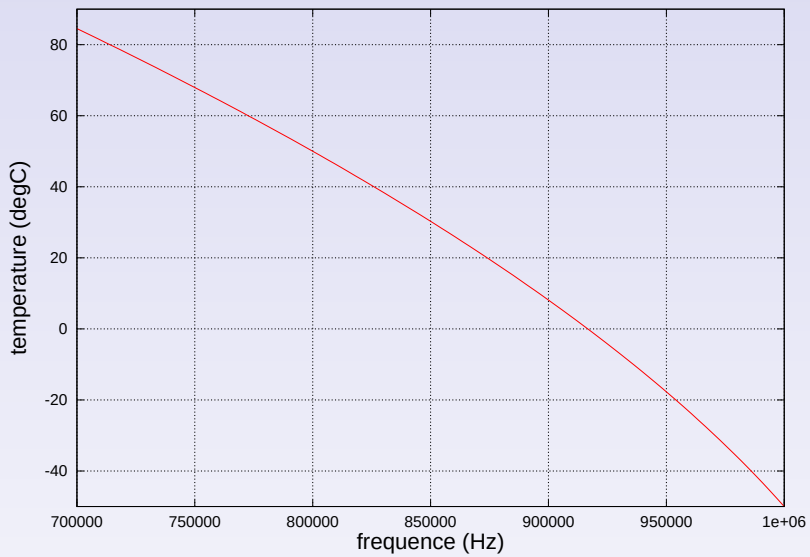
Opération	durée (μs)
ADC $\rightarrow$ entier	1,73
ADC $\rightarrow$ flottant	3,36
entier $\times 10^6/10^6$	1,08
entier $(<< 20)(>> 20)$	0,83
IIR à 2 coefficients flottants	30
IIR à 2 coefficients entiers	3

25. données acquises par G. Matten, LMA, FEMTO-ST

Exemple de capteur

- soit une mesure de fréquence f issue d'un capteur oscillant, dont la fréquence de vibration change avec la température T
- sachant que $f \simeq 1$ MHz avec une précision de 10 Hz, on veut obtenir la température à 0,1 K près :





$$f = -6,6667 \times T^2 + 2000 \times T + 916670$$

$$T = -150 + \sqrt{160000 - 0,15 \times f}$$

Exprimer cette loi d'étalonnage avec des coefficients entiers

Rappel : la suite $x_{n+1} = \frac{1}{2}\left(x_n + \frac{y}{x_n}\right)$ converge vers $\sqrt{y}$

(méthode de Newton sur $x^2 - y = 0$ avec $x_{n+1} = x_n - f/f'$)

Application des coefficients d'étalonnage

Exemple de calcul en virgule fixe compte tenu de la précision recherchée :

1. la loi liant fréquence et température est du type $f = \alpha T^2 + \beta T + \gamma \Rightarrow T = A \pm \sqrt{B + Cf}$
2. alors $(T-A)^2 = B + C \times f$ donc $2(T-A)dT - 2(T-A)dA = dB + dC \times f \Rightarrow |dT| = \left|\frac{dB}{2(T-A)}\right| + \left|\frac{dC \times f}{2(T-A)}\right| + |dA|$
3. on doit donc travailler avec chacun de ces termes inférieurs à 0,1.
4. Application numérique : par conception, $T - A \geq 100$ et $f \leq 10^6$ donc $dB \leq 10$, $dA \leq 0,1$ et $dC \leq 10^{-5}$
5. on va donc travailler sur $A \times 10$ et $C \times 10^5$ pour respecter la résolution recherchée
6. cependant, $C \simeq 0,1$ donc $C \times 10^5 \times f \simeq 10^{10}$ qui nécessite 34 bits,
7. mais f à 10 Hz donc on peut travailler sur $C \times 10^5 \times (f/10) \simeq 10^9$ qui ne nécessite que 30 bits

Conclusion : $10 \times T = 10 \times A + \left(\sqrt{B \times 10^4 + (10^5 C)(f/10)}\right)/10$

Au dela du C : un système d'exploitation

- ajout d'une couche d'abstraction supplémentaire (assembleur - C - noyau)
- les environnement exécutifs (TinyOS [26], RTEMS [27]) : développer comme sur un système d'exploitation, mais génère une application monolithique
- Pourquoi un système d'exploitation : un scheduler, un gestionnaire de mémoire (multitâche), une couche d'abstraction supposée cacher le matériel au programmeur, gestion des **systèmes de fichier** (> rawrite), **communication** (IP, TCP ...), console pour l'utilisateur.
- Par contre une contrainte additionnelle : maîtriser un nouvel ensemble de protocoles et méthodes de programmation ...
- ... afin de gérer le partage de ressources entre plusieurs programmes utilisateurs.

26. `www.tinyos.net`
27. `www.rtems.com/`

Généralités sur les systèmes d'exploitation ?

- Qu'est-ce que GNU/Linux : système d'exploitation, clone d'unix, compatible posix [12], **multiplateforme**.
- **Linux** est un noyau autour duquel fonctionnent des outils libres (**GNU**).
- Diverses bibliothèques C disponibles, avec différents impacts mémoire : glibc, uClibc, newlib ... et différentes fonctionnalités.
- Une distribution n'est qu'un emballage pour ces outils.
- uClinux pour les systèmes sans MMU [28], Linux pour les systèmes avec ; OpenWRT pour les routeurs.
- autres OS propriétaires : WinCE, LynxOS, QNX, vxWorks
- autres OS libres : NetBSD, eCOS (TinyOS, RTEMS)

28. *Memory Management Unit*, gestionnaire de mémoire chargé des contrôles d'accès et de la conversion de mémoire virtuelle en mémoire physique

Méthodes de développement

- Développement : le code est **validé sur PC** puis transféré sur **plateforme embarquée** (OS commun aux deux platformes).
- pour deux systèmes avec gestionnaire de mémoire, le code PC est directement exploitable
- en l'absence de gestionnaire de mémoire, il manque quelques fonctionnalités qu'il faut éviter (`fork`, lourdeur de `malloc`)
- NFS pour rapidement tester les nouvelles images.

```
mount -o nolock 192.168.1.2:/home /tmp/nfs
```

Environnement exécutif

- **Pas** de chargement dynamique de bibliothèque ou de programme : code statique
- concept de tâches ...
- ... et des problèmes associés (concurrence d'accès aux ressources, aux variables, priorité),
- solutions : mutex, sémaphores.
- Approprié pour le développement parallèle de tâches et réutilisation de code
- quelques "OS" libres temps réels : `http://www.freertos.org/`, `http://www.rtems.com/`, `nuttx.org/`

Exemple : FreeRTOS sur STM32

```
#include "FreeRTOS.h" //                                    6668 Aug 12 11:12 main.bin
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "croutine.h"

int global=0;
xSemaphoreHandle xMutex;

void task_rx(void* p)
{   char aff[10];
    char *t=(char*)p;
    int myInt = 0; volatile int local;
    for (myInt=0;myInt<8;myInt++)
    {xSemaphoreTake( xMutex, portMAX_DELAY );
     local=global;
     local++;
     uart_puts(t);
     global=local;

     xSemaphoreGive( xMutex );
     vTaskDelay( ( rand() & 0x1 ) );
    }
    aff[0]=' ';aff[1]=global+'0';aff[2]=' ';aff[3]=0;uart_puts(aff);
    while (1) vTaskDelay( ( rand() & 0x5 ) );
}

int main()
{ Led_Init();
  Usart1_Init();
  srand( 567 );
  xMutex = xSemaphoreCreateMutex();
  xTaskCreate(task_rx, (signed char*)"t1", STACK_BYTES(2048), "1111111\r\n\0", 1, 0);
  xTaskCreate(task_rx, (signed char*)"t2", STACK_BYTES(2048), "2222222\r\n\0", 1, 0);
  vTaskStartScheduler();
  while(1);
  return 0;}
```

Exemple : RTEMS sur STM32

```c
#include <rtems/test.h>
#include <bsp.h> /* for device driver prototypes */
#include <stdio.h>
#include <stdlib.h>

const char rtems_test_name[] = "HELLO WORLD";

rtems_task Init(
  rtems_task_argument ignored
)
{
  rtems_test_begin();
  printf( "Hello World\n" );
  rtems_test_end();
  exit( 0 );
}

#define CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER

#define CONFIGURE_MAXIMUM_TASKS             1
#define CONFIGURE_USE_DEVFS_AS_BASE_FILESYSTEM

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION

#define CONFIGURE_INIT
#include <rtems/confdefs.h>
```

70536 arm-rtems4.11/c/stm32f105rc/testsuites/samples/hello/hello.bin [29]

29. http ://wiki.rtems.com/wiki/index.php/STM32F105

Exemple : RTEMS sur NDS

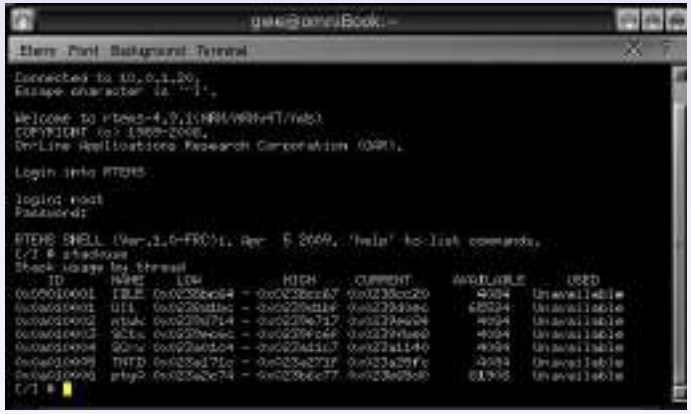
- couche d'émulation POSIX de RTEMS : un environnement de travail proche de celui connu sous Unix
- abstraction des accès au matériel : framebuffer, réseau (wifi)
- pile TCP/IP pour la liaison haut débit
- exploitation efficace des ressources (4 MB RAM)

