Une plateforme de développement rapide pour DSP563xx

Application à l'enseignement des Transmissions Numériques

Jean-Marie ORY

École Supérieure des Sciences et Technologies de l'Ingénieur de Nancy

Centre de Recherche en Automatique de Nancy

Université Henri Poincaré

jean-marie.ory@esstin.uhp-nancy.fr

1 La plateforme DSP563xx

1.1 Introduction

Le Traitement du Signal en Temps Réel: contraintes et solutions

1.1.1 De l'Analogique au Numérique … (1)

- Historiquement on a d'abord utilisé les phénomènes physiques pour traiter en temps réel des problèmes mathématiques:
- Ainsi, condensateurs et inductances sont utilisés pour leurs propriétés:
  - Opérateur de dérivation par rapport au temps d/dt

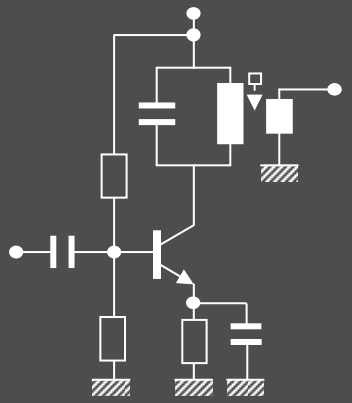
$$i = C\, dv/dt \qquad v = L\, di/dt$$

  - Opérateur d'intégration par rapport au temps $\int \ldots dt$

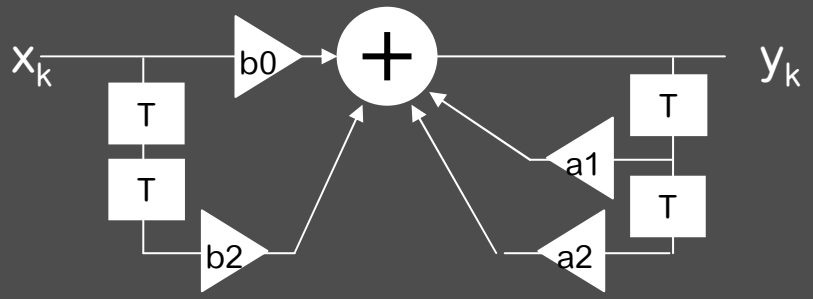
$$v = V_0 + 1/C \int i\, dt \qquad i = I_0 + 1/L \int v\, dt$$

- Jusqu'en 1970 on a utilisé de gros calculateurs analogiques pour simuler en temps réel de grands systèmes d'équations différentielles impossibles à résoudre analytiquement

1.1.1 De l'Analogique au Numérique



Etage d'ampli FI en technique analogique



Chaque Te faire:

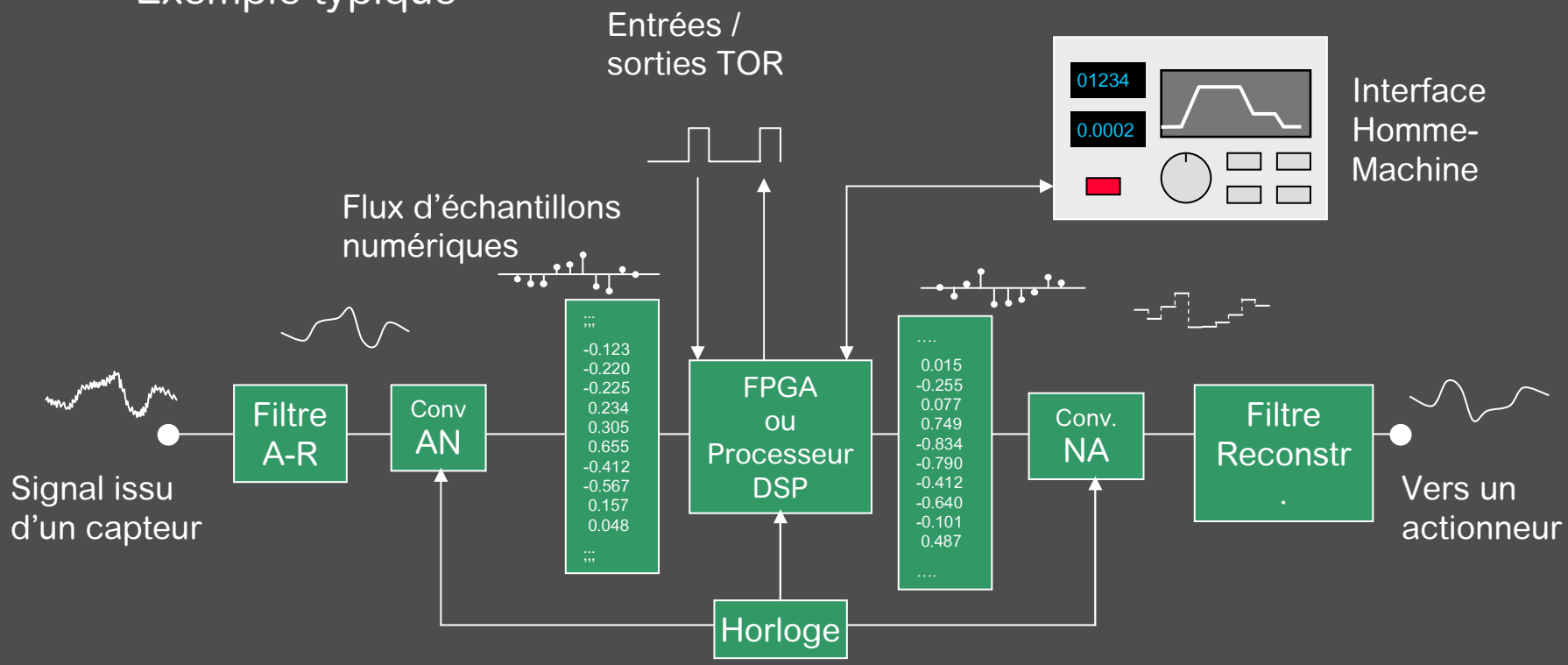
$y_k = b_0x_k + b_2x_{k-2} + a_1y_{k-1} + a_2y_{k-2}$

Le même filtre en technique numérique

- En analogique pb de précision des composants, dérives, réglages nécessaires
- En numérique: aucun composant qui dérive, aucun réglage, les seuls éléments précis sont la fréquence de l'horloge et les convertisseurs AN

1.1.2 Le Traitement Numérique du Signal en Temps Réel

Exemple typique



1.1.3 FPGA ou processeur DSP ? …

- FPGA = Field Programmable Gate Array

Réseau Logique Programmable

Sea of gates = Mer de portes interconnectables

On crée pour chaque fonction de traitement un bloc logique dédié

(par exemple: multiplication, convolution, transformée de Fourier)

Tous ces blocs logiques fonctionnent en même temps

- ➔ La rapidité ne dépend pas (ou peu) de la complexité du schéma
- ➔ La complexité est limitée par le nombre de portes disponibles
  par exemple avec 200 000 portes, on peut créer 10 multiplieurs 16 x 16 → 32
- ➔ Un FPGA se programme comme un schéma bloc où circulent les flux de données (langage VHDL ou schéma) .
- ➔ Il est mal adapté aux traitements conditionnels, aux changements de contexte
- ➔ La consommation est importante

1.1.3 FPGA ou processeur DSP ? …

- DSP = Digital Signal Processor

Processeur Numérique de Signal

Microprocesseur spécialisé

Les DSP comportent des unités de traitement parallèles
mais les algorithmes restent séquentiels.

- ➔ Le temps de traitement augmente avec la complexité du schéma
- ➔ La complexité des algorithmes n'est pratiquement pas limitée
- ➔ Un DSP se programme habituellement en langage C ou en Assembleur
- ➔ Souplesse: traitements conditionnels, changements de contexte très faciles
- ➔ Consommation faible, dépendante de la puissance de calcul demandée

1.1.3 FPGA ou processeur DSP ?

Traitement en Temps Réel:	FPGA	DSP
Signaux radio, radar	x	
Traitement d'images	x	
Modems, codage	x	x
Traitements audio		x
Algorithmes complexes		x
Décisions, conditions		x
Nombres flottants		x

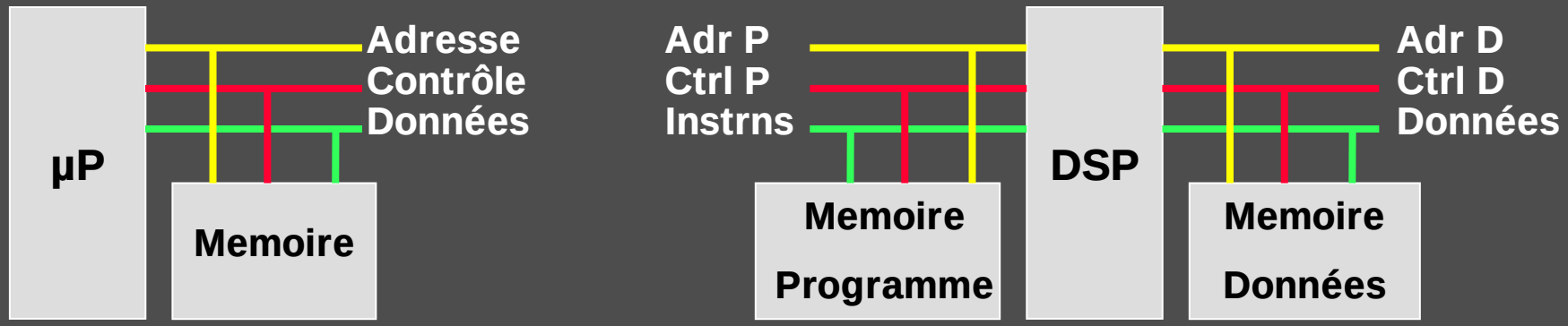
1.1.3 FPGA ou DSP ? (fin)

- En résumé:

- Rapidité: FPGA
- Souplesse, complexité: DSP
- Rapidité + complexité: FPGA + DSP

1.1.4 Processeur DSP: Architecture

- Architecture Harvard
  - Bus de Données et Bus d'Instructions distincts
  - ➔ Le processeur accède à une instruction et à 1 ou plusieurs données en un même cycle d'horloge



Von Neumann

Harvard

1.1.4 Processeur DSP: le pipeline

FETCH	Instr 1	Instr2	Instr3	Instr4
DECODE		Instr1	Instr2	Instr3
READ			Instr1	Instr2
EXECUTE				Instr1

- Il peut atteindre jusqu'à 15 étages pour certains processeurs
- Il est transparent pour certains DSP (DSP56xxx), il doit être pris en compte pour d'autres (TMS320C6xxx)
- On évite les instructions qui rompent le pipeline
- Lorsque l'on veut accéder au résultat d'une instruction non terminée, il y a insertion automatique de cycles morts

1.1.4 Processeur DSP: Le parallélisme

- Dans un même processeur, plusieurs unités arithmétiques et logiques fonctionnent en parallèle

// Exemple: code assembleur STARCORE SC140

```
[
macr        -d9,d5,d0
macr        d9,d4,d1
macr        -d11,d7,d2
macr        d11,d6,d3
move.4f     (r8)+n0,d12:d13:d14:d15
move.4f     (r9)+n0,d4:d5:d6:d7
]
```

// 4 multiplications – accumulations, 8 copies registre à mémoire,
// Durée: **1 cycle d'horloge**, soit **1ns** à 1GHz !

- Dans une même puce, on peut faire coexister plusieurs processeurs.
  - Ex. MSC8126 = 4 STARCORE dans une puce
  - Ex. extrême: PicoChip = 344 processeurs dans une puce !

