Informatique embarquée 1/16

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

11 janvier 2018

16 séances/3 h

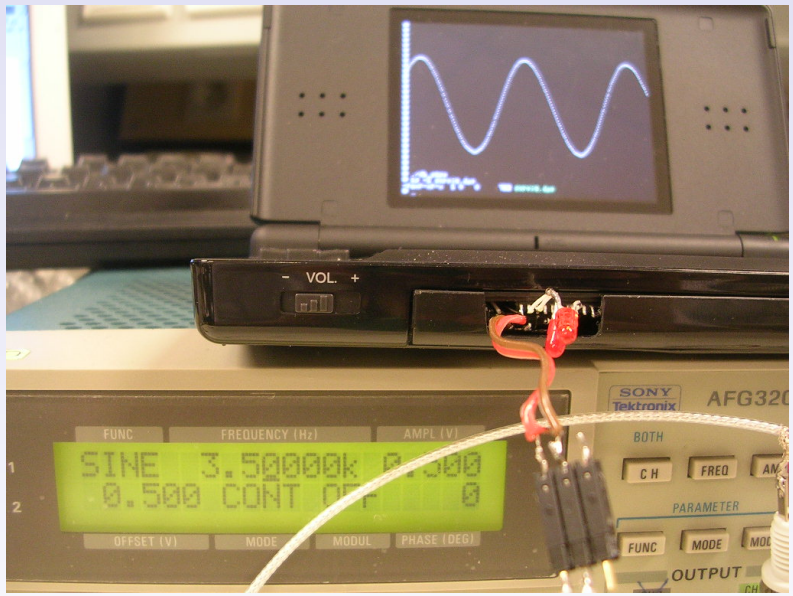
Plan des cours

1. registres de config d'un uC, lecture de datasheet, cross compilation, bootloader, flasher, description de l'architecture d'un microcontrôleur et périphériques
   → application sur ATMega32 (AVR), optimisations du C
2. C pour microcontroleurs : volatile, tailles de variables, optimisations, C v.s asm, acces aux registres, *(type*)addr=val ;, interruptions, description de gcc et étapes de compilation
   → application aux périphériques matériels & interruptions de ATMega32
3. protocoles de communication synchrones/asynchrones, bus locaux
4. du 8 au 32 bits : exploitation de bibliothèques, et en particulier libc, implications sur les ressources ; conversion analogique-numérique et traitement du signal
   → création de sa toolchain ARM Cortex+newlib pour STM32 (stubs)
5. Environnements exécutifs FreeRTOS/NuttX/RTEMS/TinyOS → TP FreeRTOS/RTEMS sur STM32
6. Linux sur un systeme embarqué : ordonnanceur, TCP/IP, système de fichiers ...
   → buildroot, exploitation du même programme sur ARM et PC, aspects utilisateur
7. unix sur système embarqué, aspects développeur, MMU et accès au matériel
   → kernel module, gestion des interruptions, méthodes mises à disposition par le noyau (mutex/sémaphore, tasklet)
8. concept de temps-réel, maîtrise des latences introduites par l'ordonnanceur, extension RT de Linux

Exposé d'un binôme : 10 minutes+5 minutes (5-6 transparents)

Introduction

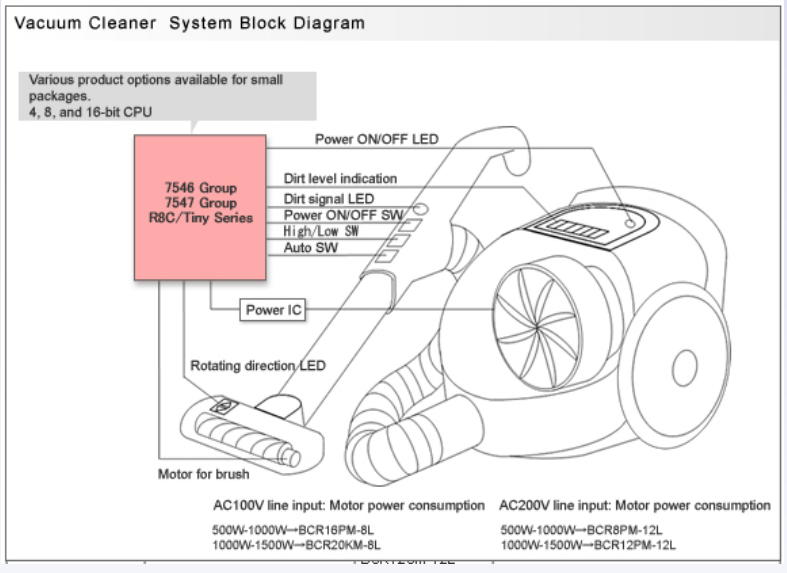
1. qu'est-ce qu'un microcontrôleur ?
2. comment appréhender un nouveau microcontrôleur ? lecture d'une *datasheet*
3. quels outils pour travailler sur un microcontrôleur ?

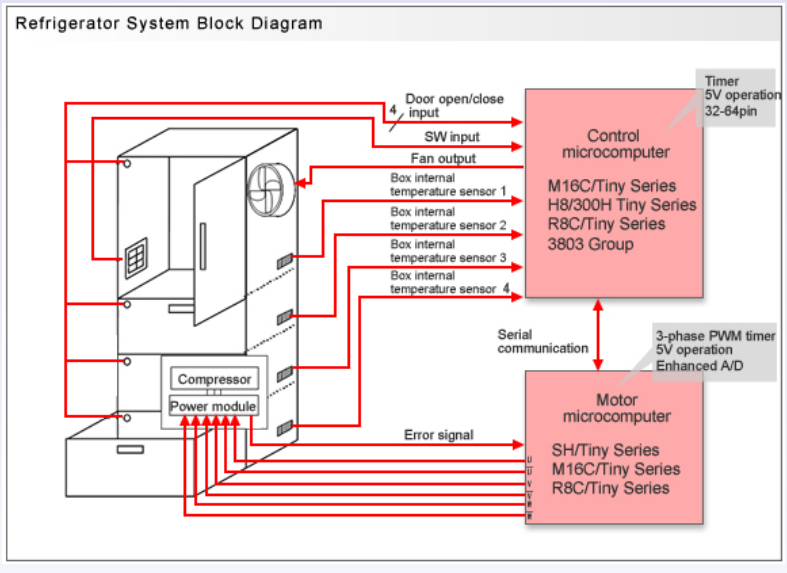


Quelques exemples de systèmes embarqués grand public.