BSP RTEMS porté à NDS par M. Bucchianeri, B. Ratier, R. Voltz et C. Gestes

`http://www.rtems.com/ftp/pub/rtems/current_contrib/nds-bsp/manual.html`

Exemple : RTEMS sur NDS

- couche d'émulation POSIX de RTEMS : un environnement de travail proche de celui connu sous Unix
- abstraction des accès au matériel : framebuffer, réseau (wifi)
- pile TCP/IP pour la liaison haut débit
- exploitation efficace des ressources (4 MB RAM)



BSP RTEMS porté à NDS par M. Bucchianeri, B. Ratier, R. Voltz et C. Gestes

`http://www.rtems.com/ftp/pub/rtems/current_contrib/nds-bsp/manual.html`

Exemple RTEMS : framebuffer

Une application RTEMS doit déclarer les ressource qu'elle exploitera

```
#include <bsp.h>
#include <rtems/fb.h>
#include <rtems/console.h>
#include <rtems/clockdrv.h>
#include "fb.h"

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <rtems/mw_fb.h>

static struct fb_screeninfo fb_info;

inline void draw_pixel(int x, int y, int color)
{
  uint16_t* loc = fb_info.smem_start;
  loc += y * fb_info.xres + x;

  *loc = color;    // 5R, 5G, 5B
  *loc |= 1 << 15; // transparence ?
}

void draw_ppm()
{int x,y,bpp;char r,g,b;
 int l=0;
 for (y=0;y<161;y++)
   for (x=0;x<296;x++) {
     r=image[l++];g=image[l++];b=image[l++];
     bpp=(((int)(r&0xf8))>>3)+(((int)(g&0xf8))<<2)+ \
         (((int)(b&0xf8))<<8);
     if (x<256) draw_pixel(x, y, bpp);
   }
}
```

```
rtems_task Init(rtems_task_argument ignored)
{
  struct fb_exec_function exec;
  int fd = open("/dev/fb0", O_RDWR);

  if (fd < 0) {printk("failed\n");exit(0);}

  exec.func_no = FB_FUNC_ENTER_GRAPHICS;
  ioctl(fd, FB_EXEC_FUNCTION, (void*)&exec);
  ioctl(fd, FB_SCREENINFO, (void*)&fb_info);

  draw_ppm();
  while (1) ;

  exec.func_no = FB_FUNC_EXIT_GRAPHICS;
  ioctl(fd, FB_EXEC_FUNCTION, (void*)&exec);
  close(fd);printk("done.\n");exit(0);
}

/* configuration information */
#define CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE

rtems_driver_address_table Device_drivers[] =
{ CONSOLE_DRIVER_TABLE_ENTRY,
  CLOCK_DRIVER_TABLE_ENTRY,
  FB_DRIVER_TABLE_ENTRY,
  { NULL,NULL, NULL,NULL,NULL, NULL }
};

#define CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS      10
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_MAXIMUM_TASKS 1
#define CONFIGURE_INIT
#include <rtems/confdefs.h>
```

Exemple RTEMS : GPIO

```
#include <bsp.h>
#include <stdlib.h>
#include <stdio.h>
#include <nds/memory.h>

rtems_id timer_id;
uint16_t l=0;

void callback()
{ printk("Callback %x\n",l);
  (*(volatile uint16_t*)0x08000000)=l;
  l=0xffff-l;
  rtems_timer_fire_after(timer_id, 100, callback, NULL);
}

rtems_task Init(rtems_task_argument ignored)
{ rtems_status_code status;
  rtems_name timer_name = rtems_build_name('C','P','U','T');

  printk( "\n\n*** HELLO WORLD TEST ***\n" );
// sysSetCartOwner(BUS_OWNER_ARM9);
  (*(vuint16*)0x04000204) = ((*(vuint16*)0x04000204) \
         & ~ARM7_OWNS_ROM);

  status = rtems_timer_create(timer_name,&timer_id);
  rtems_timer_fire_after(timer_id, 1, callback, NULL);
  rtems_stack_checker_report_usage();
           // requires #define CONFIGURE_INIT

  printk( "*** END OF HELLO WORLD TEST ***\n" );
  while(1)
    ;
  exit( 0 );
}

/* configuration information */
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

/* configuration information */
#define CONFIGURE_MAXIMUM_DEVICES                     40
#define CONFIGURE_MAXIMUM_TASKS                       100
#define CONFIGURE_MAXIMUM_TIMERS                      32
#define CONFIGURE_MAXIMUM_SEMAPHORES                  100
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES              20
#define CONFIGURE_MAXIMUM_PARTITIONS                  100
#define CONFIGURE_MAXIMUM_REGIONS                     100
/* This seetings overwrite the ones defined in confdefs.h */
#define CONFIGURE_MAXIMUM_POSIX_MUTEXES               32
#define CONFIGURE_MAXIMUM_POSIX_CONDITION_VARIABLES   32
#define CONFIGURE_MAXIMUM_POSIX_KEYS                  32
#define CONFIGURE_MAXIMUM_POSIX_QUEUED_SIGNALS        10
#define CONFIGURE_MAXIMUM_POSIX_THREADS               128
#define CONFIGURE_MAXIMUM_POSIX_TIMERS                10
#define CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS      200

#define STACK_CHECKER_ON
#define CONFIGURE_INIT

#include <rtems/confdefs.h>
```

J.-M Friedt, G. Goavec-Merou, *Interfaces matérielles et OS libres pour Nintendo DS : DSLinux et RTEMS*, GNU/Linux Magazine France Hors Série 43, août 2009

Organisation de la mémoire

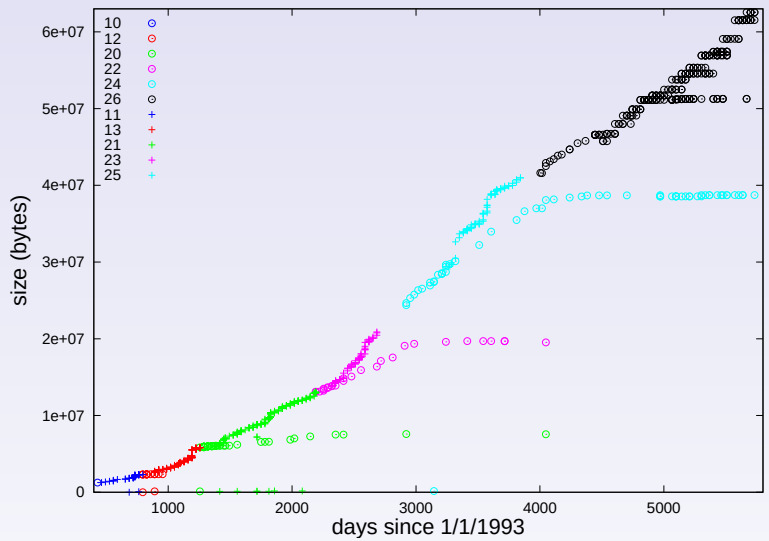
- Le *linker* doit savoir où placer quelle partie du code (programme en région non-volatie, pile en RAM, ...)
- le linker script (option -T de gcc) fournit ces informations
- un nouveau script pour chaque plateforme et chaque méthode d'utilisation (debuggage avec programme en RAM, ou programme en flash)

```
MEMORY
{ /* IntFLASH (rx) : ORIGIN = 0x00080000, LENGTH = 63488 */
  IntRAM   (rw) : ORIGIN = 0x00010000, LENGTH = 8192
}
SECTIONS
{
  /* first section is .text which is used for code */
  .text :
  { *startup.o (.text)          /* Startup code */
    *(.text)                    /* remaining code */
    *(.glue_7t) *(.glue_7)
  } >IntRAM
  . = ALIGN(4);

  /* .rodata section is used for read-only data (csts) */
  .rodata :
  { *(.rodata)
  } >IntRAM

  . = ALIGN(4);

  _etext = . ;
  PROVIDE (etext = .);
```

```
  /* .data section which is used for initialized data */
  .data : /* AT (_etext) */
  { _data = . ;
    *(.data)
    SORT(CONSTRUCTORS)
  } >IntRAM
  . = ALIGN(4);

  /* .bss section which is used for uninitialized data */
  .bss :
  { __bss_start = . ;
    __bss_start__ = . ;
    *(.bss)
    *(COMMON)
    } >IntRAM
  . = ALIGN(4);
  __bss_end__ = . ;
  __bss_end__ = . ;

  _end = .;
  PROVIDE (end = .);
[...]
```