1.1.4 Processeur DSP: Autres particularités

- Multiplication en 1 seul cycle
- Multiplication Accumulation (MAC) en 1 cycle
- Logique de saturation (virgule fixe)
- Boucle DO sans perte de temps
- Exécution conditionnelle
- Modes d'adressage spéciaux:
  - Adressage modulo m
  - Adressage en miroir (FFT)
- Transferts par DMA
- Ports série synchrones rapides

1.1.5 Tour d'horizon des fabricants de DSP …

- Texas Instruments TMS320Cxxx

C24-C28	Fixe	16 / 32b	40-150 MHz MIPS	µ Contrôleur hybride C28: Arithmétique 16.16
C54-C55	Fixe	16b	160-300MHz	SIMD 16+16 C55: 2 unités MAC
C64	Fixe	8-16b	1GHz	VLIW, 8 unités dont 4 multiplieurs parallèles
C67	Flottant	32b	300MHz	8 instructions RISC par cycle d'horloge

1.1.5 Tour d'horizon des fabricants de DSP …

- Analog Devices ADSP-xxxxx

218-219	Fixe	16b	80-160 MHz MIPS	Famille nombreuse Instructions 24b
2116-2113 Sharc	Fixe et Flottant	32/40b	200-400MHz	SIMD Instr. 48b
BF5xx Blackfin	Fixe	8-16b	750MHz	Faible consommation
TS20x Tiger Sharc	Fixe et Flottant	8-16-32-40b	600MHz	8 instructions RISC par cycle d'horloge

1.1.5 Tour d'horizon des fabricants de DSP …

- Freescale (Motorola)

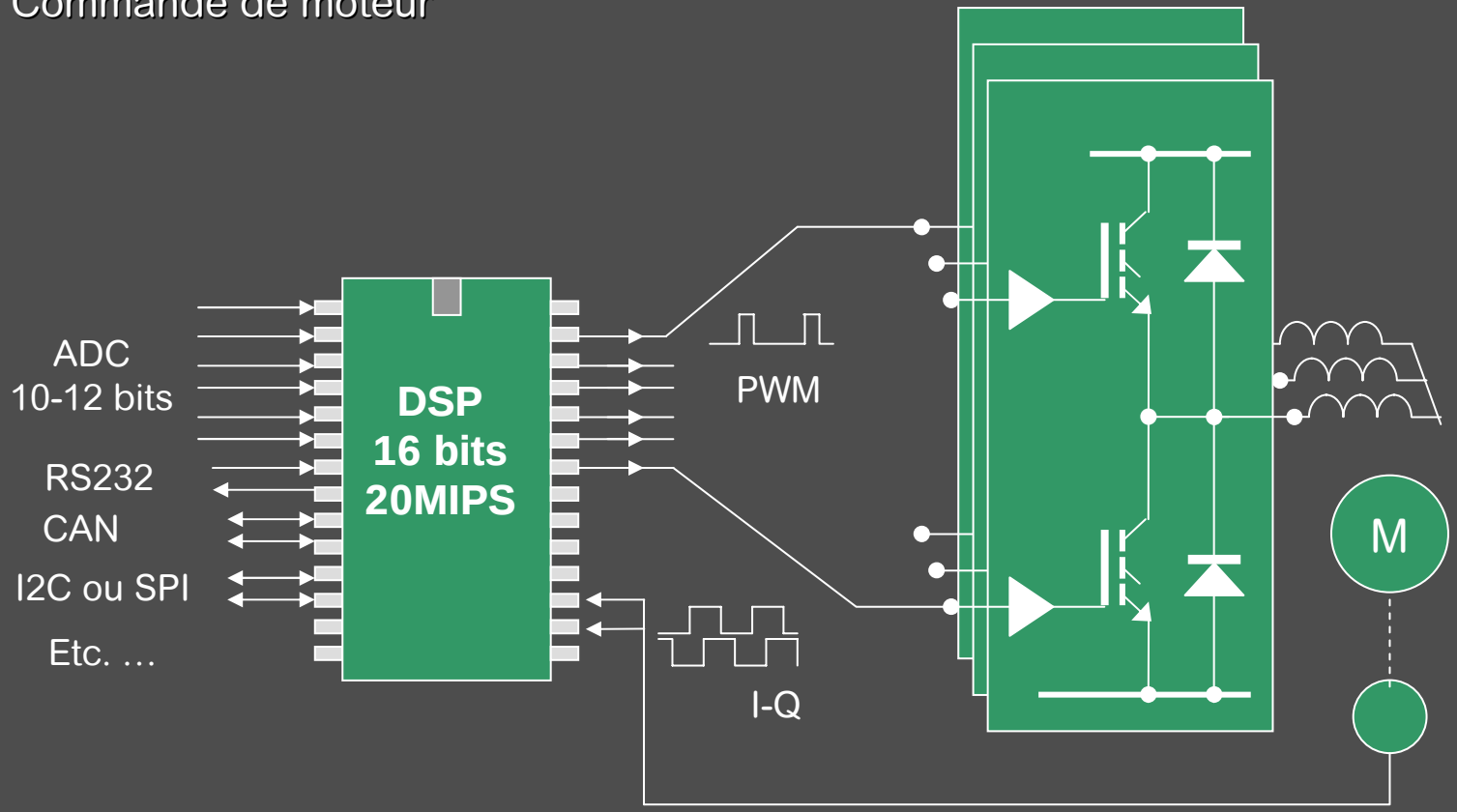
DSP563xx	Fixe	24b	275 MHz 550MMACs	Applications Audio EFCOP = copro. RIF
DSP56F8xx DSP56F8xxx	Fixe	16b	32-60 MMACS	µContrôleurs hybrides Automobile, ménager Contrôle moteurs
MSC8144 (SC140)	Fixe	8-16-32- 40b	1GHz x 4 cœurs	8 instructions RISC par cycle d'horloge
MRC6011			21 GMACs 16 processeurs	« Reconfigurable compute fabric » Stations de base 3G

1.1.5 Tour d'horizon des fabricants de DSP (fin)

Microchip DSPIC3xF	Fixe	16b	40MHz	Microcontrôleur hybride faible coût
NXP(Philips) PNX4103	Fixe Flottant	8-16-32b	350MHz	Processeur multimédia
PicoChip PC102	Fixe	16b	160MHz 197 GIPs	Massivement parallèle: 344 processeurs !
Renesas SH77xx	Fixe	16-32b	240MHz	Processeurs hybrides µP-DSP