Introduction

1. qu'est-ce qu'un microcontrôleur ?
2. comment appréhender un nouveau microcontrôleur ? lecture d'une *datasheet*
3. quels outils pour travailler sur un microcontrôleur ?

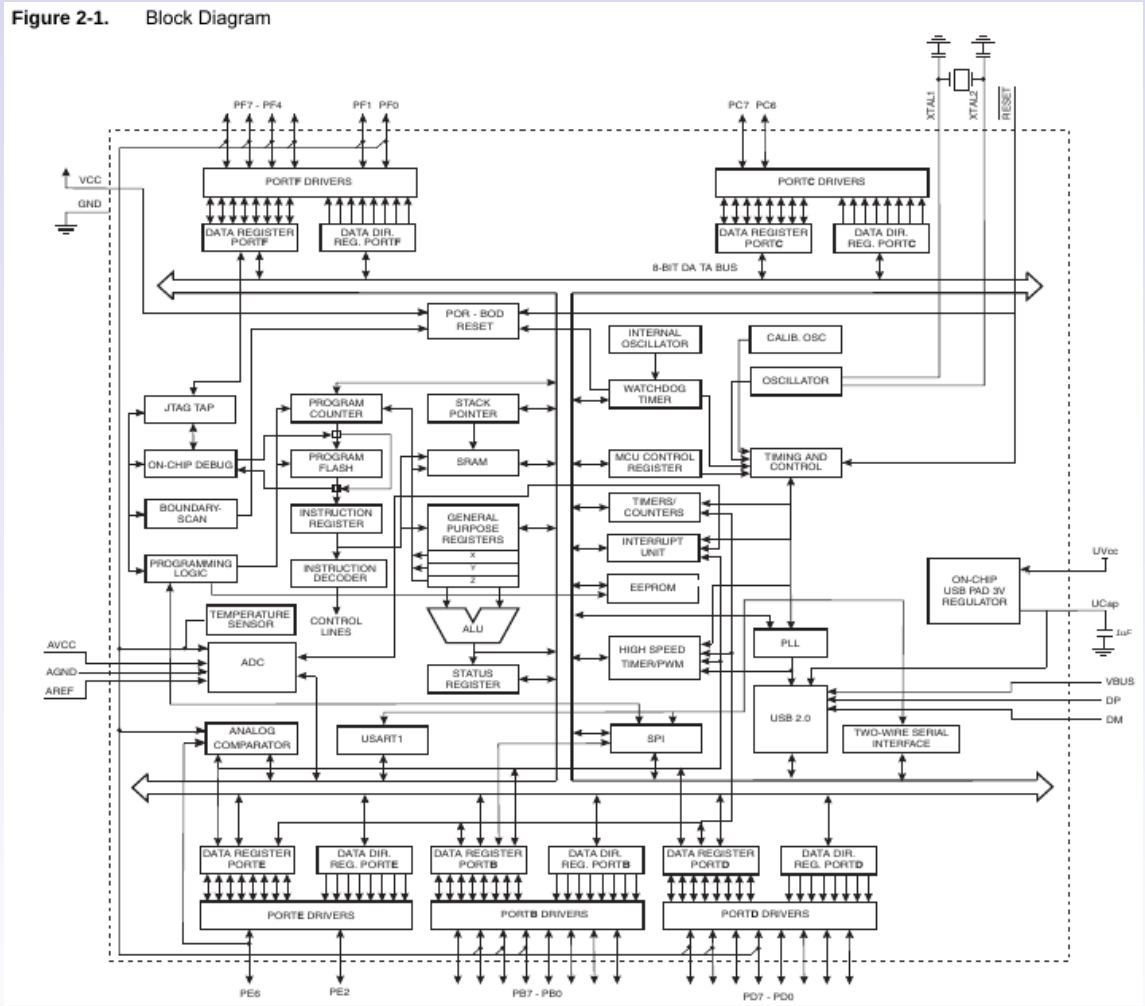




Quelques exemples de systèmes embarqués (Renesas), et choix des composants numériques associés.

Qu'est-ce qu'un microcontrôleur ?