Porter GNU/Linux à une nouvelle architecture

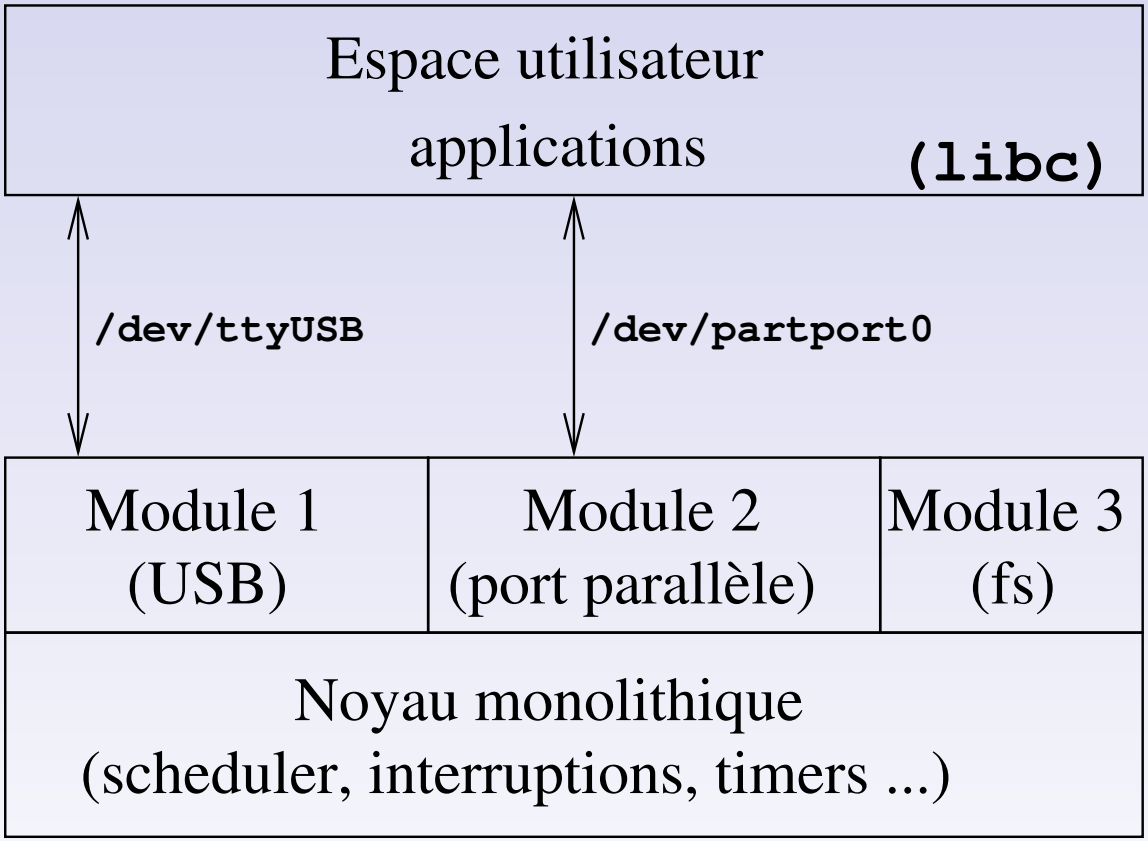
- une chaîne de compilation
- tenir compte de l'architecture mémoire de la cible (option -T de `ld`)
- développer un bootloader (initialisation des périphériques et chargement du noyau en mémoire) [30]
- développer les modules noyau pour supporter le matériel spécifique à la nouvelle architecture [10].
- impact mémoire ?



Évolution de la taille de l'archive `.tar.gz` du noyau linux (`www.kernel.org`)

30. S. Guinot, J.-M Friedt, *GNU/Linux sur Playstation Portable*, GNU/Linux Magazine France **114**, Mars 2009, pp.30-40

Architecture d'un OS



fonctions système, réseau, gestion processus et mémoire, vfs

Le répertoire /dev

- contient des pseudo-fichiers faisant le lien entre un module noyau et l'espace utilisateur
- chaque *device* est défini par un *major number* et, dans sa classe, un *minor number*
- du côté du noyau, un *driver* s'est lié au même major number et fournit des méthodes standard
- `open()`
- `close()`
- `write()`
- `read()`
- `ioctl()`

```
static struct file_operations qadc_fops =
{
        owner:          THIS_MODULE,
        read:           qadc_read,
        open:           qadc_open,
        ioctl:          qadc_ioctl,
        release:        qadc_release,
};

static int __init qadc_init (void)
{...
 register_chrdev (qadc_major, "ppp", &qadc_fops);
 ...}

module_init (qadc_init);
module_exit (qadc_exit);
```

Module noyau : gestion des interruptions

Seul un module noyau peut accéder aux interruptions du processeur (exemple : interruption 7 est associée à la broche 10 du port parallèle).

```
jmfriedt@(none):~$ cat /proc/interrupts | grep parp
  7:          0    IO-APIC-edge  parport0
```

qui se gère par [31]

```
#define PARALLEL_PORT_INTERRUPT 7
#define BASEPORT                0x378
int sortir=0;

static int interrupt_handler(void)
  {printk(">>> PARALLEL PORT INT\n");sortir=1;}

static int __init jmfpport_init_module (void) {
  ret = request_irq(PARALLEL_PORT_INTERRUPT, &interrupt_handler,
SA_INTERRUPT, "parallelport", NULL);
  enable_irq(7);
  outb_p(0x10, BASEPORT + 2);
}

static ssize_t skeleton_read (struct file *file, char *buf,
size_t count, loff_t *ppos) {
 ...
 do {} while (sortir==0);   // interruptible_sleep_on(&skeleton_wait);
 err = copy_to_user(buf,string,counter);
 ...
}
```

31. www.captain.at/howto-linux-device-driver-template-skeleton.php

Les signaux : prévenir d'un évènement

Équivalent logiciel : les signaux

```
#include <signal.h>

void my_handler (int sig) {
    printf ("I got SIGINT, number %d\n", sig);}

int main ( void ) {
    signal (SIGINT, my_handler);
    while (1) {}
}
```

qui donne chaque fois qu'on appuie sur CTRL-C (tuer par `kill -9 PID`)

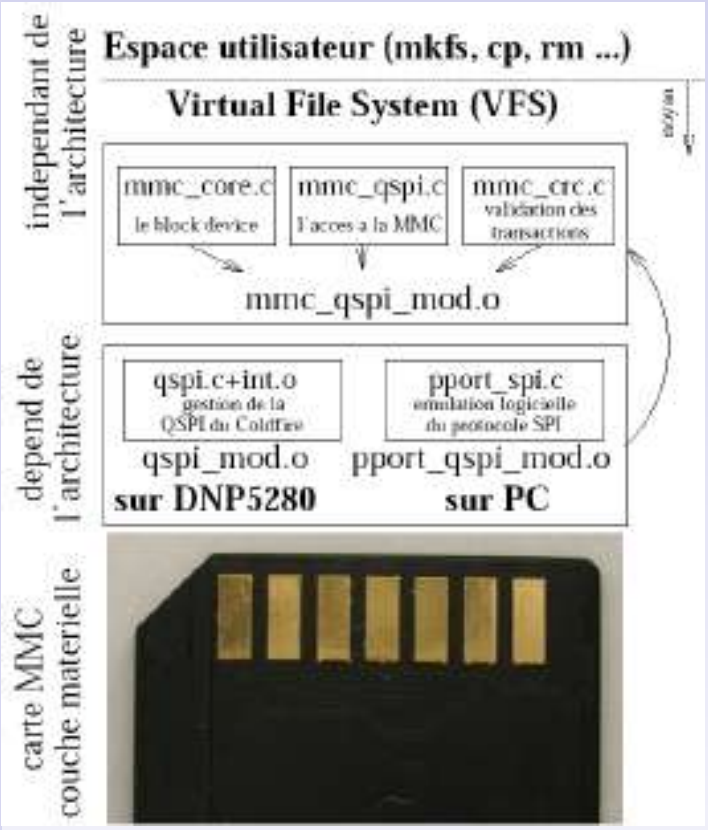
```
jmfriedt@(none):~$ ./sigint
I got SIGINT, number 2
I got SIGINT, number 2
```

Les systèmes de fichiers

- un *block device*[32] décrit un périphérique matériel
- le Virtual File System fournit l'interface entre les systèmes de fichiers (formattage) et le block device
- divers formats de fichiers existent selon les besoins (journalistion, temps d'accès, cache, compression, cryptage ...)
- pour l'embarqué : contraintes de place (compression) et de matériel (ne pas re-écrire trop de fois au même endroit)
- écriture sur mémoire flash dont la durée de vie est limitée. Des systèmes de fichiers utilisés sur disque dur, surtout journalisés et swap, ne sont pas appropriés : `jffs2` dédié à ces opérations. Compression des données pour les systèmes en flash en lecture uniquement : `cramfs`.

32. P. Ficheux, *Programmation noyau sous Linux : les pilotes en mode bloc*, GNU/Linux Magazine France, 109, pp.4-10 (Octobre 2008) ou `http://www.linuxjournal.com/article/2890`

Block device/VFS [33]



- passer par un block device au lieu de char `brw-rw---- 1 root disk 3, 0 Oct 8 18:43 /dev/hda`
- remplacer `file_operation` par `block_operation`
- compléter les appels du char. device par les appels du VFS : `do_request` et `make_request` gèrent écritures et lectures
- les transferts sont régis selon une "géométrie" de disque définie dans une structure `gendisk`
- liste de requêtes :

```
floppy_queue = blk_init_queue(do_fd_request, &floppy_lock);
blk_queue_make_request(lo->lo_queue, loop_make_request);
```

33. S. Guinot, J.-M Friedt, *Stockage de masse non-volatile : un block device pour MultiMediaCard*, GNU/Linux Magazine France, Hors Série 25 (Avril 2006)

La communication sur ethernet

- la couche physique : ethernet (gestion de collisions, paquets)
- IP
- ICMP, UDP, TCP, ARP [11]
- notions de client-serveur et sockets ⇒ `fork()`

Modèle OSI de la hiérarchie d'un réseau

	couche/fonction	exemple
7	application	(DNS, FTP, NTP ...)
6	représentation des données	(ASCII, EBDIC, ...)
5	session	(socket)
4	transport	(TCP, UDP)
3	paquets réseau	(IP)
2	accès (MAC)	(AppleTalk, 802.3 "ethernet")
1	physique	(RS232, 10BaseT, 802.11 "Wifi")

Tout ordinateur supportant IP a son adresse locale : 127.0.0.1 (localhost)

Configuration réseau sous Linux

1. les *interfaces* : `ifconfig eth0 192.168.1.1`

```
jmfriedt@dhcp-221:~/$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:48:54:55:09:6D
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::248:54ff:fe55:96d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1203876 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1616275 errors:0 dropped:0 overruns:1 carrier:0
          collisions:397422 txqueuelen:1000
          RX bytes:666243681 (635.3 MiB)  TX bytes:1888543246 (1.7 GiB)
          Interrupt:10 Base address:0xe800

eth1      Link encap:Ethernet  HWaddr 00:48:54:39:44:7A
          inet addr:172.16.120.21  Bcast:172.16.120.255  Mask:255.255.255.0
          inet6 addr: fe80::248:54ff:fe39:447a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5818227 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3024232 errors:0 dropped:0 overruns:1 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3049792520 (2.8 GiB)  TX bytes:1779165735 (1.6 GiB)
          Interrupt:11 Base address:0xec00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:340887 errors:0 dropped:0 overruns:0 frame:0
          TX packets:340887 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:44641584 (42.5 MiB)  TX bytes:44641584 (42.5 MiB)
```

2. la table de routage :

```
jmfriedt@dhcp-221:~/eagle$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
172.16.120.0    *               255.255.255.0   U     0      0        0 eth1
default         172.16.120.254  0.0.0.0         UG    0      0        0 eth1
```

Quelques définitions

- IP : identification de l'ordinateur. Couplé à ARP, chaque nom+domaine est associé à une adresse unique. ICMP : gestion du réseau.
- UDP : mode non-connecté. Les paquets sont émis sur le réseau sans garantie de latence ou de réception.
- TCP : mode connecté. La réception des paquets et l'ordre sont garantis. Gourmand en ressources [34]
- les services, le protocole utilisé et le port de connexion sont standardisés : `/etc/services`
- sous dimensionner le matériel rend vulnérable aux attaques DoS.