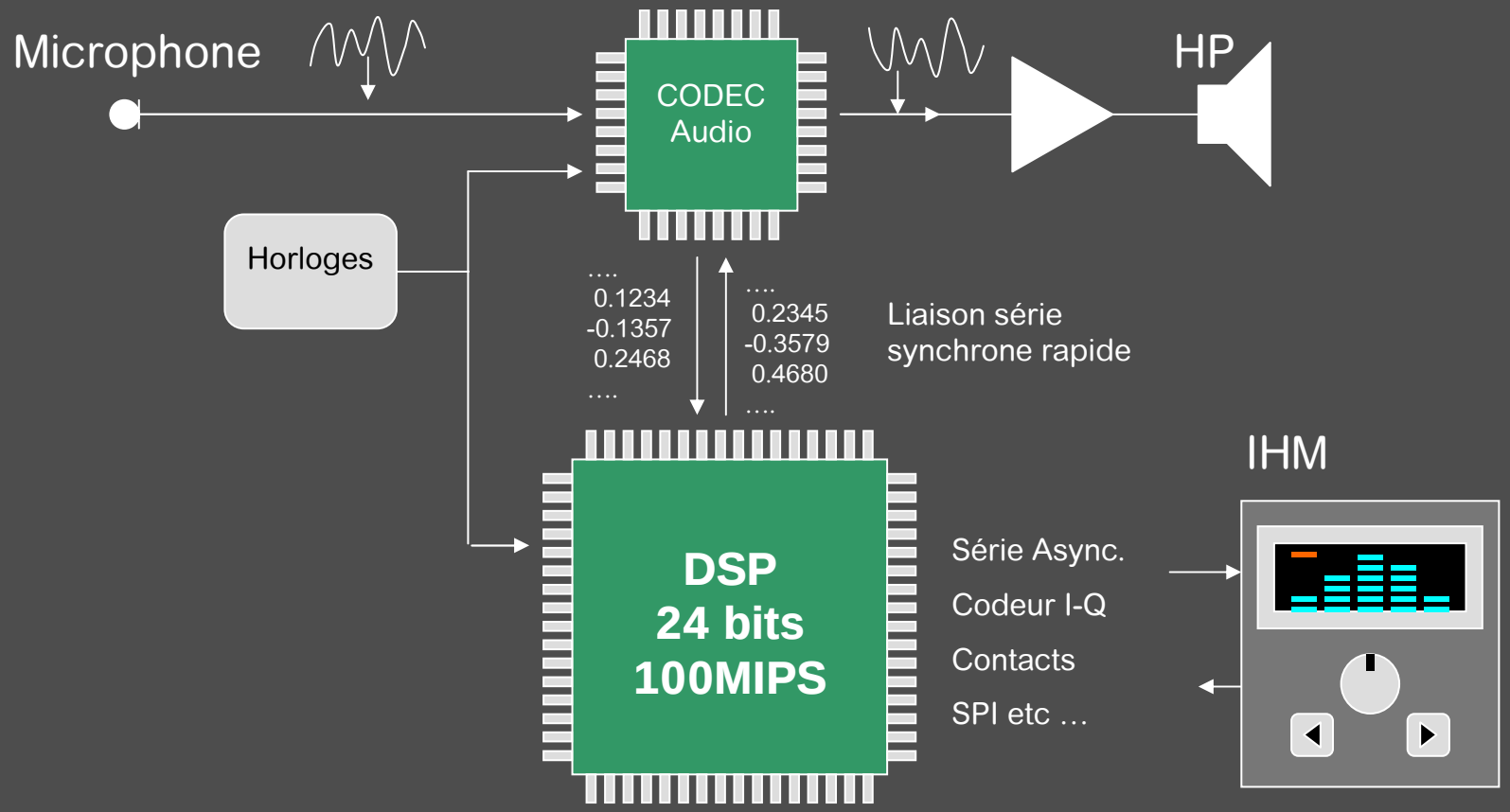
1.1.6 Exemple DSP 20 MIPS

Commande de moteur



1.1.6 Exemple DSP 100 MIPS:

Égaliseur audio à phase linéaire



1.1.6 Exemple DSP 1000 MIPS

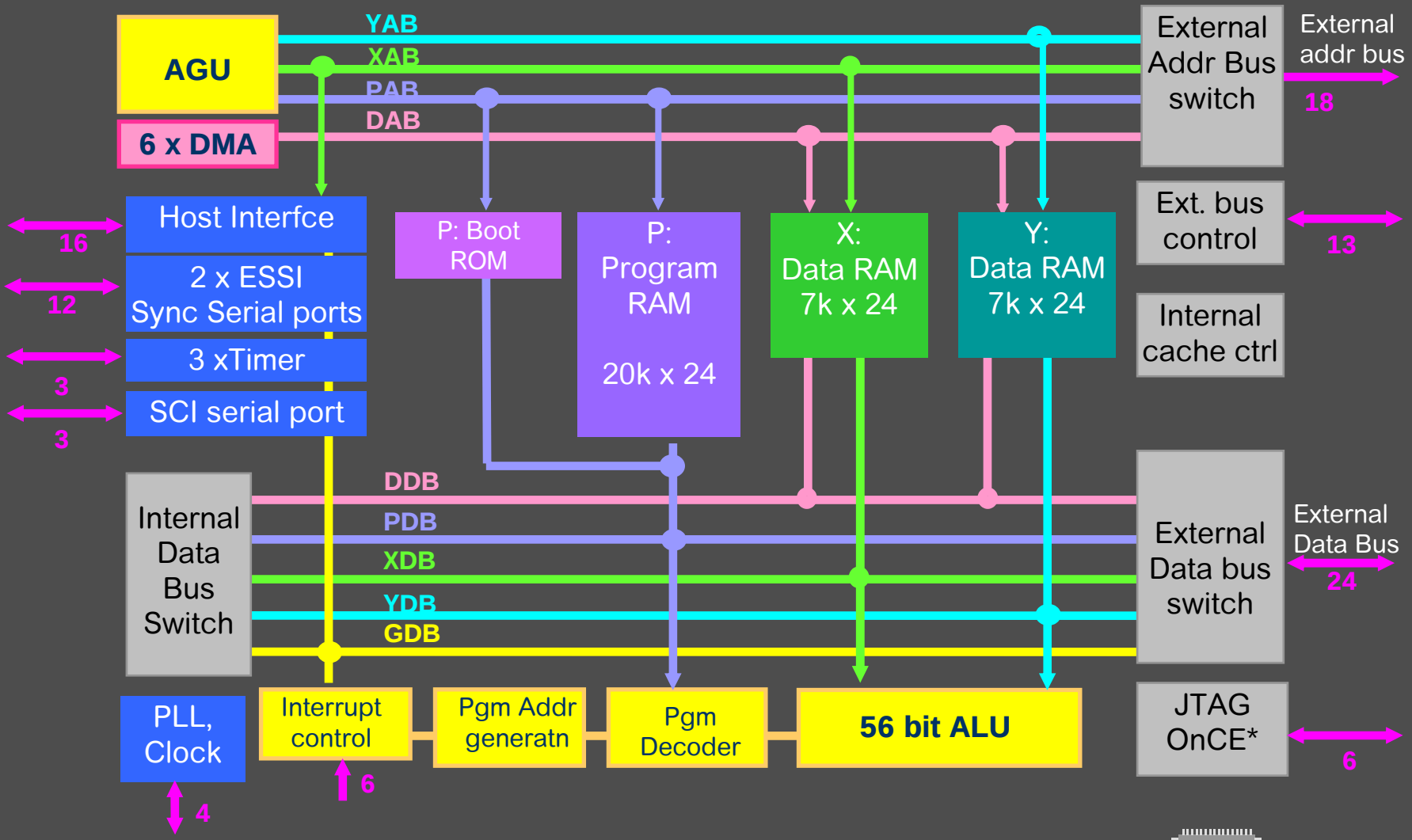
Compression d'image en TR

Micro caméra pour télésurveillance

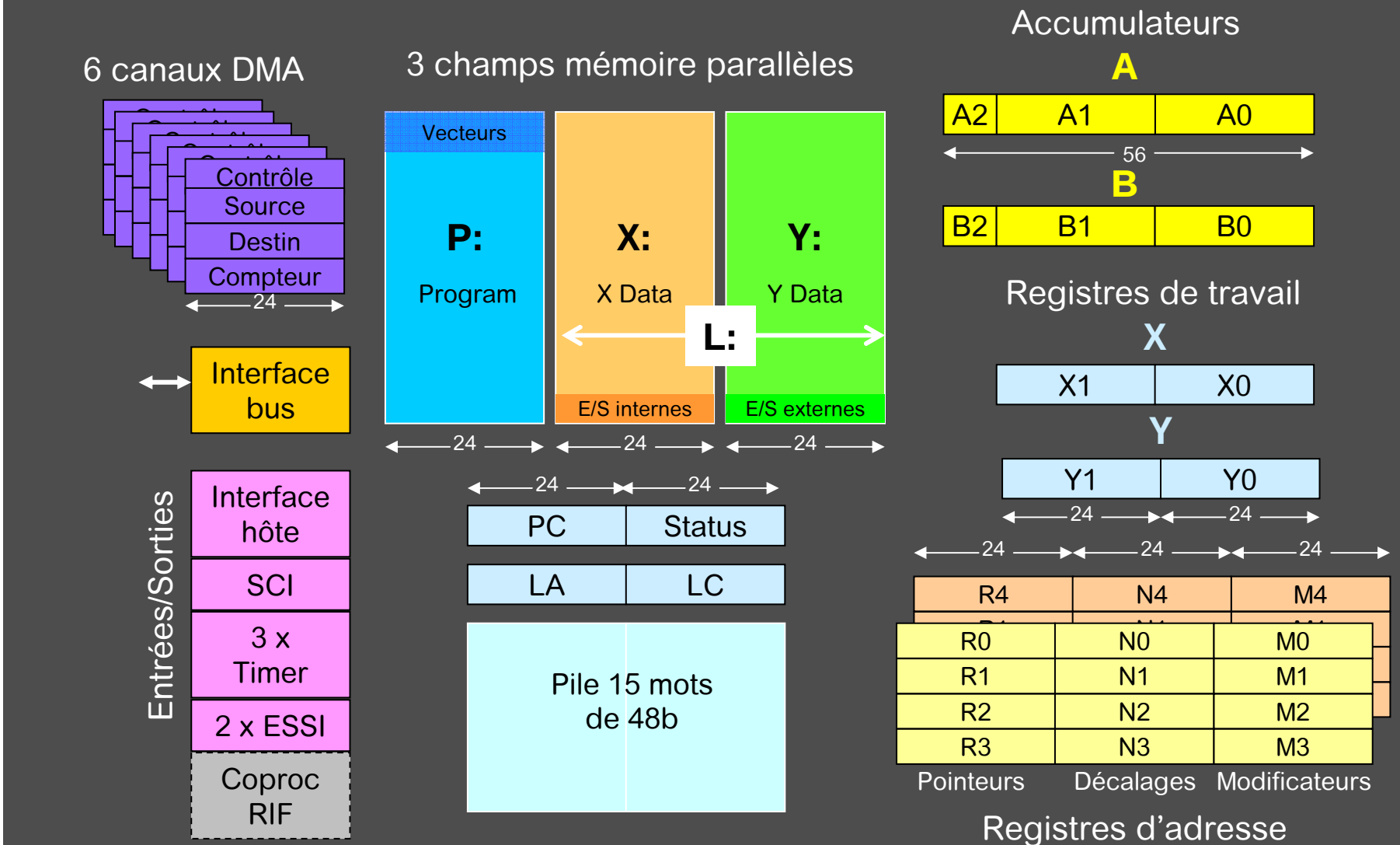
(Source: CYNOVE)

1.2 La famille de processeurs DSP563xx de Freescale

1.2.1 Architecture interne (DSP56309)

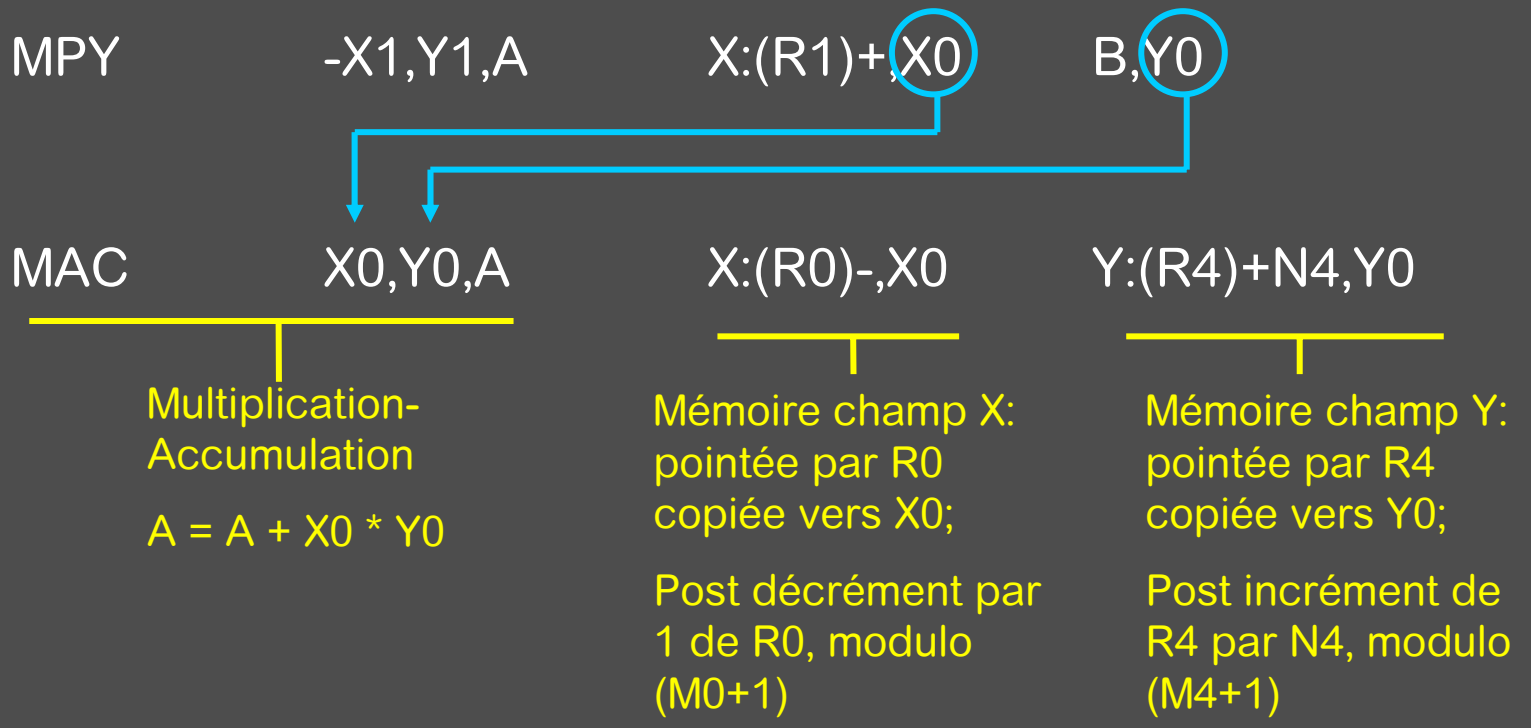


1.2.2 Modèle de programmation



1.2.3 Programmation des processeurs DSP56300 …

Les instructions parallèles



Chaque instruction de ce type s'exécute en 1 cycle d'horloge (10ns)

1.2.3 Programmation des processeurs DSP56300

Arithmétique virgule fixe fractionnaire

Toutes les variables de type réel évoluent dans l'intervalle [-1.. +1[

Une grandeur physique **x** est représentée par le nombre $\mathbf{x} / |x|_{max}$

On peut travailler à différentes précisions, les MSB restant inchangés

Mot mémoire ou registre 24 bits [-1..+1[dynamique 146dB

Mot mémoire ou registre 48 bits [-1..+1[dynamique 290dB

Accumulateur 56 bits [-256 .. +256[dynamique 338dB

1.2.3 Programmation des processeurs DSP56300
Machines à virgule fixe: débordement et saturation

- Si |accu| > 1

- move a,<destination> → saturation (propre aux DSP)
- move a1,<destination> → débordement (µP classique)

```
                    var     ad1
                    var     da1
                    var     da2

loop                ada     1e5
                    move    x:ad1,a
                    asl     #2,a,a          ; -4 <= A <+4
                    move    a,x:da1         ; saturation
                    move    a1,x:da2        ; débordement
                    jmp     loop
```

1.2.3 Programmation des processeurs DSP56300

Instructions arithmétiques …

Mettent en œuvre la totalité de l'accumulateur (56 bits)

ABS	A	Absolute value	A := |A|
ADC	X,A	Add long w. carry	A:=A+X+C (48b)
ADD	X0,A	Add	A:= A+X0 (24b)
ADDL	X0,A	Shift left and add	A:=2A+X0
ADDR	X0,A	Shift right and add	A:=A/2 + X0
ASL	A	Arithmetic shift left	A:=2A
ASL	#5,A,B	ASL multi-bit	B:=25A
ASR	A	Arithmetic shift right	A:=A/2
ASR	#5,A,B	ASR multi-bit	B:=2-5A
CLR	A	Clear accumulator	A:=0
CMP	X0,A	Compare	(A-X0) ?
CMPM	X0,A	Compare magnitude	(|A|-|x0|) ?
CMPU	X0,A	Compare unsigned	(A-X0) ?
DEC	A	Decrement accumulator	A:= A-1
DIV	X0,A	Divide iteration (one bit at a time)	
DMAC	X0,Y0,A	Double mac	A:=A.2-24 + x0.y0
INC	A	Increment accu	A:=A+1

1.2.3 Programmation des processeurs DSP56300
Instructions arithmétiques

MAC	X0,Y0,A	Multiply-accumulate	A:=A+x0.y0
MACR	X0,Y0,A	Multiply accumulate and round	
MAX	A,B	if A > B then B:=A	
MAXM	A,B	if |A| > |B| then B:=A	
MPY	X0,Y0,A	Multiply	A:=X0.Y0
MPYR	X0,Y0,A	Multiply and round	
NEG	A	Negate accumulator	A:= -A
NORM	R0,A	Normalization	
NORMF	B1,A	Fast normalization	
RND	A	Round accumulator	
SBC	X,A	Subtract long w. carry	A:=A-X-C
SUB	X0,A	Subtract from accumulator	A:=A-X0
SUBL	X0,A	Shift left and subtract	A:=2A-X0
SUBR	X0,A	Shift right and subtract	A:=A/2-X0
Tcc	X0,A R0,R1	Transfer on condition <cc> true	
TST	A	Test accumulator, update CCR	

1.2.3 Programmation des processeurs DSP56300

Instructions logiques

Modifient la partie A1 de l'accumulateur

AND	X0,A	Logical AND	A1 := A1 & X0
BCHG	#0,X:$12	Bit test and toggle	
BCLR	#0,X:$12	Bit test and clear	
BSET	#0,X:$12	Bit test and set	
BTST	#0,X:$12	Bit test	
CLR	A	Clear accumulator	
EOR	X0,A	Exclusive OR	A1:= A1 xor X0
LSL	A	Logic shift left	A1:= 2A1
LSL	#10,a	LSL multi-bit	A1:= A1<<10
LSR	A	Logic shift right	A1:= ½ A1
LSR	#10,a	LSR multi-bit	A1:= A1>>10
NOT	A	Logical complement	A1:= A1\
OR	X0,A	Logical OR	A1:= A1 v X0
ROL	A	Rotate A1 left	
ROR	A	Rotate A1 right	

1.2.3 Programmation des processeurs DSP56300
Instructions de transfert de données

Ne modifient pas le registre d'état

Si la source est un accumulateur: saturation automatique / calibrage

Si la destination est un accumulateur: extension signée à 56 bits

LUA	(R0)+N0,R1	Load upd addr R1:=R0+N0
LRA	label,R0	Load PC-rel addr R0:=Label
MOVE	sce,dest	Copy source to destination
MOVEC	#0,SP	Control register move
MOVEM	X0,P:(R0)+	Program memory move
MOVEP	Y:$FFFFFF,X:0	Peripheral I/O move
TFR	X0,A	Register to accu transfer