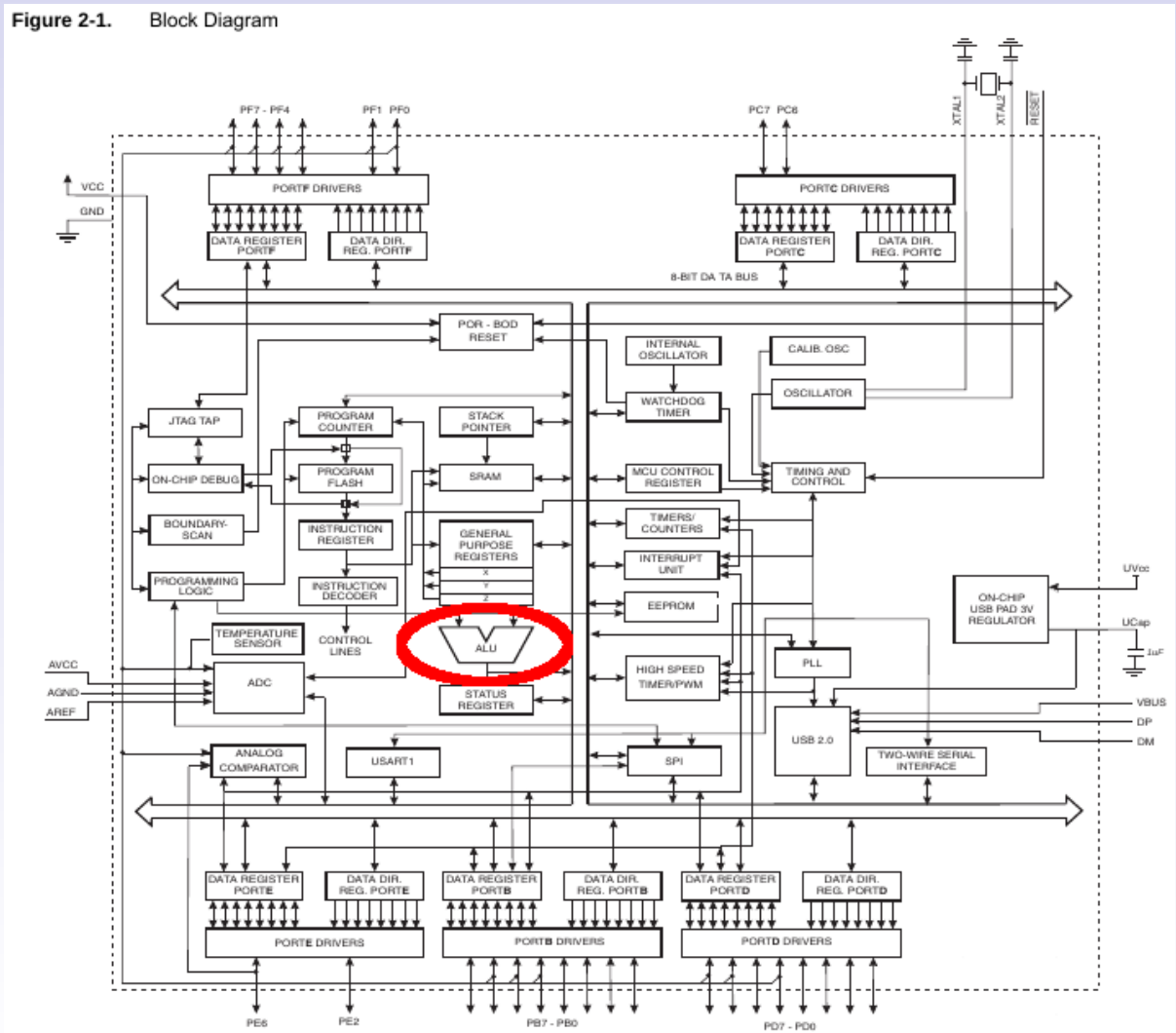
- une unité de calcul – unité arithmétique et logique – chargée de décoder des instructions et de les appliquer sur des opérandes



p.4 de ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf

Qu'est-ce qu'un microcontrôleur ?

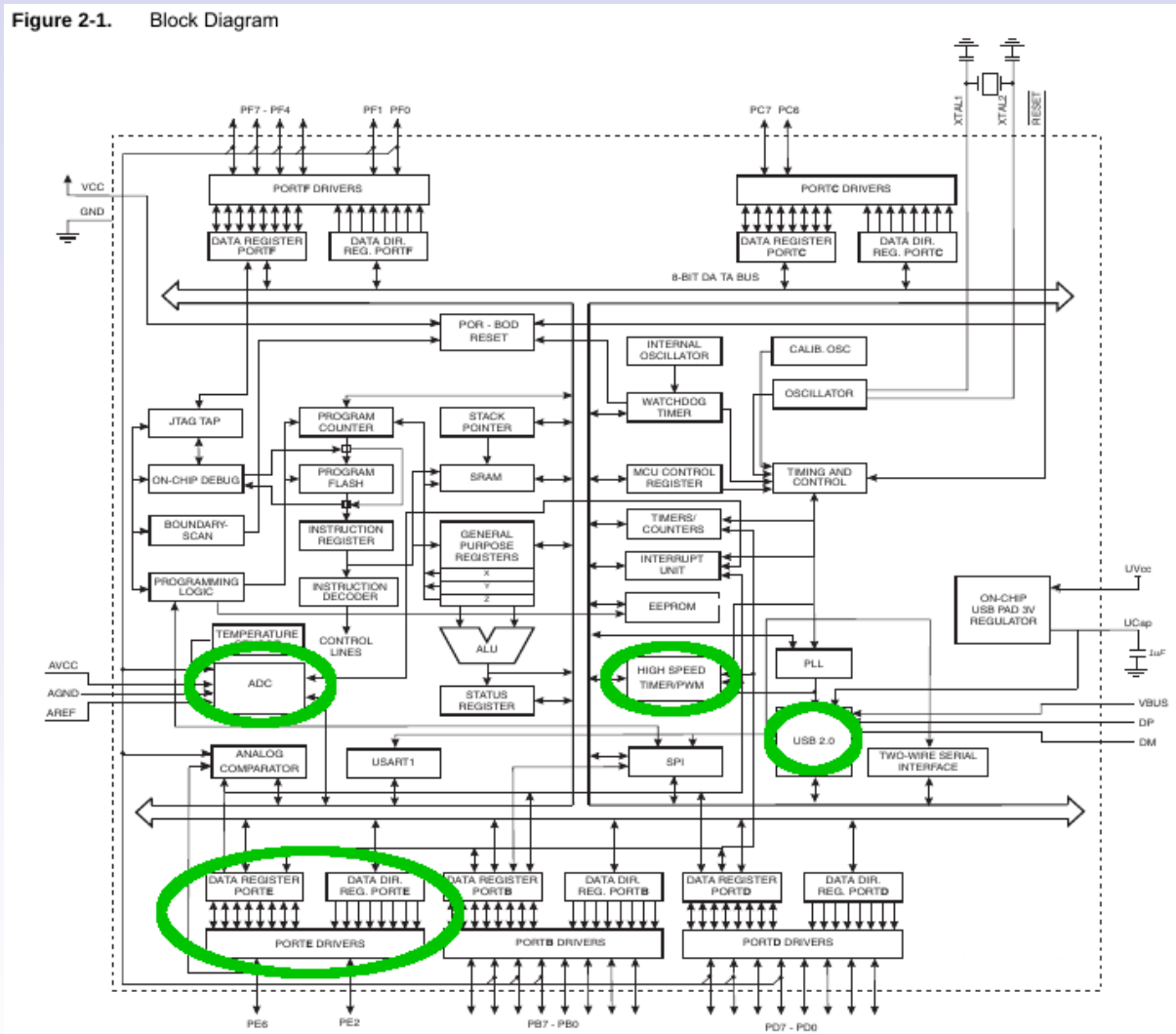
- une unité de calcul – unité arithmétique et logique – chargée de décoder des instructions et de les appliquer sur des opérandes