34. E. Gergori, *Small Footprint ColdFire TCP/IP Stack*, Freescale AN5307 : 15 kB RAM et 35 kB flash

Exemple de client-serveur UDP

Exemple de communication UDP [35] :

Un serveur est à l'écoute d'une connection

```
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>

#define BUFSIZE         1024

void alltoupper(char* s)
{while ( *s != 0 ) *s++ = toupper(*s);}

int main()
{ char buffer[BUFSIZE];
  struct sockaddr_in addr;
  int sd, addr_size, bytes_read;

  sd = socket(PF_INET, SOCK_DGRAM, 0);
  addr.sin_family = AF_INET;
  addr.sin_port = htons(9999);
  addr.sin_addr.s_addr = INADDR_ANY;
  bind(sd, (struct sockaddr*)&addr, sizeof(addr));
  do {bzero(buffer, BUFSIZE);addr_size = BUFSIZE;
      bytes_read=recvfrom(sd,buffer,BUFSIZE,0, \
         (struct sockaddr*)&addr,&addr_size);
      printf("Connect: %s:%d %s\n",inet_ntoa(addr.sin_addr),\
         ntohs(addr.sin_port), buffer);
      alltoupper(buffer);
      sendto(sd,buffer,bytes_read,0,(struct sockaddr*)&addr, \
         addr_size);
  } while ( bytes_read > 0 );
  close(sd);return 0;
}
```

Un client se connecte au serveur pour amorcer une transaction

```
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>

#define BUFSIZE         1024

int main(int argc, char **argv)
{     char buffer[BUFSIZE];
      struct sockaddr_in addr;
      int sd, addr_size;

      if ( argc != 2 )
         {printf("usage: %s <msg>\n", argv[0]);exit(0);}
      sd = socket(PF_INET, SOCK_DGRAM, 0);
      addr.sin_family = AF_INET;
      addr.sin_port = htons(9999);
      inet_aton("127.0.0.1", &addr.sin_addr);
      sendto(sd, argv[1], strlen(argv[1])+1, 0, \
         (struct sockaddr*)&addr, sizeof(addr));
      bzero(buffer, BUFSIZE);
      addr_size = sizeof(addr);
      recvfrom(sd,buffer,BUFSIZE,0,(struct sockaddr*)&addr,\
        &addr_size);
      printf("Reply: %s:%d %s\n",inet_ntoa(addr.sin_addr), \
        ntohs(addr.sin_port), buffer);
      close(sd);
      return 0;
}
```

35. http://www.cs.utah.edu/~swalton/listings/sockets/programs/

Exemple de serveur TCP

```
#include <sys/socket.h>
#include <resolv.h>
#include <strings.h>
#include <arpa/inet.h>

#define MY_PORT         9999
#define MAXBUF          1024

int main()
{int sockfd;
 struct sockaddr_in self;
 char buffer[MAXBUF];

 sockfd = socket(AF_INET, SOCK_STREAM, 0);  // ICI LE TYPE DE SOCKET

 bzero(&self, sizeof(self));
 self.sin_family = AF_INET;
 self.sin_port = htons(MY_PORT);
 self.sin_addr.s_addr = INADDR_ANY;

 bind(sockfd, (struct sockaddr*)&self, sizeof(self));
 listen(sockfd, 20);

 while (1)
 {struct sockaddr_in client_addr;
  int clientfd,addrlen=sizeof(client_addr);

  clientfd = accept(sockfd, (struct sockaddr*)&client_addr, &addrlen);
  printf("%s:%d connected\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
  send(clientfd, buffer, recv(clientfd, buffer, MAXBUF, 0), 0);

  close(clientfd);
 }
 close(sockfd);return(0);  // Clean up (should never get here)
}
```

La façon la plus simple de tester un serveur : `telnet IP port`

Applications

Une bonne compréhension du réseau est nécessaire pour les applications

- de sécurité : exemple de firewall ou routeurs dédiés. Filtrage des paquets, gestion en un temps minimum
- d'instrumentation contrôlée à distance. Même sans se soucier des sécurité des données, il faut savoir éviter des attaques de type DoS.
- l'ensemble des couches n'est souvent pas nécessaire (souhaitable) pour une application embarquée : maîtriser l'ensemble de la chaîne de communication pour n'en garder que les éléments utiles (p.ex raw-IP).

Dans nos exemples, l'argument de `socket` définit le protocole de transport (`SOCK_DGRAM` (UDP), `SOCK_STREAM` (TCP)).

`SOCK_RAW` $\Rightarrow$ entête de 20 octets contenant IP source et destination, suivi de la charge à communiquer (routage)

```
45 00 00 34 19 a9 00 00 3c ff 66 20 7f 00 00 01    <- source = 127.0.0.1
7f 00 00 01 30 30 30 30 30 30 30 30 30 30 30 30    <- dest = 127.0.0.1
```

Pour système embarqué léger : raw-ethernet (Malå ProEx), pas de routage

Thread/fork

La gestion des processus :

- le changement de contexte est une opération lourde [36]
- `thread` : occupe les mêmes ressources mémoire que le père mais contient sa propre pile ⇒ partage de ressources mais élimine les problèmes de communication entre processus
- `fork` : dupliquer un processus en héritant de tous les attributs du père ⇒ possède sa propre mémoire virtuelle ⇒ pas de problème de propriété des données
- pour les systèmes sans MMU, `vfork`

36. `https://computing.llnl.gov/tutorials/pthreads/`

Les threads

Au delà de la simple connexion point à point : un serveur doit gérer une multitude de clients.

- implémentation des `threads` sous Linux : la bibliothèque `pthreads` ⇒ compiler avec `-lpthread`
- Multithreading pour les applications clients/server. Mutex pour éviter les race conditions
- Méthode pour diviser un processus en sous-tâches indépendantes (client-serveur ou applications graphiques).
- Gestion des variables partagées entre threads : les `mutex`[37]

37. `http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html`

Exemple de thread

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

#define NTHREADS 10

void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;

main()
{int d[10];
 pthread_t thread_id[NTHREADS];
 int i, j;

 for (i=0; i < NTHREADS; i++)
     {d[i]=i;
      pthread_create( &thread_id[i], NULL, thread_function, &d[i] );} // CREATION THREADS
 for(j=0; j < NTHREADS; j++) {pthread_join( thread_id[j], NULL);}     // ATTENTE DE LEUR MORT
 printf("Final counter value: %d\n", counter);
}

void *thread_function(void *d)
{printf("Thread number %d: %lx\n", *(int *)d, pthread_self());
 pthread_mutex_lock( &mutex1 );
 usleep(500000); // 500 ms
 counter++;
 pthread_mutex_unlock( &mutex1 );
}
```

Que se passe-t-il si on *sort* le `usleep()` du mutex ?