1.2.3 Programmation des processeurs DSP56300

Instructions de contrôle de l'exécution du programme …

Bcc	label	Branch (relative) if condition cc is true
BRA	label	Branch always (PC relative)
BRCLR	#3,Y:$FFFF,label	Branch if bit clear
BRSET	#3,Y:$FFFF,label	Branch if bit set
BRKcc		If cc true, exit current DO loop
BScc	label	Branch to subroutine if cc is true
BSR	label	Branch to subroutine
DEBUG		Enter debug mode
DEBUGcc		Enter debug mode if cc is true
DO	X0,label	Start DO loop up to absolute label
DOR	#n,label	Start DO loop up to PC relative label
DOFOREVER	label	Start infinite DO loop, absolute
DORFOREVER	label	Start infinite DO loop, PC relative
ENDDO		Abort current DO loop
IFcc (in parallel field)		Conditionally execute instruction
Jcc	label	Absolute jump if condition cc is true
JCLR	#1,X:0,label	Absolute jump if bit clear
JMP	label	Absolute jump
JScc	label	Absolute jump to SR if cc is true

1.2.3 Programmation des processeurs DSP56300
Instructions de contrôle de l'exécution du programme

JSCLR	#1,X:0,label	Abs jump to subroutine if bit clear
JSET	#1,X:0,label	Absolute jump if bit set
JSR	label	Jump to subroutine
JSSET	#1,X:0,lab	Absolute jump to subroutine if bit set
NOP		No operation
PLOCK	|	
PUNLOCK	|	
PLOCKR	|	
PUNLOCKR	|	Program cache management
PFREE	|	
PFLUSH	|	
PFLUSHUN	|	
REP	#n	Repeat next instruction
RESET		Initialize peripherals
RTI		Return from interrupt
RTS		Return from subroutine
STOP		Low power standby, clocks halted
TRAP		Software interrupt
TRAPcc		Trap if condition cc true
WAIT		Wait for interrupt (low power standby)

1.2.3 Programmation des processeurs DSP56300
Modes d'adressage …

Addressing modes	Assembler syntax
Register direct	
Accumulators	`A, A2, A1, A0,A10,AB, B, B2, B1, B0, B10`
Input registers	`X, X1, X0, Y, Y1, Y0`
Control registers	`PC,SR,LA,LC,SSH,SSL,SP,SC,SZ,OMR,VBA`
Address register	`Rn, Nn, Mn  n = 0 .. 7`
Address register indirect	
No update	`(Rn)`
Post increment by 1	`(Rn)+`
Post decrement by 1	`(Rn)-`
Post increment by offset Nn	`(Rn)+Nn`
Postdecrement by offset Nn	`(Rn)-Nn`
Indexed by offset Nn	`(Rn+Nn)`
Predecrement by 1	`-(Rn)`
Short / long displacement	`(Rn+displ)`

1.2.3 Programmation des processeurs DSP56300
Modes d'adressage

PC relative	
Short / long displacement, PC relative	`(PC+displ)`
Address register	`(PC+Rn)`
Special	
Short immediate	`#[<]dd`
Long immediate	`#>dddddd`
Absolute address	`{X|Y|L|P}:aaaaaa`
Absolute short address	`[<]{X|Y|L}:aa`
Short jump address	`[<]aaa`
I / O short address	`[<<]{X|Y}:aa`
Implicit	

1.2.3 Programmation des processeurs DSP56300
Syntaxe d'une instruction Assembleur DSP563xx

[Label] Operator {Operands} [Parall. mv1] [Parall. mv2] [;Comments]

Exemple:

```
conv    mac     x0,y0,A     x:(r0)+,x0     y:(r4)-,y0     ; convolution
```

1.2.3 Programmation des processeurs DSP56300
La fonction « calculette en ligne » du compilateur ASM …

- Pour évoquer une constante, on peut donner sa valeur en décimal, par exemple pour créer une mémoire initialisée:

```
org     x:constantes    ; emplacement
….
dc      0.12345                 ; valeur à stocker
```

- Si la valeur résulte d'un calcul, l'assembleur peut réaliser ce calcul en ligne:

Ex

```
move    #(fin_table-table),N0  ; N0= longueur table

dc      @sin(pi/4)              ; stocker valeur 0.707
```

1.2.3 Programmation des processeurs DSP56300
La fonction « calculette en ligne » du compilateur ASM …

1 Math functions

@ABS - Absolute value
@ACS - Arc cosine
@ASN - Arc sine
@AT2 - Arc tangent
@ATN - Arc tangent
@CEL - Ceiling function
@COH - Hyperbolic cosine
@COS - Cosine
@FLR - Floor function
@L10 - Log base 10
@LOG - Natural logarithm
@MAX - Maximum value
@MIN - Minimum value
@POW - Raise to a power
@RND - Random value
@SGN - Return sign
@SIN - Sine
@SNH - Hyperbolic sine
@SQT - Square root
@TAN - Tangent
@TNH - Hyperbolic tangent
@XPN - Exponential function

2 Conversion Functions

@CVF - Convert integer to floating point
@CVI - Convert floating point to integer
@CVS - Convert memory space
@FLD - Shift and mask operation
@FRC - Convert floating point to fractional
@LFR - Convert floating point to long fraction
@LNG - Concatenate to double word
@LUN - Convert long fractional to float
@RVB - Reverse bits in field
@UNF - Convert fractional to floating point

3 String Functions

@LEN - Length of string
@POS - Position of substring in string
@SCP - Compare strings

4 Macro Functions

@ARG - Macro argument function
@CNT - Macro argument count
@MAC - Macro definition function
@MXP - Macro expansion function

5 Assembler Mode Functions

@CCC - Cumulative cycle count
@CHK - Current instruction/data checksum
@CTR - Location counter type
@DEF - Symbol definition function
@EXP - Expression check
@INT - Integer check
@LCV - Location counter value
@LST - LIST directive flag value
@MSP - Memory space
@REL - Relative mode function

1.2.3 Programmation des processeurs DSP56300
La fonction « calculette en ligne » du compilateur ASM

Exemple: créer une table de sinus (1 période, 100 points)

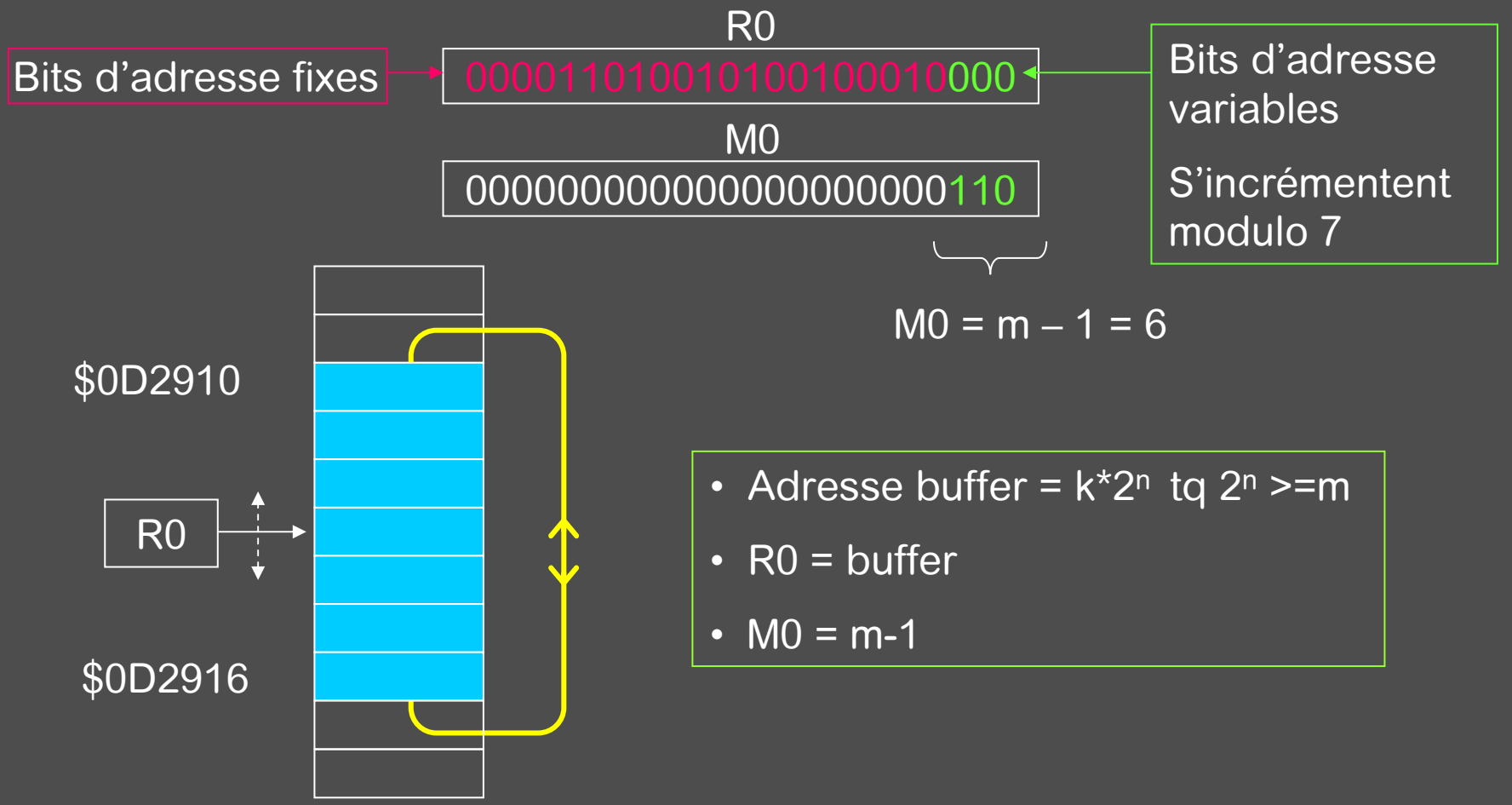
```
incr    equ     pi/50.0
angle   set     0.0

        org     y:0
sinus
        dup     100
        dc      @sin(angle)
angle   set     angle+incr
        endm
```

1.2.3 Programmation des processeurs DSP56300
Unité de Génération d'Adresses (AGU)

R0	N0	M0
R1	N1	M1
R2	N2	M2
R3	N3	M3
R4	N4	M4
R5	N5	M5
R6	N6	M6
R7	N7	M7

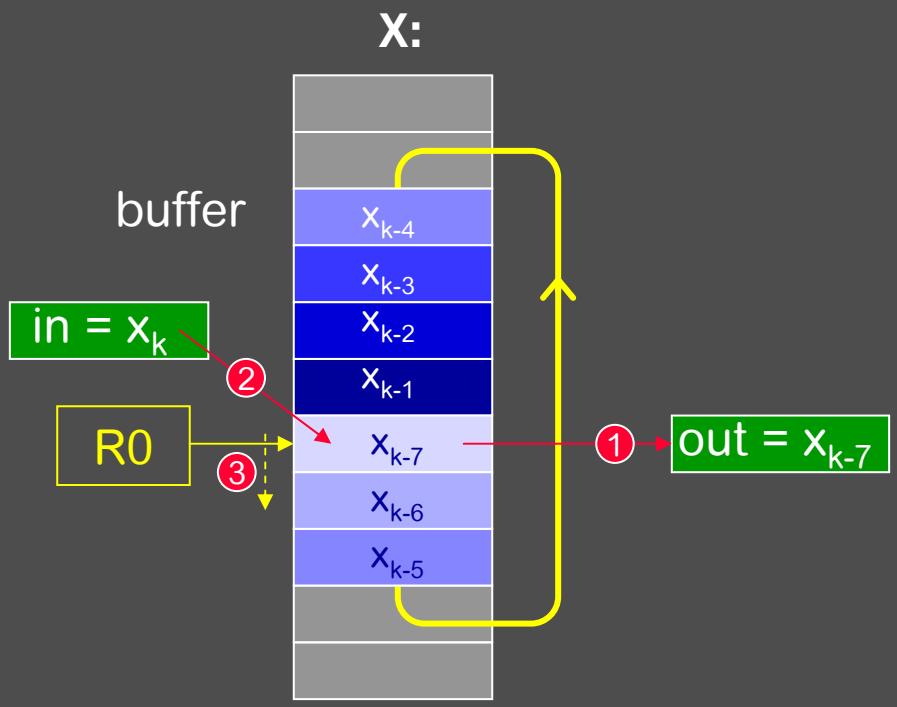
- 8 x 3 registres de 24 bits:
- R0 .. R7 = pointeurs
  - X:(R0) = le mot d'adresse R0 dans le champ X:
- M0 .. M7 = modificateurs
  - M0 = 0 => R0 s'incrémente en binaire miroir
  - M0 = 1 .. 7FFF => R0 s'incrémente modulo (M0+1)
  - M0 = 8001, 8003, … => repliement multiple modulo 2, 4, …
  - M0 = FFFFFF => R0 s'incrémente linéairement (incrémentation normale)
- N0 .. N7 = offsets
  - X:(R0+N0) = mot d'adresse R0 + N0 dans le champ X:
  - (R0)+N0 = post-incrémentation de R0 par N0
  - (R0)-N0 = post-décrémentation de R0 par N0