Architecture du processeur 8-bits Atmega 32U4

Qu'est-ce qu'un microcontrôleur ?

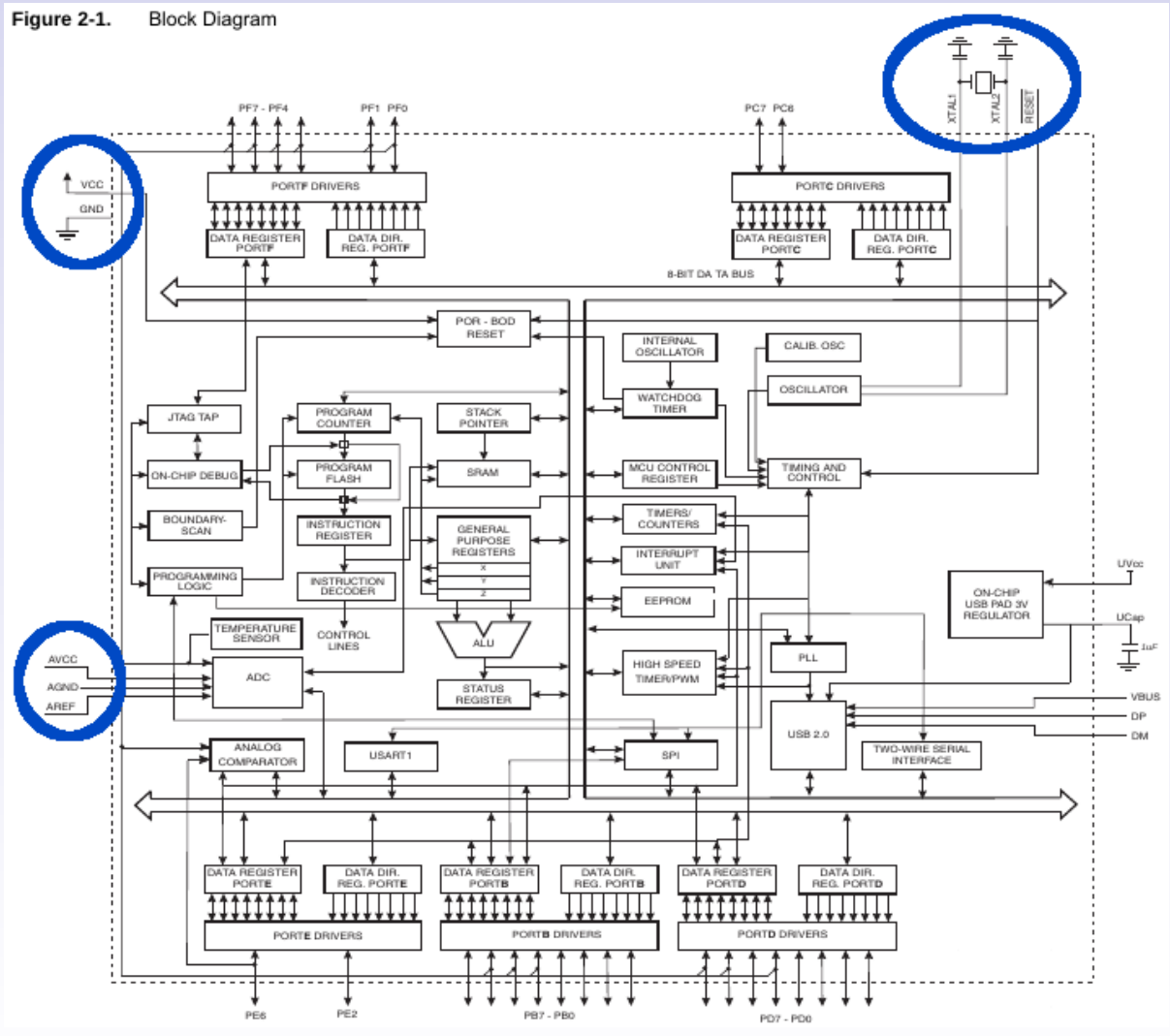
- des périphériques, souvent intégrés au sein de la même puce (compteur, générateur de signaux de communication ...)



Architecture du processeur 8-bits Atmega 32U4

Qu'est-ce qu'un microcontrôleur ?