```
$ time ./thread                    $ time ./thread
...                                ...
real    0m5.003s                   real    0m0.501s
user    0m0.004s                   user    0m0.000s
sys     0m0.000s                   sys     0m0.000s
```

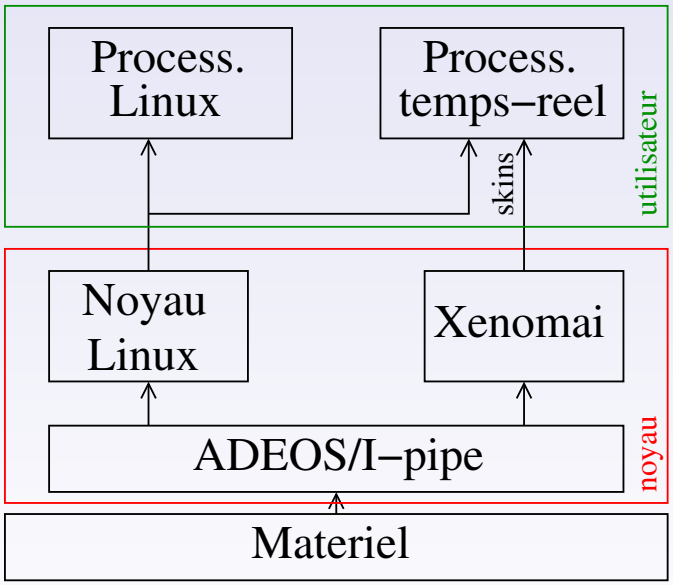
L'OS n'est pas toujours bien

- un OS doit booter : prend du temps et donc de l'énergie
- un OS nécessite de maîtriser des API
- un OS nécessite de la mémoire et de la puissance de calcul

$\Rightarrow$ bien peser les avantages (bibliothèques, contributions externes, stabilité des outils) et les contres avant de faire un choix.

Le temps réel : concepts [39]

- Notion de temps réel : plus on ajoute de couches d'abstractions, plus on augmente la latence.
- Le temps réel garantit une latence bornée.
- Extension temps réel de Linux : xenomai [38].
- Les interruptions sont source de latence (on fait autre chose que la tâche prioritaire en cours) ⇒ processus atomique qui ne peut être interrompu (désactiver les interruptions matérielles en entrée et réactiver en sortie ... risque de pertes de données).



38. `http://www.xenomai.org/index.php/Main_Page`
39. P. Ficheux, É. Bénard, *Linux embarqué – 4. Ed.*, Eyrolles (2012)

Le temps réel : exemple

Système "classique"

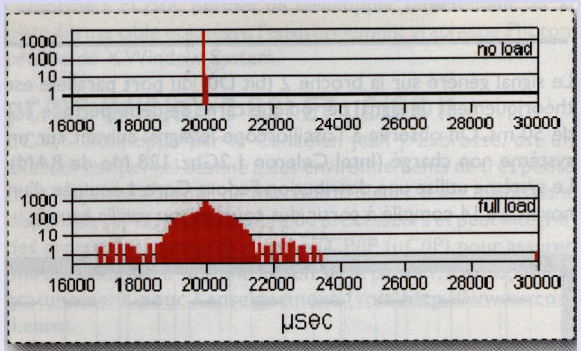
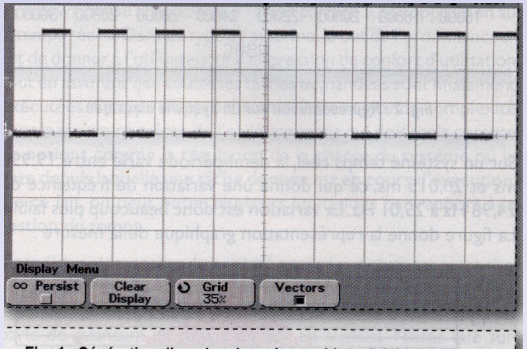


Fig. 2 : Représentation sur un système classique



Fig; 4 : Génération d'un signal carré sous Linux 2.6.14 non chargé

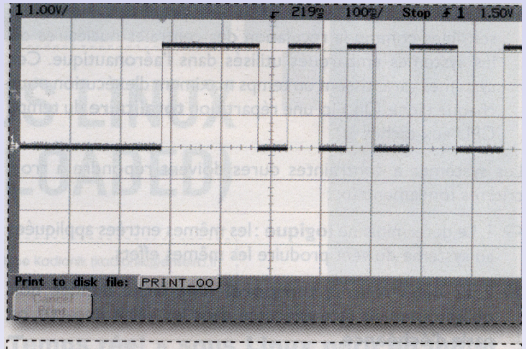


Fig. 5 : Génération d'un signal carré sous Linux 2.6.14 chargé

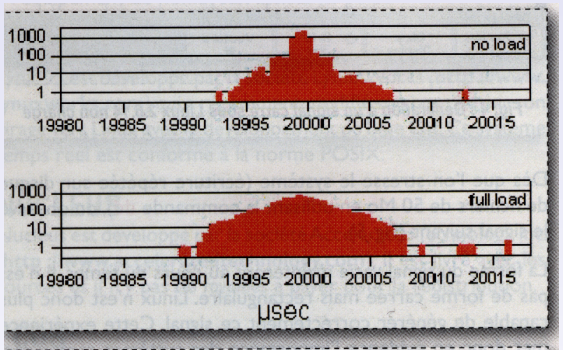


Fig. 3 : Représentation sur un système temps réel

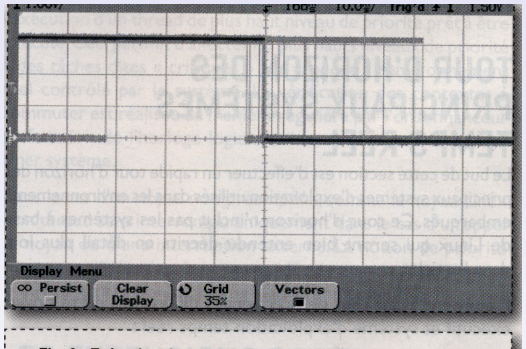


Fig. 6 : Exécution de square sur un noyau 2.6 préemptif stressé

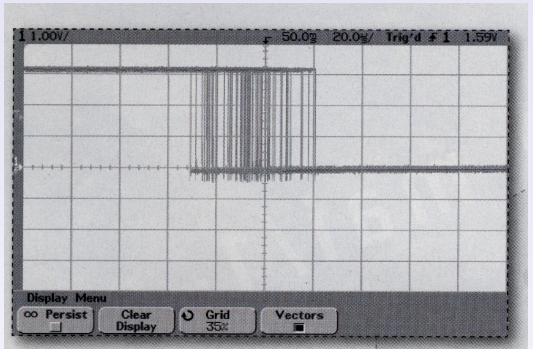


Fig. 10 : Exécution de xenomai_square_module en mode noyau

Extension `xenomai` temps-réel de Linux[40]

40. Toutes le figures sont extraites de P. Ficheux &. P. Kadionik, *Temps réel sous Linux (reloaded)*, GNU/Linux Magazine France, Hors Série 24 (Février-Mars 2006), pp.72-81

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [41] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
int status=0;
void test(int signum) {
    sunxi_gpio_output(PG9,status);
    status^=0xff;
}

int main()
{int i,status=0;
 struct sigaction sa;
 struct itimerval timer;

 sunxi_gpio_init();
 sunxi_gpio_set_cfgpin(PG9,OUTPUT);

 memset(&sa, 0,sizeof(sa));
 sa.sa_handler = &test;
 sigaction(SIGVTALRM,&sa, NULL);
 timer.it_value.tv_sec = 0;
 timer.it_value.tv_usec = TIMESLEEP;
 timer.it_interval.tv_sec = 0;
 timer.it_interval.tv_usec = TIMESLEEP;
 setitimer(ITIMER_VIRTUAL, &timer, NULL);
 while(1);
 sunxi_gpio_cleanup();
 return 0;}
```

En charge

Sans charge

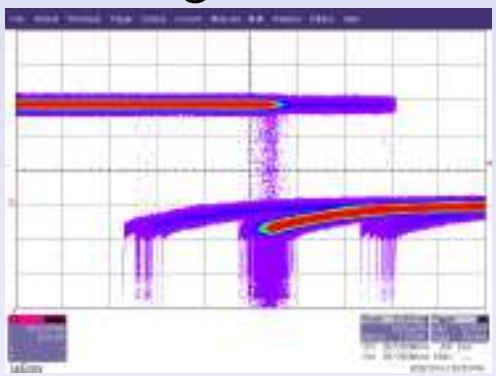
41. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [42] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
int main()
{int i,status=0;

 sunxi_gpio_init();
 sunxi_gpio_set_cfgpin(PG9,OUTPUT);

 while (1) {
    sunxi_gpio_output(PG9,status); usleep(500);
    status^=0xff;
 }
 sunxi_gpio_cleanup();
 return 0;
}
```

En charge



Sans charge

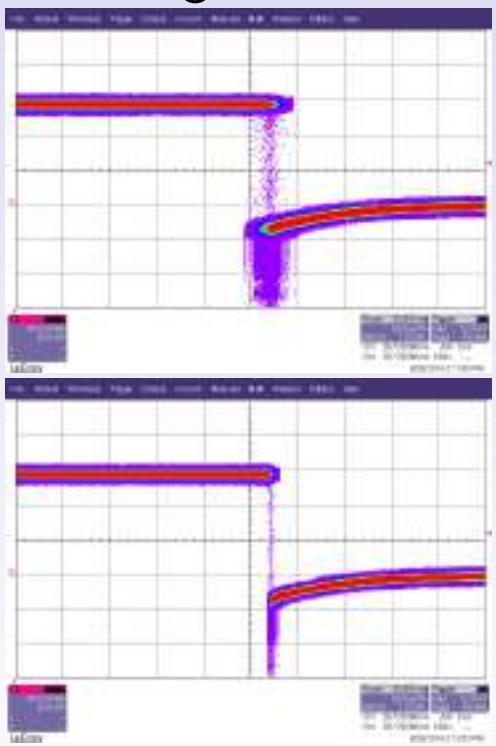
42. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [43] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
RT_TASK blink_task;

void blink(void *arg){
 int status=0;
 struct timespec tim = {0,TIMESLEEP};

 while(1)
   {sunxi_gpio_output(PG9,status);
    status^=0xff;
    nanosleep(&tim,NULL);
   }
}

int main(int argc, char **argv) {
 sunxi_gpio_init();
 sunxi_gpio_set_cfgpin(PG9,OUTPUT);
 /* Avoid memory swapping for this program */
 mlockall(MCL_CURRENT|MCL_FUTURE);
 rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
 rt_task_start(&blink_task, &blink, NULL);
 pause();
 rt_task_delete(&blink_task);
 return 0;
}
```

En charge



Sans charge

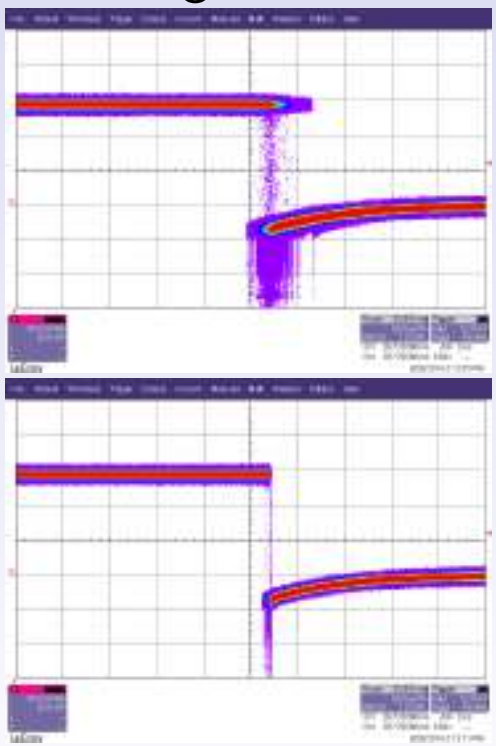
43. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [44] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
RT_TASK blink_task;

void blink(void *arg){
 int status=0;
 struct timespec tim = {0,TIMESLEEP};

 rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP*1000);
 while(1)
   {rt_task_wait_period(NULL);
    sunxi_gpio_output(PG9,status);
    status^=0xff;
   }
}