1.2.3 Programmation des processeurs DSP56300
L'adressage modulo m Exemple: m=7



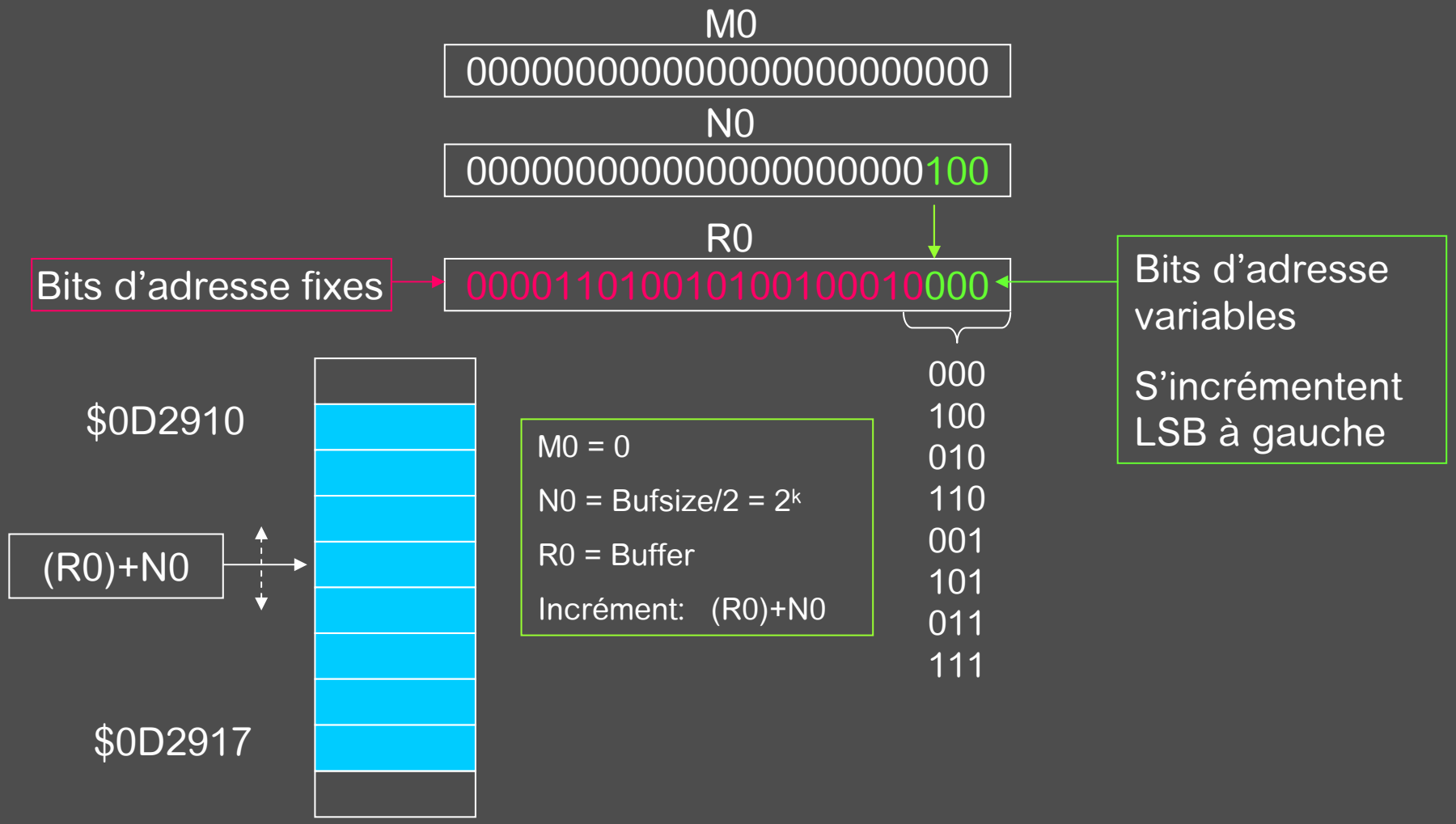
- Adresse buffer = k*2^n tq 2^n >=m
- R0 = buffer
- M0 = m-1

1.2.3 Programmation des processeurs DSP56300
Application de l'adressage modulo: La ligne à retard




```
            org     x:
buffer      dsm     7

            org     p:
Init        move    #buffer,r0
            move    #6,m0

            …
delay       move    x:(r0),a
            move    a,x:out
            move    x:in,x0
            move    x0,x:(r0)+
```

1.2.3 Programmation des processeurs DSP56300
L'adressage en miroir Exemple: buffer 8 mots

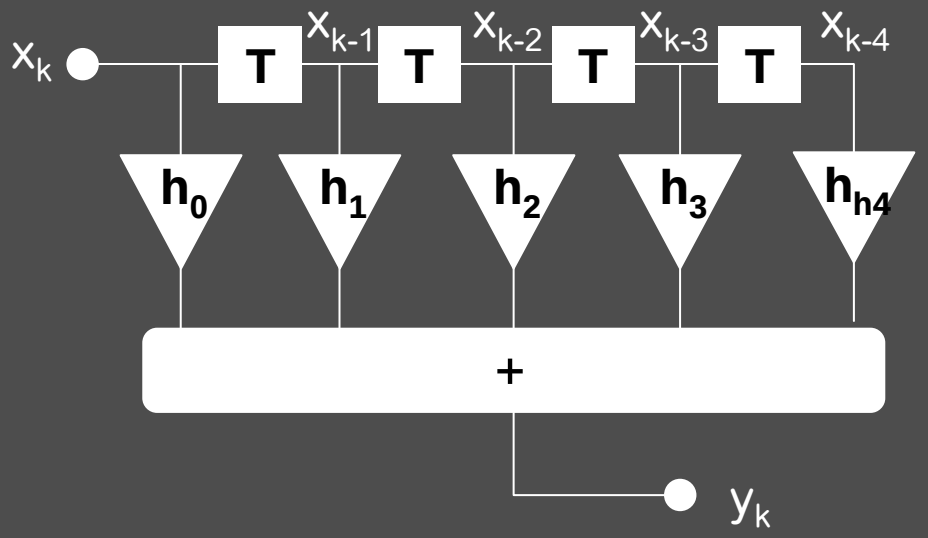


1.3 Genèse d'une plateforme de développement rapide pour DSP563xx

1.3.1 Comparaison Compilateur C ANSI ⇔ ASM natif
Exemple: Filtre RIF (Réponse Impulsionnelle Finie)

Sortie $\{y_k\}$ = produit de convolution du signal $\{x_k\}$ par la réponse impulsionnelle $\{h_n\}$

$$y_k = \sum_{n=0}^{N-1} x_{k-n} h_n$$



1.3.1 Comparaison Compilateur C ANSI ⇔ ASM natif
Exemple: Filtre RIF (Réponse Impulsionnelle Finie)

- Compilateur C (optimisation en taille)

```
_fract FiltreFIR(_fract x)
{
  int n;
  _fract y=0;
  x_buffer[ptr_x_buffer++] = x;
  ptr_x_buffer %= NB_COEF;

  for( n = (NB_COEF-1) ; n >= 0 ; n-- )
  {
    y += coef[n] * x_buffer[ptr_x_buffer++];
    ptr_x_buffer %= NB_COEF;
  }
  return(y);
}
```

Compilation →

```
FFiltreFIR:
                move        a,r3
                move        x:Fptr_x_buffer,r5
                move        #0,n3
                move        r5,n6
                move        #Fx_buffer,r6
                move        r3,x:(r6+n6)
                move        (r5)+
                move        r5,a
                move        #>100,x0
                jsr         Rmod_iii
                move        a,r5
                move        x0,n5
                move        #Fcoef+99,r6
                do          n5,L5
                move        x:(r6)-,y0
                move        #Fx_buffer,n5
                move        x:(r5+n5),y1
                move        n3,b
                macr        y0,y1,b
                move        b,n3
                move        (r5)+
                move        r5,a
                jsr         Rmod_iii
                move        a,r5
                nop
                nop
L5:             void        a, b, x0, y1, y0,
n5, r6
                move        n3,a
                move        r5,x:Fptr_x_buffer
                rts
```

Taille programme :	**59** mots
Temps d'exécution:	**2991** cycles

1.3.1 Comparaison Compilateur C ANSI ⇔ ASM natif
Exemple: Filtre RIF (Réponse Impulsionnelle Finie)

- Langage C (optimisation en vitesse)

```
FFiltreFIR:
                move            a,n1
                move            x:Fptr_x_buffer,b
                move            #0,r1
                move            b,n6
                move            #Fx_buffer,r6
                move            n1,x:(r6+n6)
                add             #1,b
                move            #>100,y0
                cmpm            y0,b            b,a
                jcs             L6
                abs             b
                extractu #$18018,b,b
                do              #25,L7
                div             y0,b
L7:             btst            #23,y0
                add             y0,b            ifcc
                sub             y0,b            ifcs
                lsr             b
                move            #0,b0
                btst            #23,a
                neg             b               ifcs.u
L6:             tst             b
                jclr            #23,a1,L8
                jeq             L8
                btst            #23,y0
                add             y0,b            ifcc
                sub             y0,b            ifcs
                tst             b
L8:             move            b,r5
                move            y0,n2
                move            #Fcoef+99,r6
                do              n2,L5
                move            x:(r6),y1
                move            #Fx_buffer,n5
                move            x:(r5+n5),x0
                move            r1,b
                macr            y1,x0,b
                move            b,r1
                move            (r5)+
                move            r5,b
                cmpm            y0,b            b,a
                jcs             L9
                abs             b
                extractu #$18018,b,b
                do              #25,L10
                div             y0,b
L10:            btst            #23,y0
                add             y0,b            ifcc
                sub             y0,b            ifcs
                lsr             b
                move            #0,b0
                btst            #23,a
                neg             b               ifcs.u
L9:             tst             b
                jclr            #23,a1,L11
                jeq             L11
                btst            #23,y0
                add             y0,b            ifcc
                sub             y0,b            ifcs
                tst             b
L11:            move            b,r5
                move            (r6)-
L5:             void            a, b, x0, y1, y0, n5,
r6
                move            r1,a
                move            r5,x:Fptr_x_buffer
                rts
```

Taille programme **72 mots**

Temps exécution **2486 cycles**

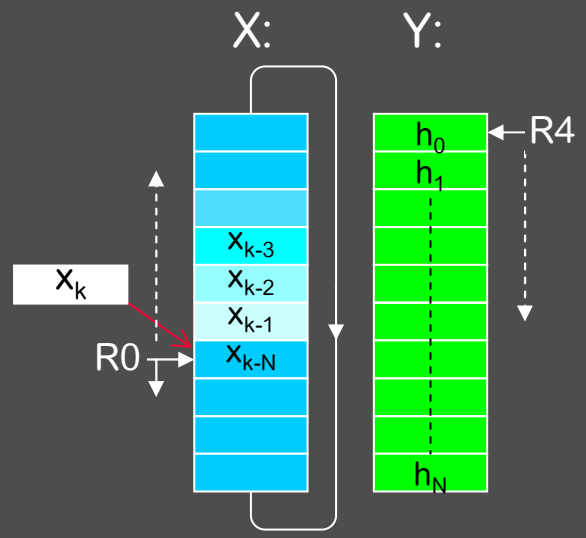
1.3.1 Comparaison Compilateur C ANSI ⇔ ASM natif
Exemple: Filtre RIF (Réponse Impulsionnelle Finie)

- Assembleur écrit et optimisé à la main

```
        move                x:ptrr0,r0
        move                x:ptrr4,r4
        move                x:modm0,m0
        move                m0,m4

        movep               y:ad1,x:(r0)
        clr     a           x:(r0)-,x0    y:(r4)+,y0

        do      m0          finboucle
        mac     x0,y0,a     x:(r0)-,x0    y:(r4)+,y0
finboucle
        macr    x0,y0,a     (r0)+

        move                a,x:resultat
        move                r0,x:ptrr0
```