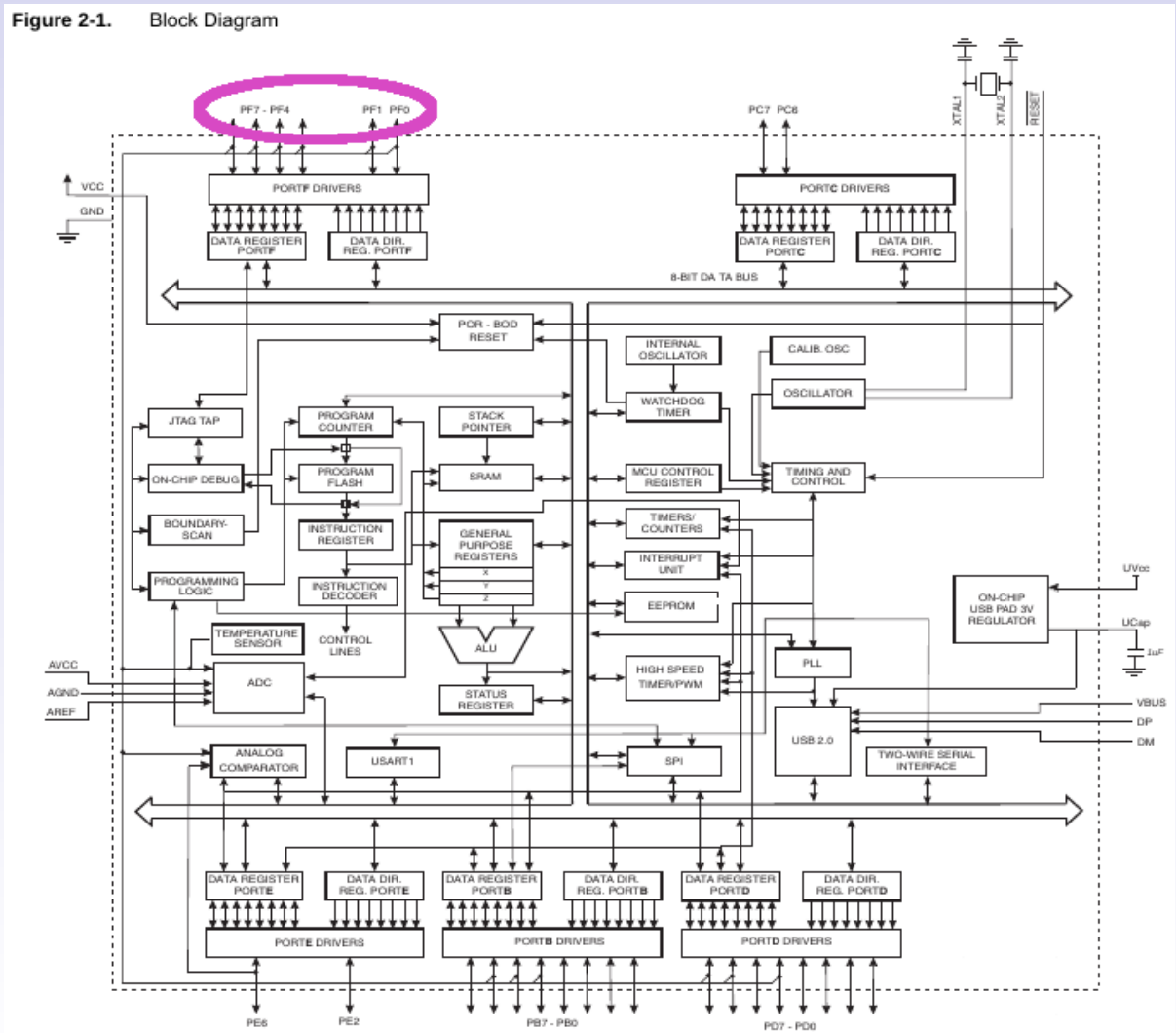
- en entrée : alimentation, **horloges** (circuit synchrone), tensions de référence



Architecture du processeur 8-bits Atmega 32U4

Qu'est-ce qu'un microcontrôleur ?

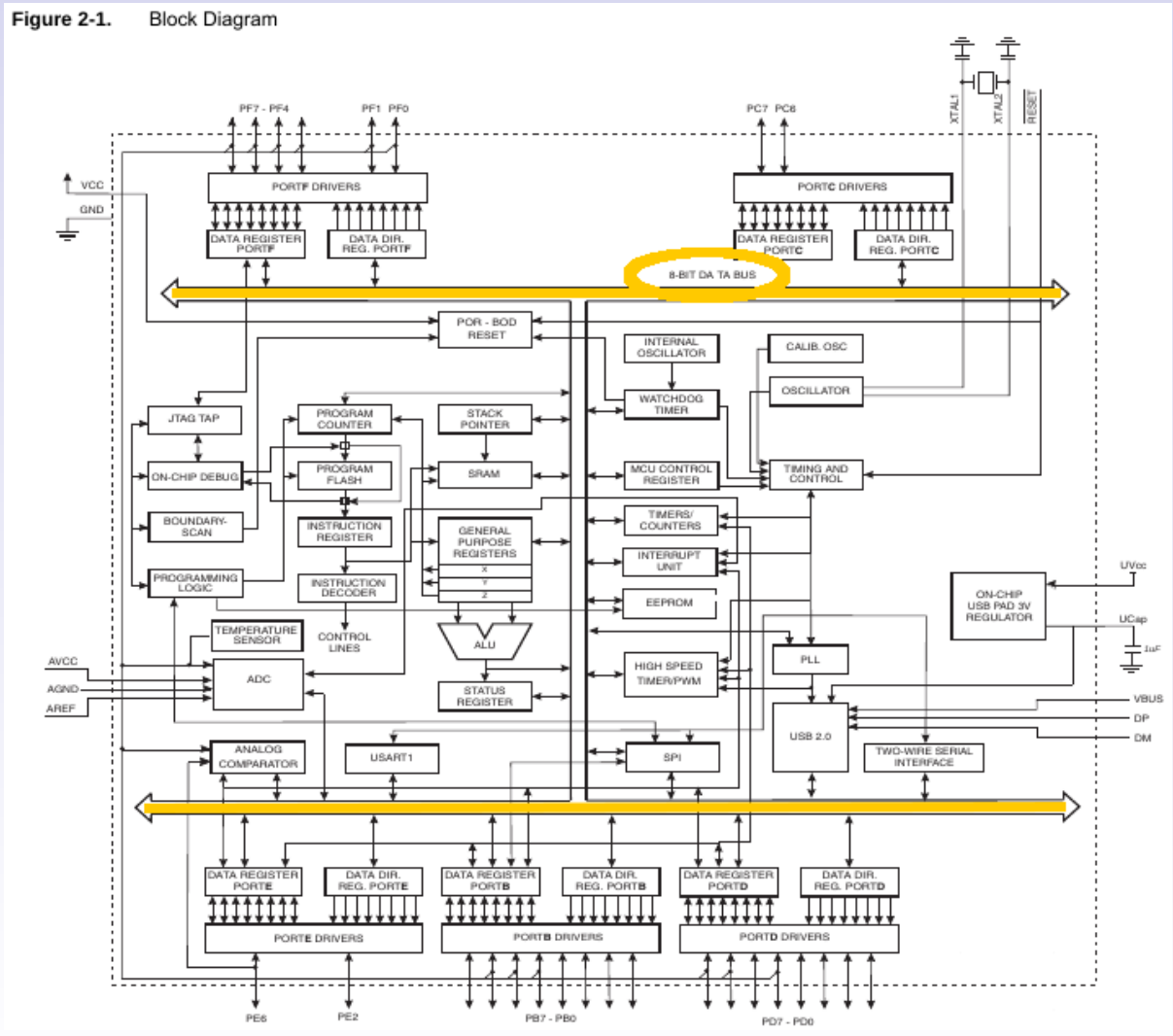
- en sortie : des signaux numériques (0/1=mass/alimentation) ou analogiques, des signaux périodiques