int main(int argc, char **argv) {
 sunxi_gpio_init();
 sunxi_gpio_set_cfgpin(PG9,OUTPUT);
 mlockall(MCL_CURRENT|MCL_FUTURE); // Avoid memory swapping for→
       ↪ this program
 rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
 rt_task_start(&blink_task, &blink, NULL);
 pause();
 rt_task_delete(&blink_task);
 return 0;}
```

En charge

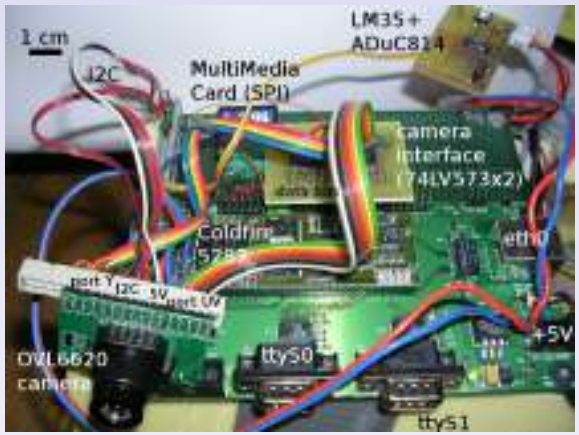


Sans charge

44. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Le temps réel : application

Exemple de l'acquisition d'une image :
sur un OS multitâches, il faut couper toutes les interruptions pendant la capture des pixels ⇒ perte du scheduler (monotâche) et de la communication réseau.

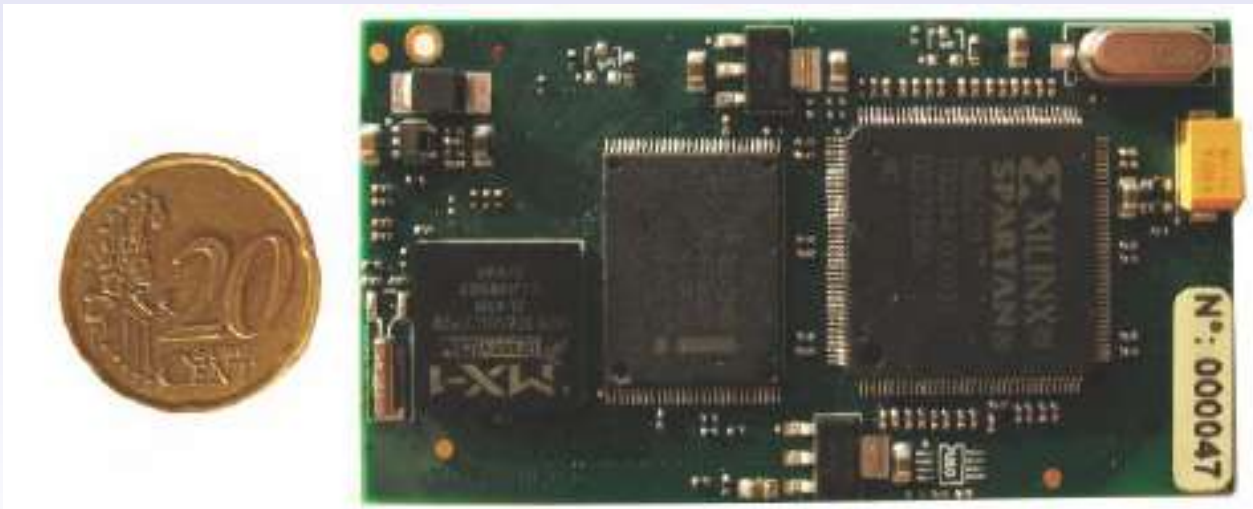


Exemple d'une carte son implémentée de façon logicielle
→ exploitation de la Queued ADC (tampon) [45]

45. S. Guinot, J.-M Friedt, *La réception d'images météorologiques issues de satellites : utilisation d'un système embarqué*, GNU/Linux Magazine France, Hors Série 24 (Février 2006)

Au dela des processeurs ...

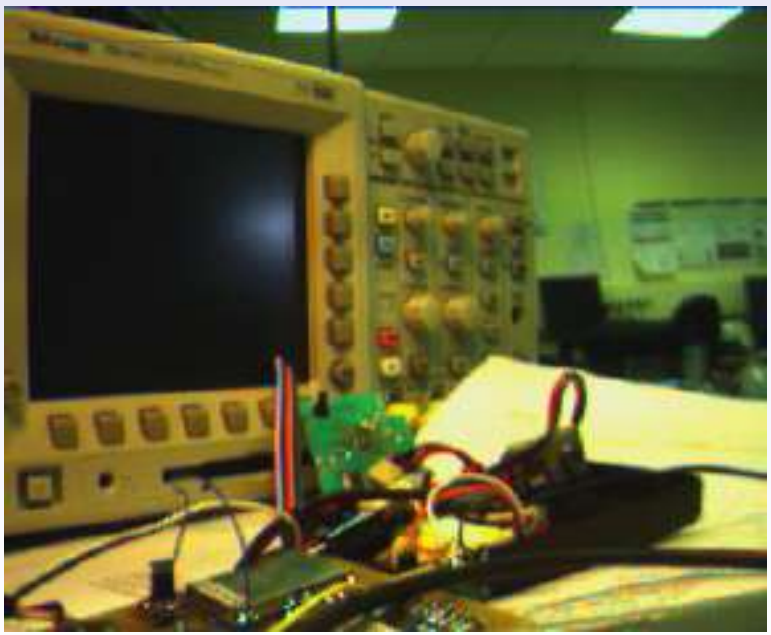
- Ajouter des périphériques pour décharger les processeur : coprocesseurs dédiés pour décharger le processeur de tâches spécifiques complexes (codec audio, ethernet).
- Alternative au temps réel : le matériel reconfigurable se charge des opérations rapides et décharge le processeur de tâches immédiates (UART, FIFO, ADC du coldfire par exemple) : temps-réel matériel
- SOC : Xilinx Zynq, Altera CycloneV



www.armadeus.org

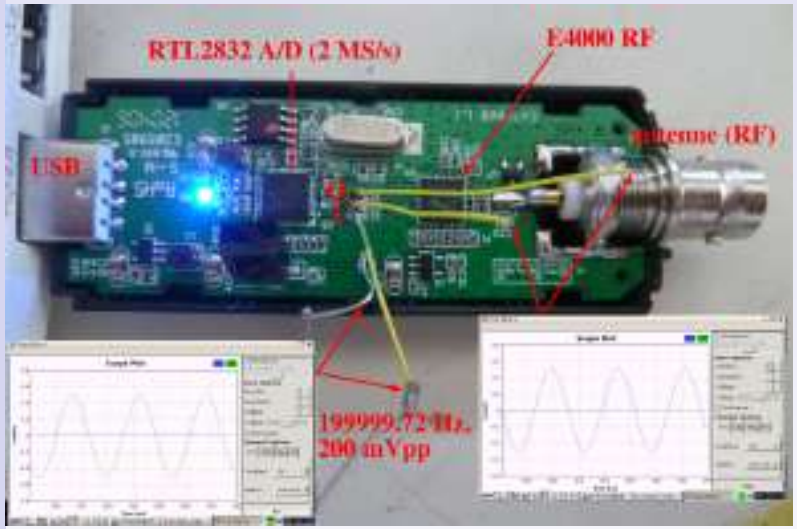
Matériel reconfigurable : le FPGA

- Le FPGA permet de s'adapter à chaque nouvelle application sans nécessiter d'ajouter du matériel (théorique : mise en forme des signaux).
- FPGA : Field Programmable Gate Array, matrice de plusieurs centaines de milliers de portes reconfigurables par logiciel.
- Une nouvelle philosophie : les signaux se propagent dans toutes les portes simultanément ⇒ "vrai" parallélisme ⇒ de nouveaux langages (VHDL, Verilog), `http://www.opencores.org/`
- capacité à implémenter un processeur dans un FPGA !



Radio définie par logiciel (SDR)

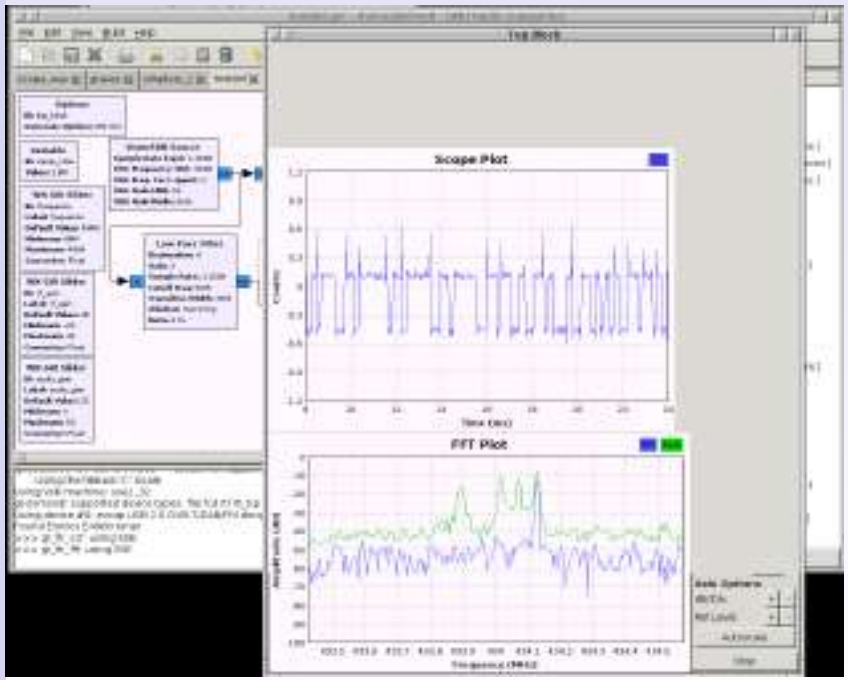
- `gnuradio`[46] et son environnement graphique associé `gnuradio-companion` fournissent les outils de traitement du signal de base (acquisition, filtres, démodulation, sortie audio ou fichier).
- EZCAP[47] basés sur le composant Elonics E4000 (récepteur DVB $\leq$20 \$ sur port USB) est un récepteur radiofréquence généraliste sans IF, avec un double-ADC à 2.5 MS/s



46. `gnuradio.org`
47. J.-M Friedt, G. Goavec-Mérou, *La réception radiofréquence définie par logiciel (Software Defined Radio – SDR)* GNU/Linux Magazine France 153 (Octobre 2012), pp.4-33, `jmfriedt.free.fr/lm_sdr.pdf`

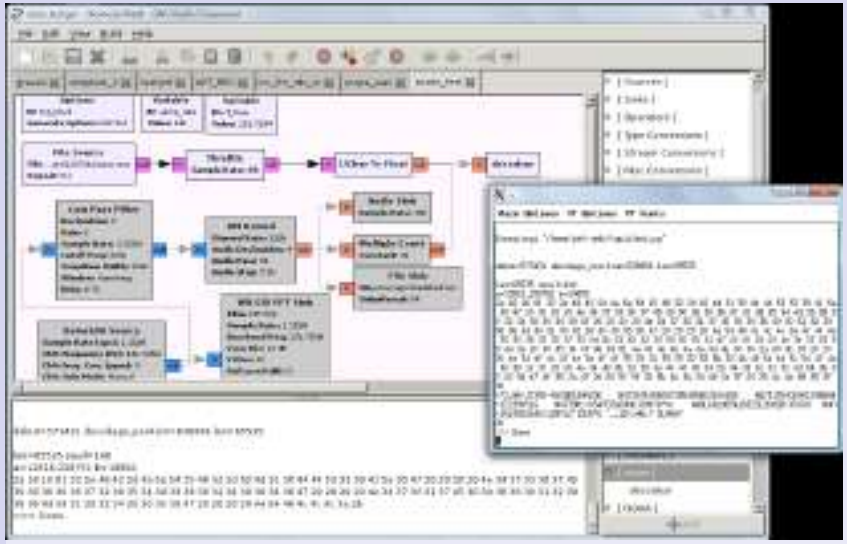
Démodulation de modes numériques (gnuradio)

- Traitement logiciel du signal radiofréquence ⇒ uniquement du logiciel à modifier pour traiter une nouvelle modulation/flux de données (WFM, AM, GPS, *pagers* ...).
- Unique limitation : *bande passante* du convertisseur analogique-numérique



1. Frequency shift keying (FSK) : la sortie du démodulateur représente directement le codage binaire
2. Retrouver l'information numérique se limite à une seuillage, identification du début du message, et calcul du code correcteur d'erreur (checksum)

Démodulation de modes numériques : ACARS (gnuradio)



Plus que l'utilisation des blocs de traitement disponibles : exploiter les fonctions de base de `gnuradio` (démodulation) et les compléter avec nos propres fonctionnalités :

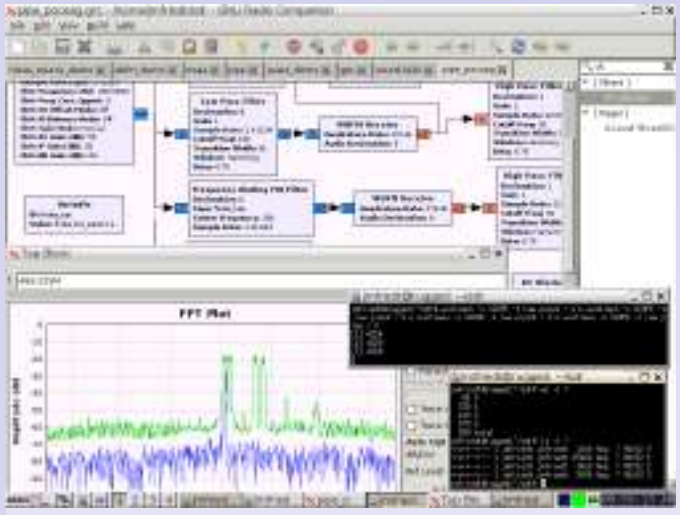
1. enregistrer le signal démodulé dans un fichier binaire (son)
2. développer tous les algorithmes de traitement du signal dans un langage interprété (GNU/Octave, SciPy)
3. convertir ces algorithmes en C(++)
4. respecter l'architecture de `gnuradio-companion`

Démodulation de modes numériques : ACARS (gnuradio)

Plus que l'utilisation des blocs de traitement disponibles : exploiter les fonctions de base de `gnuradio` (démodulation) et les compléter avec nos propres fonctionnalités :

1. enregistrer le signal démodulé dans un fichier binaire (son)
2. développer tous les algorithmes de traitement du signal dans un langage interprété (GNU/Octave, SciPy)
3. convertir ces algorithmes en C(++)
4. respecter l'architecture de `gnuradio-companion`

Les *pagers*

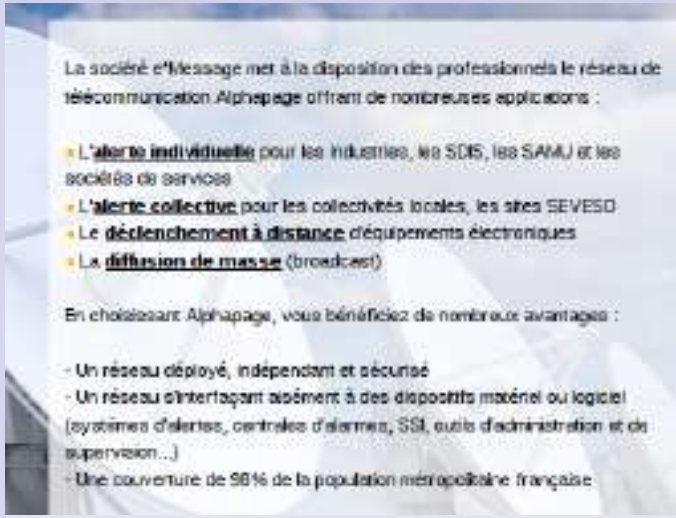


- Service grand public (tamtam, tatoo) devenu professionel
- 466 MHz, 6 canaux (`http://fr.wikipedia.org/wiki/POCSAG`)
- FSK, décodé par `multimon` (bas débit)