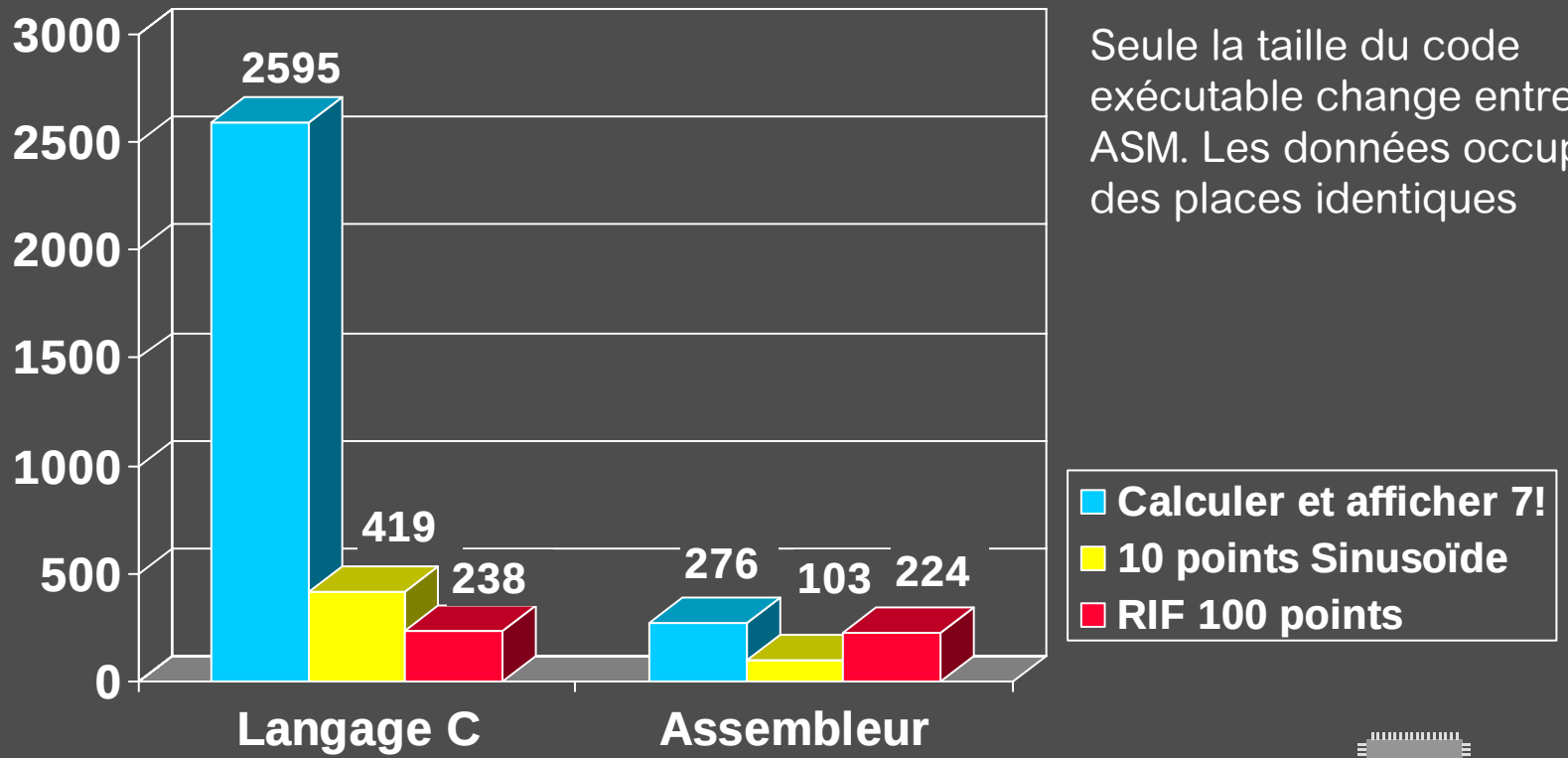
Taille programme **12 mots**

Temps exécution **118 cycles**



1.3.1 Petit benchmark C ⇔ Assembleur (DSP563xx)

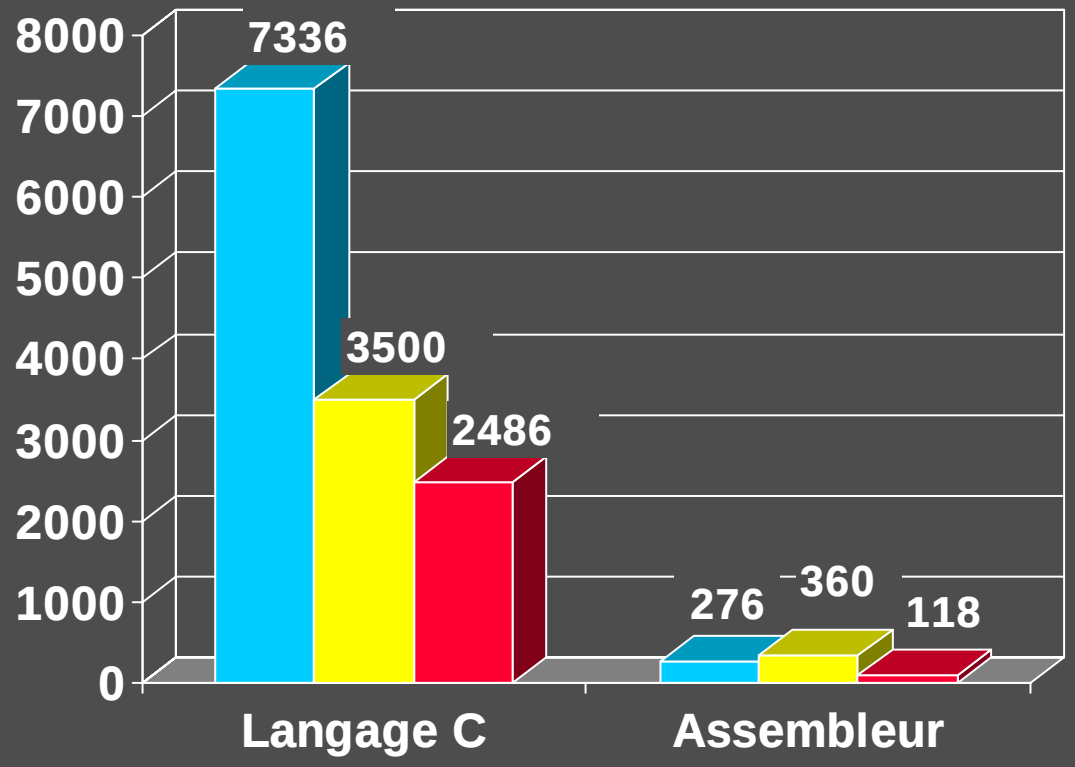
Taille du code (mots mémoire 24 bits)



Seule la taille du code exécutable change entre C et ASM. Les données occupent des places identiques

1.3.1 Petit benchmark C ⇔ Assembleur (DSP563xx)

Temps d'exécution (cycles machine)



La programmation en assembleur produit un code 10 fois plus rapide que le C !

1.3.1 Petit benchmark C ⇔ Assembleur (DSP563xx)
Discussion sur ces résultats …

- Le DSP56300 n'est pas optimisé pour les compilateurs C (le compilateur ne sait pas tirer parti du double Harvard)
- Le mauvais score du filtre FIR provient du fait que le compilateur ne sait pas utiliser les modificateurs m0..m7 pour l'adressage modulo
- L'affichage « Factorielle (7) = 5040 » est donné comme exemple par le constructeur du compilateur car le programme s'écrit en 3 lignes en utilisant la récursivité … mais l'utilisation de la récursivité est très chronophage !!!
- Le compilateur utilise la division dans ses algorithmes alors que les DSP sont optimisés pour les multiplications
- La majeure partie du temps perdu vient de l'enrobage inutile fait autour des blocs de traitement (appel de fonctions, copies de variables, passage de paramètres sur la pile, …)

1.3.1 Plaidoyer pour des outils DSP légers et simples d'utilisation … mais efficaces !

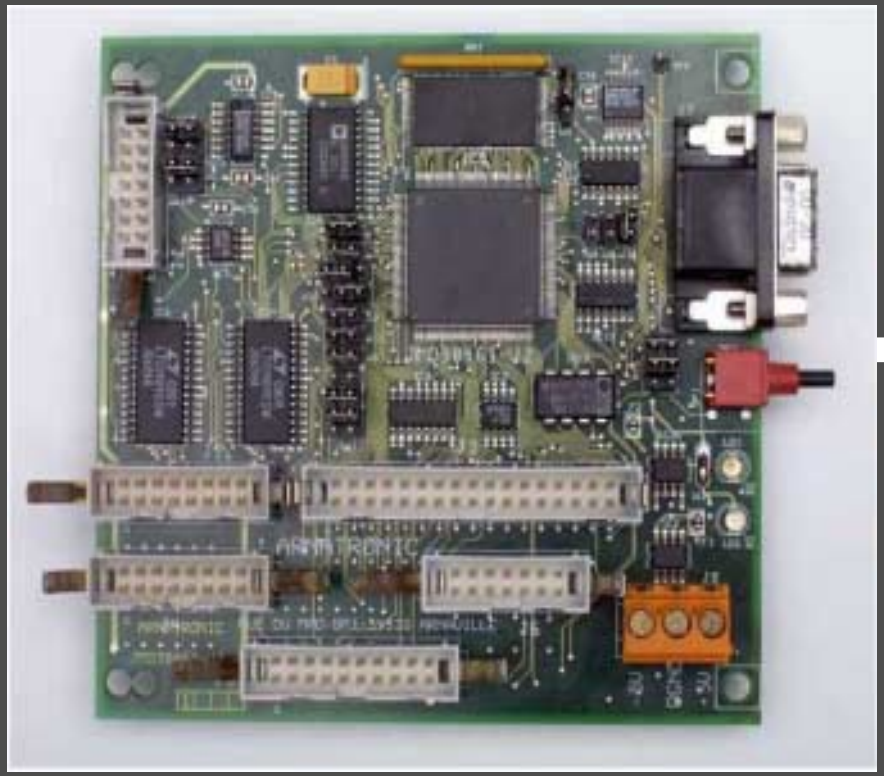
- Dans les cas suivants
  - PME non spécialisées qui intègrent un DSP dans leurs produits et veulent un résultat rapide
  - Enseignants qui ne disposent que de quelques heures de Travaux Pratiques pour montrer aux étudiants un concept en Théorie des Signaux ou en Transmissions …
  - Chercheurs qui veulent réaliser une maquette pour vérifier une théorie
- **fibula** Environnement de développement rapide
- Collaboration CRAN – IFETURA - ARNATRONIC :
  mu.psi une carte DSP industrielle (1998)
- Collaboration DIDALAB (2009):
  Carte DSP pédagogique

1.3.2 La cible matérielle: la carte **mu.psi** (1998)
Une carte DSP pour l'industrie

- DSP 24 - 48 bits 100 MIPS
- RAM interne pour chargements rapides (enseignement)
- Flash pour durcir les applications (industrie)
- Convertisseurs AN-NA 12 bits rapides (tc<1µs)
- Pas de filtre anti-repliement

Port analogique

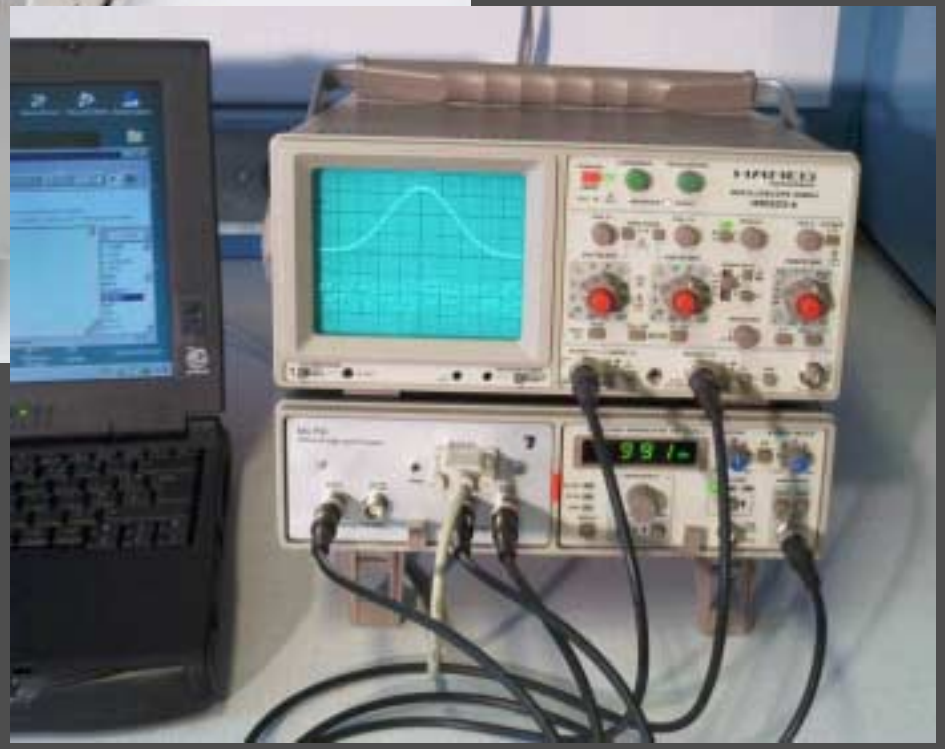
12 bits



RS232

RESET

Ports logiques Hôte parallèle,
2 ESSI, 3 Timer, SCI, JTAG



mu.psi Education Kit

1.3.2 Automne 2009 2 Nouvelles cartes DSP basées sur DSP5672x Processeur double cœur 2 x 250 MIPs SYMPHONY

Didalab:

Carte pédagogique

IFETURA Arnatronic:

Carte industrielle

Architecture double coeur

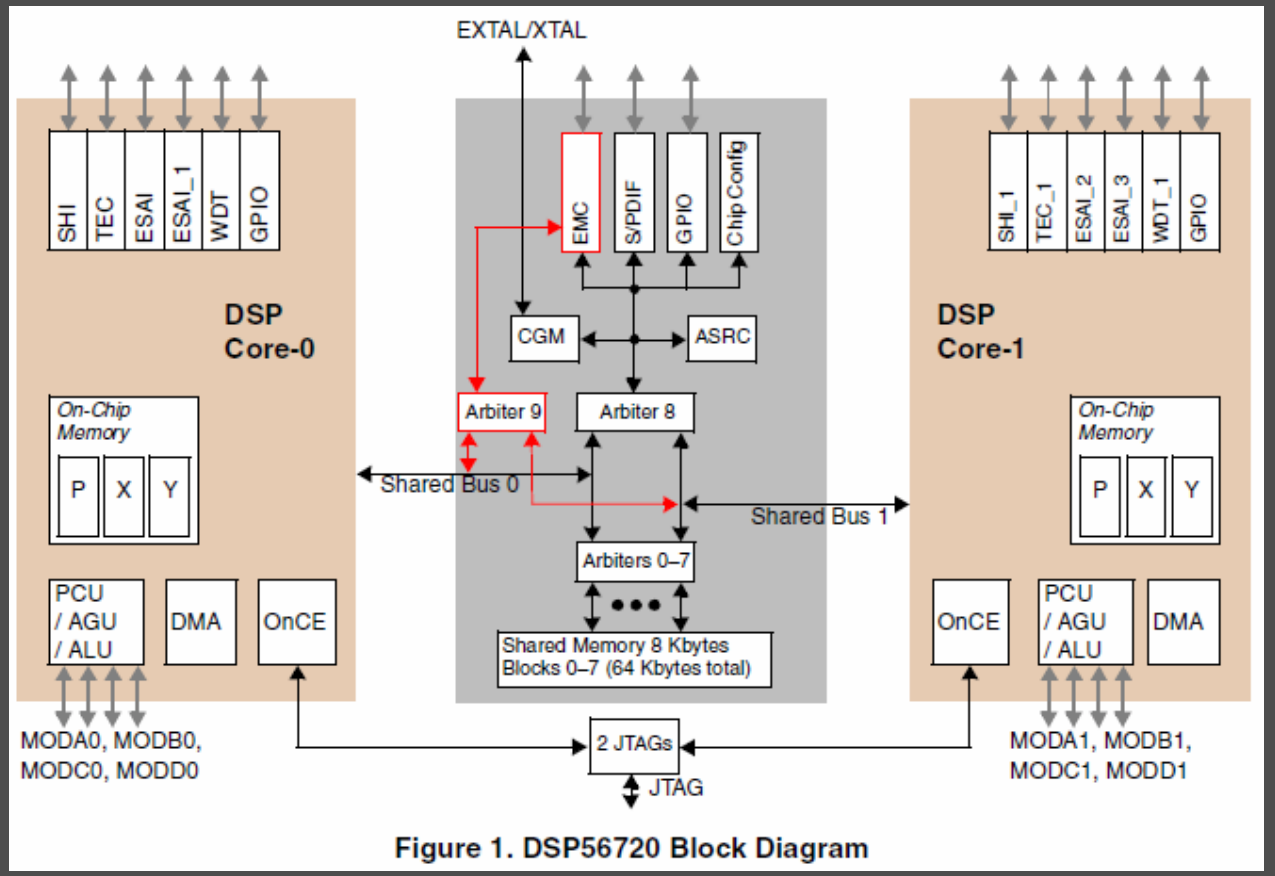


Figure 1. DSP56720 Block Diagram

1.3.3 Programmation par blocs fonctionnels

- Il est facile d'écrire quelques lignes de code assembleur
- Il est difficile de développer un grand projet en assembleur

- Idée:
  créer une librairie de blocs fonctionnels prêts à l'emploi.

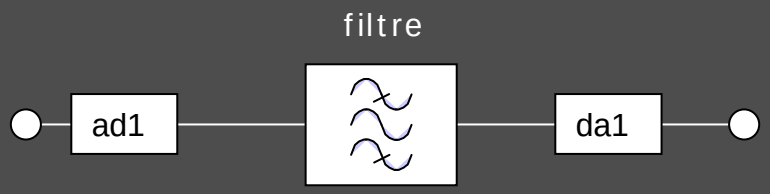
- Structure hiérarchique: un bloc est constitué de blocs
- La librairie contient un ensemble de fonctions atomiques
- L'utilisateur peut créer des blocs spécialisés à partir des blocs existants
- D'où le nom: **fibula**
  functional **i**nterconnected **b**locks **u**ser **la**nguage

1.3.4 Fibula textuel (2002): Un compilateur fondé sur une bibliothèque de macros écrites en assembleur.

- Blocs et connexions sont décrits sous forme de texte
- Les connexions définissent les variables communes à plusieurs blocs, et définissent leurs types.

Exemple:

Insérer un filtre passe-bande entre les bornes AD1 et DA1

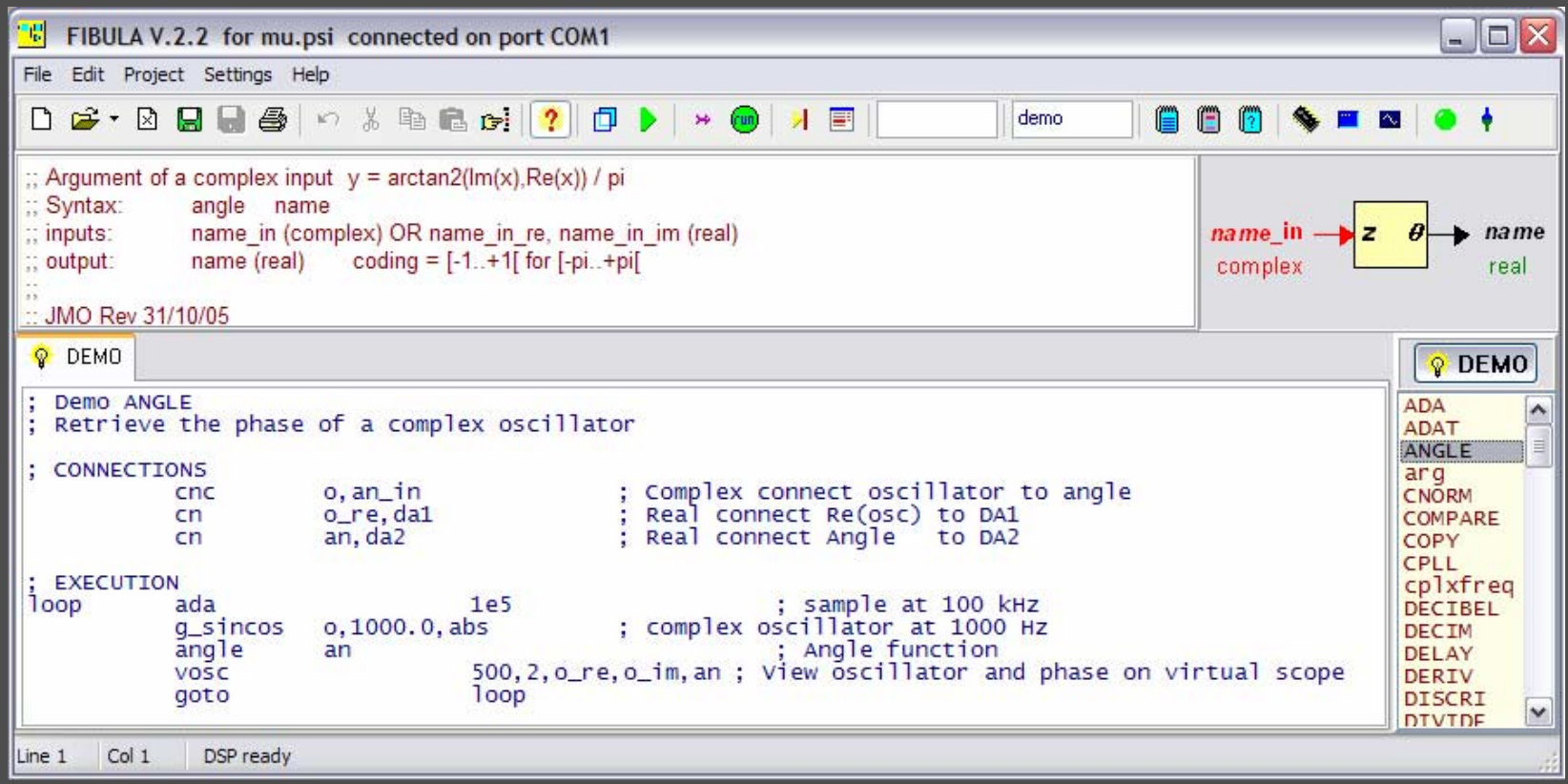


Filtre récursif passe-bande du 2nd ordre

F0=1000Hz, Q=100

```
        cn      ad1,filtre_in
        cn      filtre,da1
loop    ada     1E5
        iir2     filtre,bp,1000.,100,abs
        goto    loop
```

1.3.4 Fibula textuel (V.2.2 en 2005)
L' environnement de développement intégré



1.3.4 Fibula textuel:
Retour d'expérience (7ans d'utilisation)

- La prise en mains est assez rapide, mais le langage est trop tributaire de la syntaxe ASM, source de nombreuses erreurs.
- Les démonstrations (par exemple Th. de Shannon) sont bien comprises et retenues.
- Les exercices consistant à trouver les paramètres d'un programme pour qu'il fonctionne se ramènent souvent à un gradient par approximations successives … !

- Les exercices consistant à créer une application nouvelle (par exemple créer un démodulateur FSK) sont toujours décevants, car:
  - Le logiciel n'offre pas d'emblée une vision hiérarchique du problème à résoudre
  - Les étudiants n'ont pas l'habitude de raisonner à l'aide de schémas (ils appliquent des formules).

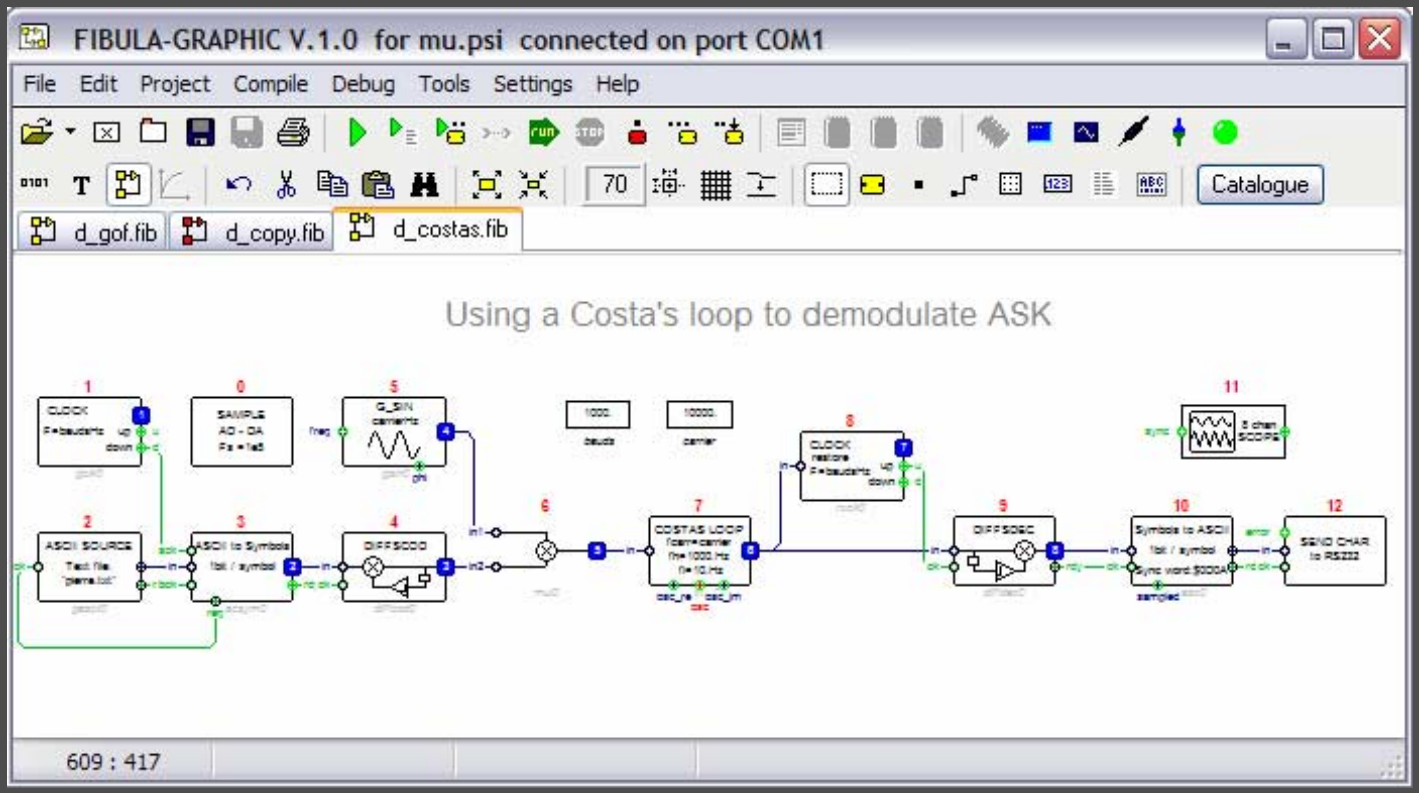
- Les étudiants qui s'investissent dans un projet industriel ou ludique utilisent correctement et efficacement la plate-forme, sauf lorsqu'il s'agit d'écrire le code en ASM d'un nouveau bloc fonctionnel.