Architecture du processeur 8-bits Atmega 32U4

Qu'est-ce qu'un microcontrôleur ?

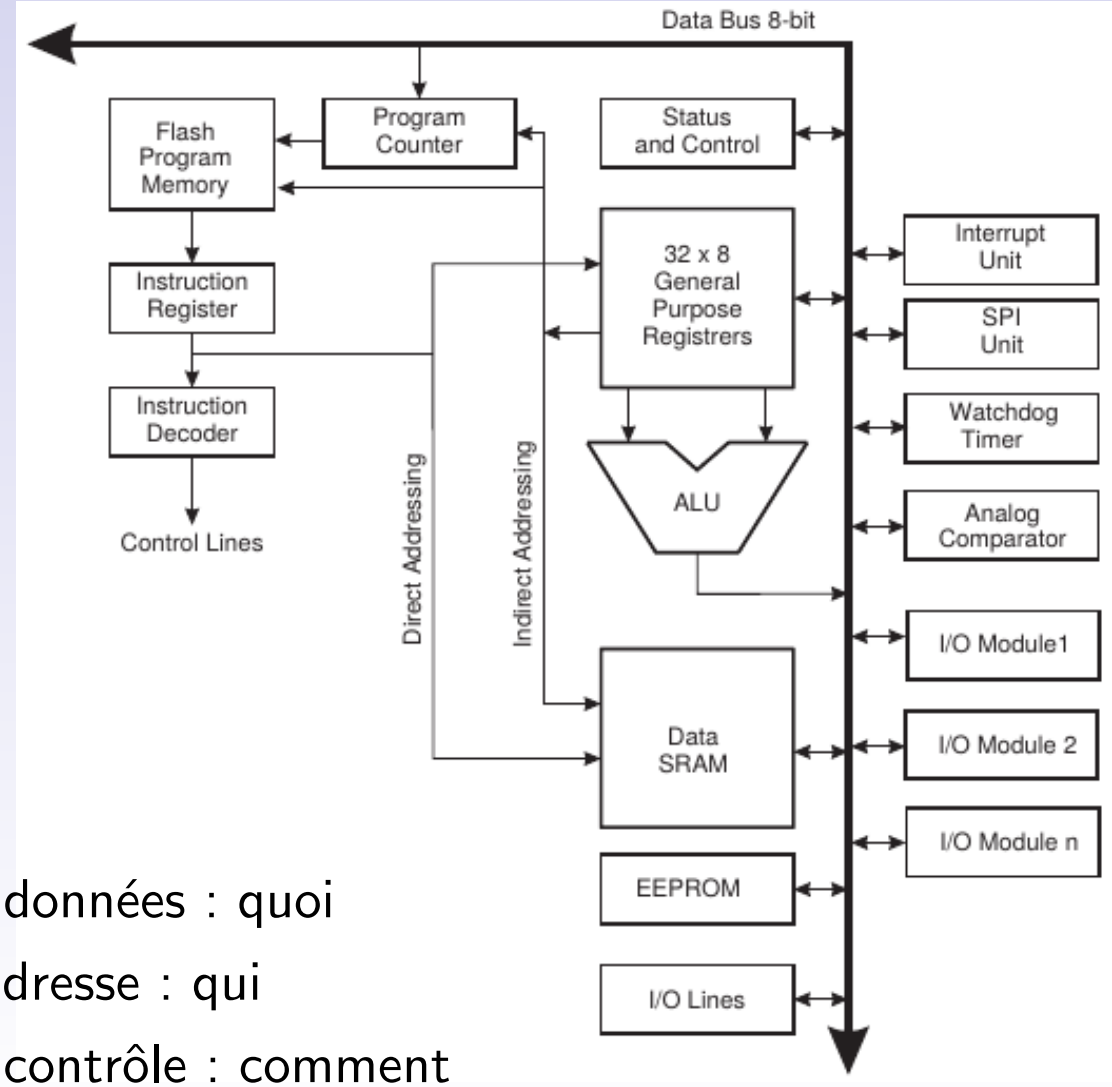
- interaction avec le monde extérieur au travers de bus de communication



Architecture du processeur 8-bits Atmega 32U4

Qui fait quoi ? les registres

Liaison entre ALU et les périphériques : adressage



- bus de données : quoi
- bus d'adresse : qui
- bus de contrôle : comment

Qui fait quoi ? les registres

Comment connaître la position de chaque registre et sa fonction ?