```
-TUDURI ANTHONY-AUXERRE Z1 69 BOURNEIL (Rue)-E10-Iph: BOUVIER-Eta: 2-Pte: 27-Prov: Vp-Mtf: H-25-A-DL GORGE,VOMI,TETE-101
A:10:30 De:DLP.HOTEL.HONEYWELL.disney.com Objet:space mountain                Msg:  <  space mountain
A:10:36 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:DIEGO ET SOS EN RTE SFE CH18019 BILAN DEMANDE TRAD UK MERCI PBX Msg:
-AMELOT ALAIN-AUXERRE Z1 3/33 AUSTERLITZ (Rue d')-C7-Eta: 1-Pte: 3/33-Prov: Vp-Mtf: H-66-A-VERTIG-APPEL DE L ASSOCIATION
A:10:45 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:APRES BILAN AVEC SAMU SFE CH18019 SOS MAINTENU MERCI PBX Msg:<EOT><E
A:10:48 De:aaipsi.grandlyon.org Objet:Inter Asset Msg:Inter dans Asset<EOT><EOT><NUL><NUL>
10:50:00 ESSAI BLS 1<EOT><EOT>
10:50:53 ESSAI BLS FF<EOT><EOT><NUL><NUL>
A:10:50 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:PALACE ET SOS MED EN RTE DLH CH3469 TRAD UK MERCI PBX Msg:<EOT><EOT>
-PIEL MICHELLE-AUXERRE Z1 2 CARRE du PUITS des DAMES -F9-Prov: Vp-Mtf: F-89-A-DL M INF-DOUBLE OPERATION DES HANCHES 2011
10:55:38 ESSAI BLS 3<EOT><EOT>
-PIEL MICHELLE-AUXERRE Z1 2 CARRE du PUITS des DAMES -F9-Prov: Vp-Mtf: F-89-A-DL M INF-DOUBLE OPERATION DES HANCHES 2011
083660111111<EOT><EOT>
07/08/2014 10:57:11 Origine OUE 0130987700 DEPART : GROUPE D'INTERVENTIONS (MULTIPLES) -CIS LES MUREAUX -RUE PIERRE CURI
DI BAT EF NBC *<EOT><EOT>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
```

48. J.-M Friedt, *La réception de signaux venus de l'espace par récepteur de télévision numérique terrestre*, OpenSilicium 13 (Dec 2014/Jan-Fev 2015)

Les *pagers*



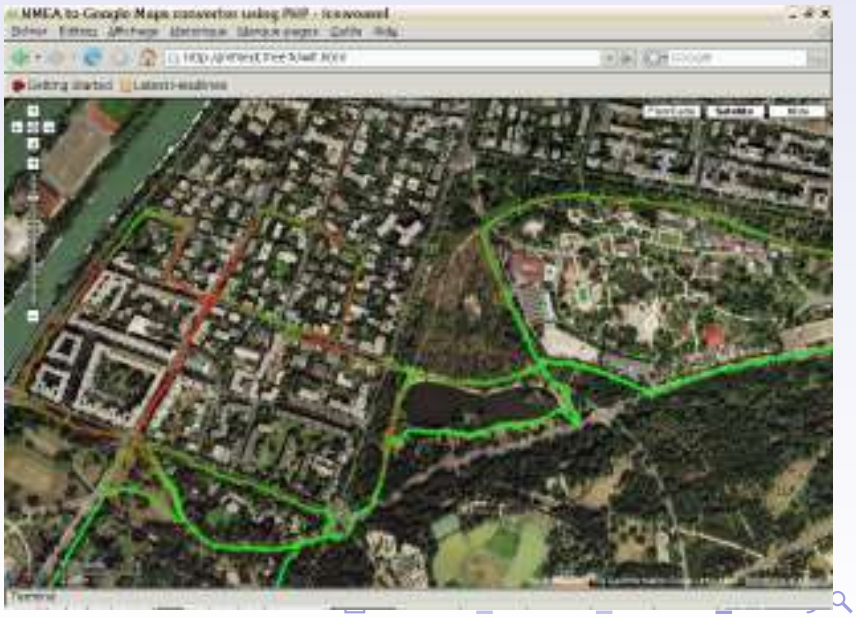
- Service grand public (tamtam, tatoo) devenu professionel
- 466 MHz, 6 canaux (`http://fr.wikipedia.org/wiki/POCSAG`)
- FSK, décodé par `multimon` (bas débit)