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1	
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	-	-	-	-	-	-	-	-	
0x13 (0x33)	Reserved	-	-	-	-	-	-	-	-	
0x12 (0x32)	Reserved	-	-	-	-	-	-	-	-	
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	-	-	PORTF1	PORTF0	
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	-	-	DDF1	DDF0	
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	-	-	PINF1	PINF0	
0x0E (0x2E)	PORTE	-	PORTE6	-	-	-	PORTE2	-	-	
0x0D (0x2D)	DDRE	-	DDE6	-	-	-	DDE2	-	-	
0x0C (0x2C)	PINE	-	PINE6	-	-	-	PINE2	-	-	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	
0x08 (0x28)	PORTC	PORTC7	PORTC6	-	-	-	-	-	-	
0x07 (0x27)	DDRC	DDC7	DDC6	-	-	-	-	-	-	
0x06 (0x26)	PINC	PINC7	PINC6	-	-	-	-	-	-	
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	
0x02 (0x22)	Reserved	-	-	-	-	-	-	-	-	

Table 10-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Qui fait quoi ? les registres

Comment connaître la position de chaque registre et sa fonction ?

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1	
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	-	-	-	-	-	-	-	-	
0x13 (0x33)	Reserved	-	-	-	-	-	-	-	-	
0x12 (0x32)	Reserved	-	-	-	-	-	-	-	-	
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	-	-	PORTF1	PORTF0	
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	-	-	DDF1	DDF0	
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	-	-	PINF1	PINF0	
0x0E (0x2E)	PORTE	-	PORTE6	-	-	-	PORTE2	-	-	
0x0D (0x2D)	DDRE	-	DDE6	-	-	-	DDE2	-	-	
0x0C (0x2C)	PINE	-	PINE6	-	-	-	PINE2	-	-	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	
0x08 (0x28)	PORTC	PORTC7	PORTC6	-	-	-	-	-	-	
0x07 (0x27)	DDRC	DDC7	DDC6	-	-	-	-	-	-	
0x06 (0x26)	PINC	PINC7	PINC6	-	-	-	-	-	-	
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	
0x02 (0x22)	Reserved	-	-	-	-	-	-	-	-	

La *datasheet* fournit la description complète du processeur. Ici, Atmel, *8-bit Microcontroller with 16/32K Bytes of ISP Flash and USB Controller – ATmega16U4 & ATmega32U4*, disponible à `ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf`

Masques

1. OU : mettre les bits à 1 ($\forall$ OU $1 = 1$)
2. ET : mettre les bits à 0 ($\forall$ AND $0 = 0$)
3. fabriquer le masque du ET : $!(1 \ll N)$ met des 1 sur tous les bits sauf le Nème
4. XOR : intervertir les bits (X XOR $1 = !X$ alors que X XOR $0 = X$)

Tables de vérité :

OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

XOR	0	1
0	0	1
1	1	0

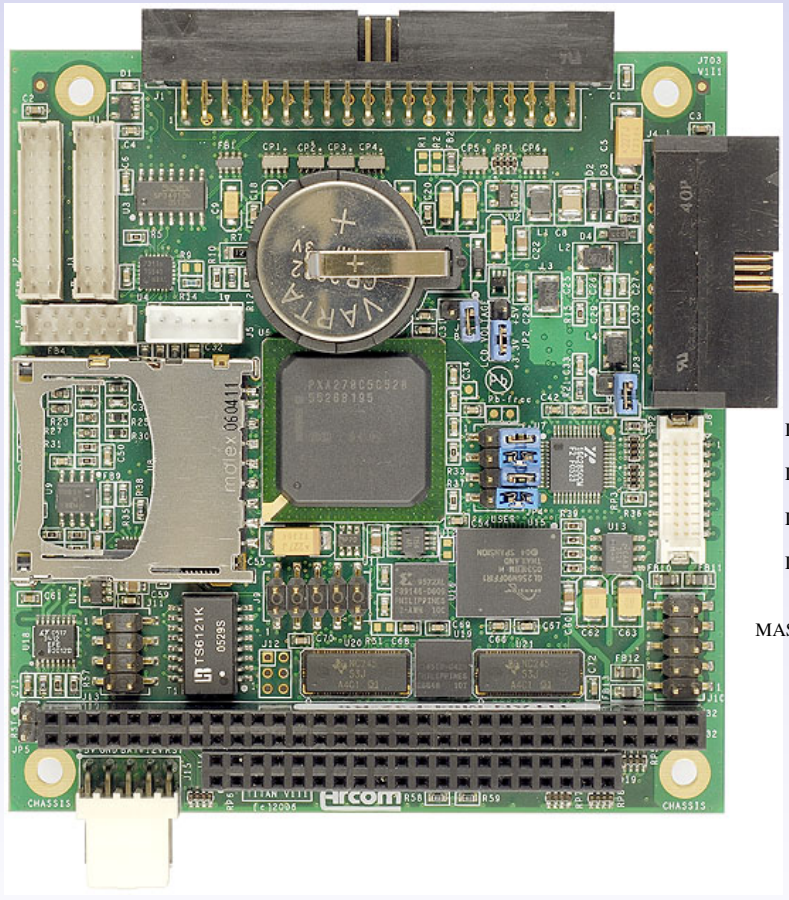
Logiciel embarqué

Absence de système d'exploitation

- le programme commence par une initialisation (pile, variables, périphériques)
- le programme se poursuit par une boucle infinie `while (1) {...}`
- le processeur ne doit **jamais** être laissé sans instruction à exécuter

Lien entre le logiciel et le matériel

1. tout périphérique communique avec un microprocesseur au travers de 3 bus : *data*, *address* et *control* Le bus prend son origine *dans* le processeur mais est accessible à l'extérieur.



D		C
GND	0	GND
M16#		SBHE#
IO16#		LA23
IRQ10		LA22
IRQ11		LA21
IRQ12		LA20
IRQ15		LA19
IRQ14		LA18
DACK0#		LA17
DRQ0		MRDC#
DACK5#		MWTC#
DRQ5		SD08
DACK6#		SD09
DRQ6		SD10
DACK7#		SD11
DRQ7		SD12
+5		SD13
MASTER16#		SD14
GND		SD15
GND	19	KEY

A		B
CHCHK#	1	GND
SD7		RESDRV
SD6		+5
SD5		IRQ9
SD4		–5
SD3		DRQ2
SD2		–12
SD1		ENDXFR#
SD0		+12
CHRDY		KEY
AEN		SMWTC#
SA19		SMRDC#
SA18		IOWC#
SA17		IORC#
SA16		DACK3#
SA15		DRQ3
SA14		DACK1#
SA13		DRQ1
SA12		REFRESH#
SA11		CLK
SA10		IRQ7
SA09		IRQ6
SA08		IRQ5
SA07		IRQ4
SA06		IRQ3
SA05		DACK2#
SA04		TC
SA03		BALE
SA02		+5
SA01		OSC
SA00		GND
GND	32	GND

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

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);
}
```

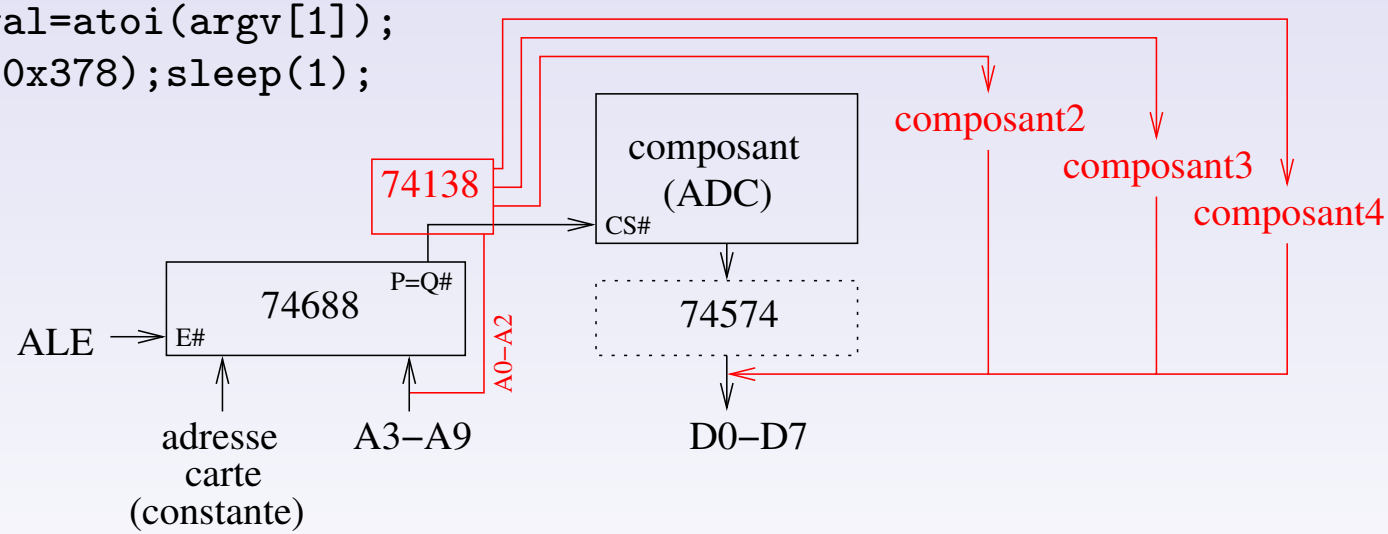


Schéma générique de connection d'un composant à un bus

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

Périphériques matériels : GPIO

General Purpose Input Output : entrée-sortie numérique

- Un *latch*, aussi appelé GPIO, mémorise son entrée sur un coup d'horloge ou un niveau de commande
- 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.

Périphériques matériels : *timers*

- Un microprocesseur est un composant numérique synchrone : base de temps cadence toutes les opérations
- Des compteurs internes sont configurés pour diverses tâches :
  - déclencher périodiquement un évènement (*e.g.* ordonnanceur préemptif de Linux (`kernel/timer.c`

```
signed long __sched schedule_timeout(signed long timeout)
{ struct timer_list timer;
  unsigned long expire;
  switch (timeout)
        {case MAX_SCHEDULE_TIMEOUT:
           schedule();
           goto out;
        default:
           if (timeout < 0) {dump_stack();
              current->state = TASK_RUNNING;
              goto out;
            }
        }
        expire = timeout + jiffies;
        setup_timer_on_stack(&timer, process_timeout, (unsigned lo
        __mod_timer(&timer, expire, false, TIMER_NOT_PINNED);
        schedule();
...
}
```

Périphériques matériels : *timers*

- Un microprocesseur est un composant numérique synchrone : base de temps cadence toutes les opérations
- Des compteurs internes sont configurés pour diverses tâches :
  - déclencher périodiquement un évènement (*e.g.* ordonnanceur préemptif de Linux (`kernel/timer.c`
  - déclencher une évènement sur un GPIO sans contrôle explicite du logiciel (oscillation périodique, largeur d'impulsion – PWM)
  - capturer un évènement (*Input Capture*) pour le dater
  - commander l'activation d'autres périphériques

Outils de compilation

- Toutes nos applications exploiteront gcc pour convertir un code source en C vers un binaire exécutable sur la cible
- la cible n'est *pas* basée sur l'architecture x86 d'intel du PC ⇒ cross-compilation
- gcc est exploitable pour à peu près toutes les architectures : liste des cibles supportées par gcc [1]

```
* 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
```

⇒ un outil unique à maîtriser pour s'affranchir des limitations sur le logiciel

1. http://www.gnu.org/software/gcc/backends.html

Outils de compilation

- le *bootloader* se charge de réceptionner les octets émis par le PC à destination du microcontrôleur et de les écrire en mémoire
- un programme spécifique sur le PC fournit les données dans le format attendu (programmeur)
- `binutils` fournit les outils de conversion entre formats : `objdump` et `objcopy`

Mise en pratique

Premier pas sur l'Atmega 32U4

- Outils : sous Debian/GNU Linux, paquets `avrdude`, `gcc-avr` et `avr-libc`
- Carte : Olimex