```
 http://www.alphapage.fr/index.aspx
-TUDURI ANTHONY-AUXERRE Z1 69 BOURNEIL (Rue)-E10-Iph: BOUVIER-Eta: 2-Pte: 27-Prov: Vp-Mtf: H-25-A-DL GORGE,VOMI,TETE-101
A:10:30 De:DLP.HOTEL.HONEYWELL.disney.com Objet:space mountain                Msg:  <  space mountain
A:10:36 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:DIEGO ET SOS EN RTE SFE CH18019 BILAN DEMANDE TRAD UK MERCI PBX Msg:
-AMELOT ALAIN-AUXERRE Z1 3/33 AUSTERLITZ (Rue d')-C7-Eta: 1-Pte: 3/33-Prov: Vp-Mtf: H-66-A-VERTIG-APPEL DE L ASSOCIATION
A:10:45 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:APRES BILAN AVEC SAMU SFE CH18019 SOS MAINTENU MERCI PBX Msg:<EOT><E
A:10:48 De:aaipsi.grandlyon.org Objet:Inter Asset Msg:Inter dans Asset<EOT><EOT><NUL><NUL>
10:50:00 ESSAI BLS 1<EOT><EOT>
10:50:53 ESSAI BLS FF<EOT><EOT><NUL><NUL>
A:10:50 De:DLP.VILLAGE.URGENCE.STD.disney.com Objet:PALACE ET SOS MED EN RTE DLH CH3469 TRAD UK MERCI PBX Msg:<EOT><EOT>
-PIEL MICHELLE-AUXERRE Z1 2 CARRE du PUITS des DAMES -F9-Prov: Vp-Mtf: F-89-A-DL M INF-DOUBLE OPERATION DES HANCHES 2011
10:55:38 ESSAI BLS 3<EOT><EOT>
-PIEL MICHELLE-AUXERRE Z1 2 CARRE du PUITS des DAMES -F9-Prov: Vp-Mtf: F-89-A-DL M INF-DOUBLE OPERATION DES HANCHES 2011
083660111111<EOT><EOT>
07/08/2014 10:57:11 Origine OUE 0130987700 DEPART : GROUPE D'INTERVENTIONS (MULTIPLES) -CIS LES MUREAUX -RUE PIERRE CURI
DI BAT EF NBC *<EOT><EOT>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
RADOPS01 : Essai quotidien (Astreinte DIN) - RADOPS01=OK.<EOT><EOT><NUL>
```

48. J.-M Friedt, *La réception de signaux venus de l'espace par récepteur de télévision numérique terrestre*, OpenSilicium 13 (Dec 2014/Jan-Fev 2015)

Les liaisons radiofréquence

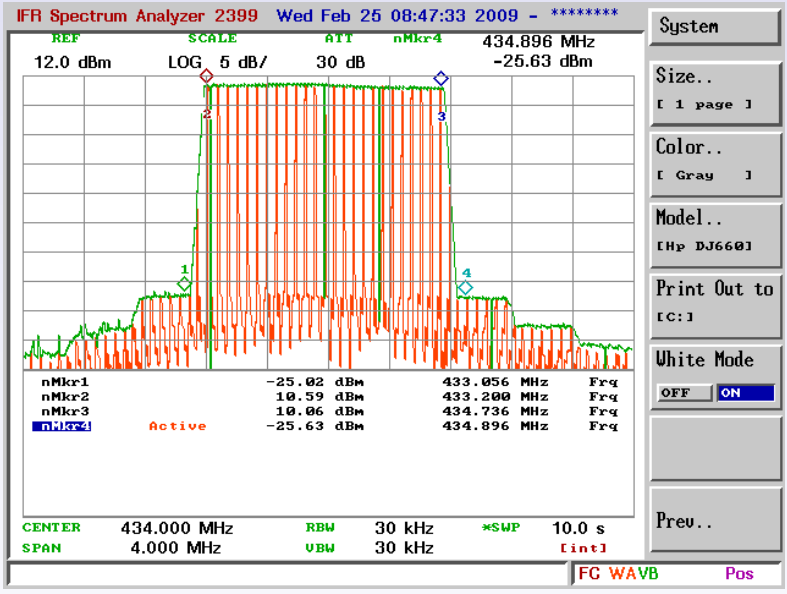
- très grourmandes en énergie ⇒ prévoir des stratégies de mise en veille ⇒ stcokage des données et compression
- peu fiables
- diverses technologies en fonction des objectifs : radiomodems, wifi, bluetooth/zigbee
- l'utilisation dépend des objectifs. Penser à la redondance (stockage local).
- performances variables selon que le système soit mobile ou statique
- plus la fréquence est basse, plus la portée est longue mais plus l'antenne est encombrante
- sécurité ?



Certifications

Un produit certifié doit respecter un certain nombre de contraintes

- robustesse à l'alimentation (bruit fourni par le secteur)
- ne pas perturber autrui (bruit généré le secteur)
- ne pas rayonner involontairement (respect des bandes et gabarits d'émission électromagnétiques)
- être résistant aux perturbations extérieures [49]
- ESD (4-8 kV) sur boîtier et connecteurs



49. norme ETSI EN 300 220-1 V2.1.1 (2006-04), disponible à http://www.etsi.org/WebSite/Standards/StandardsDownload.aspx

Combien ça coûte ?

Modèle	taille	prix	quantités
ADuC814	8	4.57 $	1000 AD
MSP430F149	16	7.55 $	100 TI
68HC912B32	16	12.22 $	1000 Freescale
H8/3048	16	1000 yens=6.2 euros	1 akizuki
ADuC7026	32	8.91 $	1000 AD
Fox	32 linux	139 euros	1 ACME
SSV 5280	32 linux	162 euros	1 lextronic
Arcturus 5272	32 linux	100 $	500 arcturus
Arcturus 5282	32 linux	100 $	500 arcturus
iMX + FPGA	32 linux	120 euros	1 armadeus
PCM-9387F-M0A2E	32 linux	245 euros	1 advantech (3.5" SBC)
PCM-3380F-S4A2E	32 winCE	395 euros	1 advantech (PC104)
Fonera, Rasberry PI	32 linux	30 euros	1

Conclusion

- le développement de systèmes embarqués nécessite de multiples compétences : électronique analogique et numérique, informatique, programmation mais aussi mécanique, RF ...
- l'exploitation optimale de ressources réduites nécessite de bien comprendre son système au niveau matériel
- seul un choix judicieux des outils permet d'atteindre un résultat efficace $\Rightarrow$ se tenir au courant des évolutions, domaine très dynamique
- méthode fondamentale pour réduire la consommation : **dormir**
- domaine en plein essor avec la mode des instruments mobiles et autonomes
- disponibilité d'outils libres performants pour s'initier au domaine


A. Tanenbaum, *Structures computer organization, 2nd Ed.*, Prentice Hall International editions (1984)


D.E. Knuth, *The art of computer programming*, **Vol. 4** "A draft of section 7.1.3 : Bitwise tricks and techniques", Addison-Wesley (2008), disponible à `www-cs-faculty.stanford.edu/~knuth/fasc1a.ps.gz`


M. Rafiquzzaman, *Microprocessors and microcomputer-based system design, 2nd Ed.*, CRC Press (1995)


T. Shanley & D. Anderson, *ISA System Architecture*, Mindshare Inc. Addison & Wesley (1995)


R. Gaonkar, *The Z80 microprocessors*, Merrill Macmillan Publishing (1993)


S.H. Hollingdale, *High speed computing – methods and applications*, The English Universities Press (1959)


J.-C. Buisson, *Concevoir son microprocesseur*, Ellipses (2006), disponible à `http://diabeto.enseeiht.fr/download/divers/archibook.pdf`


G. Brocard, *La propulsion électrique des modèles réduits d'avions et de planeurs*, New Power Modélisme Édition (2004)


C. Nagy, *Embedded Systems Design Using the TI MSP430 Series*, Newnes (2003) ou plus utile, SLAP0124 à `http://focus.ti.com/lit/ml/slap124/slap124.pdf` et SLAA024 (*MSP430 Family Mixed-Signal Microcontroller Application Reports*)


J. Corbet, A. Rubini & G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly (2005), disponible à `http://lwn.net/Kernel/LDD3/`


W.R. Stevens, *TCP/IP Illustrated Vol. 1*, Addison & Wesley (1994)


L. Torvalds & D. Diamond, *Just for Fun : The Story of an Accidental Revolutionary*, Collins (2002)


S. Wozniak & G. Smith, *iWoz : Computer Geek to Cult Icon : How I Invented the Personal Computer, Co-Founded Apple, and Had Fun Doing It*, Norton (2007)


A. Hertzfeld, *Revolution in The Valley : The Insanely Great Story of How the Mac Was Made*, O'Reilly Media (2004)


K. Hafner & M. Lyon, *Where Wizards Stay Up Late*, Simon & Schuster (1998)


D.A. Mindell, *Digital Apollo – Human and Machine in Spaceflight*, MIT Press (2008)


D.A. Mindell, *Between Human and Machine : Feedback, Control, and Computing before Cybernetics*, Johns Hopkins University Press, 2004,

`http://video.mit.edu/watch/between-human-and-machine-feedback-control-and-computing-before-cyberr`


E.C. Hall, *Journey to the Moon : The History of the Apollo Guidance Computer*, AIAA Inc. (1996)