1.3.5 **fibula-g** Version graphique

- Réclamé par les clients utilisant Simulink ™ ou Labview ™
- Entièrement graphique, mais:

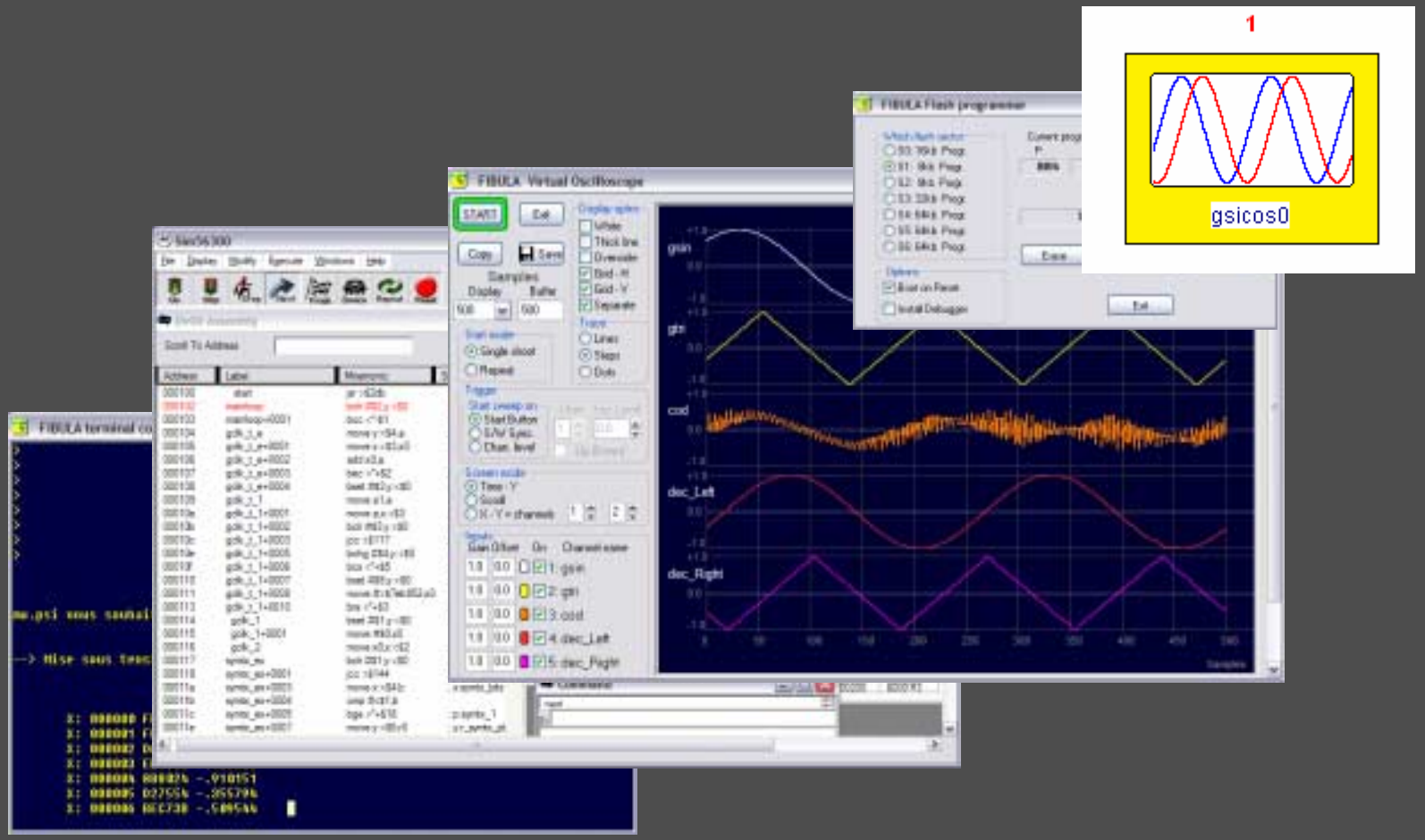
- Mode textuel possible
- Outils identiques (Terminal, Scope, EEprog)
- Compilateur:
  Schéma→ Fib.Textuel → ASM → code
- Compilateur inverse Fib.Textuel → Schéma
- Pages à onglets, catalogue de blocs, démos

Le nouvel environnement
fibula-g



1.3.5 **fibula-G** Outils et accessoires:

Terminal, Simulateur, Scope 8 voies, Graveur d'e2prom, miniscope

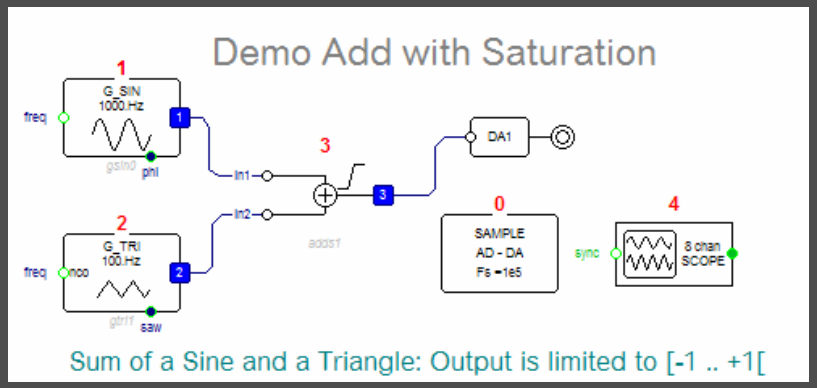


1.3.6 Les objets graphiques: le plan

- Le plan = la page blanche sur laquelle on crée un schéma.
- On peut déplacer le plan entier avec le bouton droit de la souris, on zoome avec la roue.
- On peut basculer en mode texte ou en mode graphique à l'aide de ces boutons:

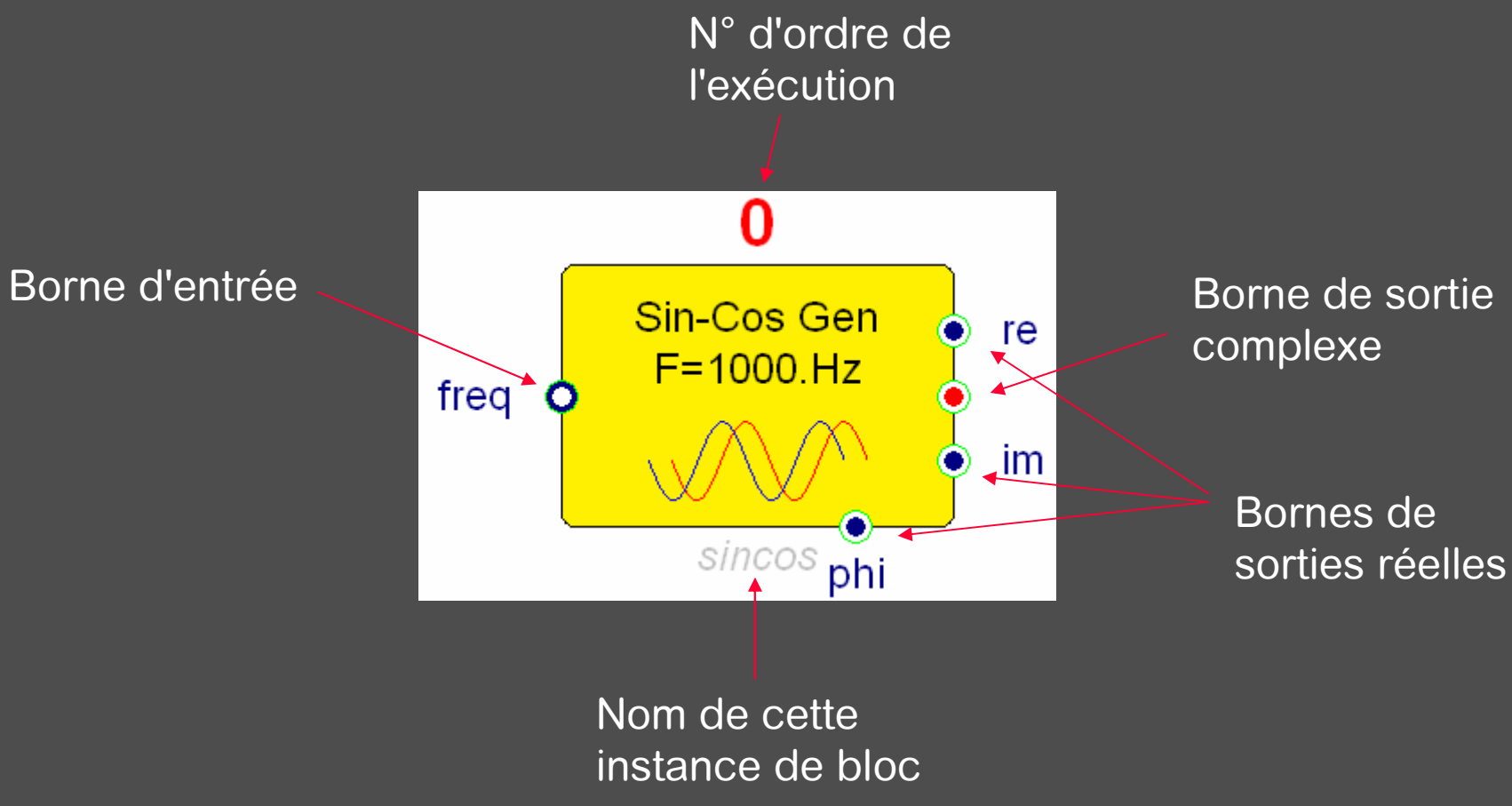
- Pour rafraîchir un plan dont certains objets ont été redéfinis, on bascule en mode texte puis à nouveau en mode graphique.

1.3.6 Mode graphique ⇔ Mode texte



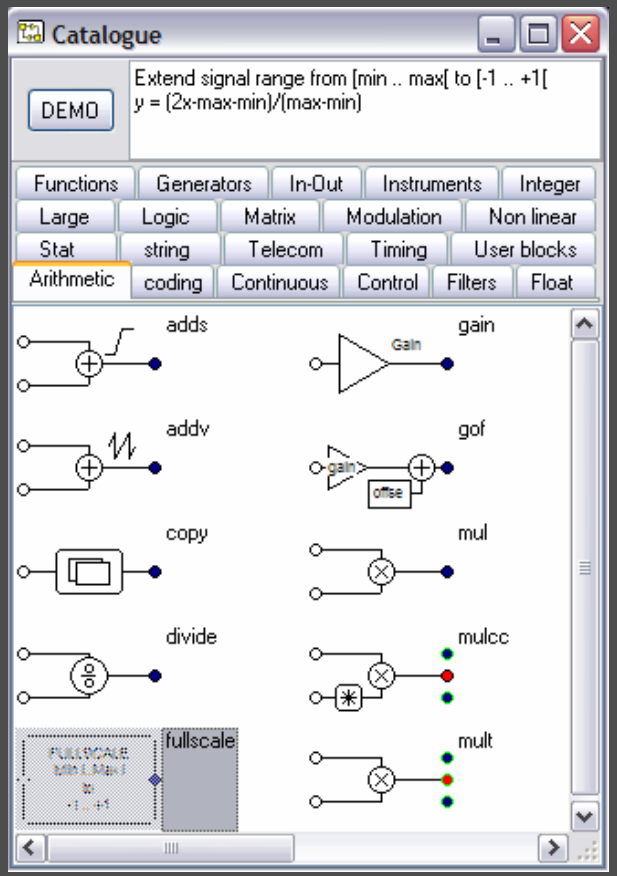

```
PGM
BLOCK
ada,,1
1e5
270,216,72,48,0
BLOCK
g_sin,gsin0,1
1000.,abs
36,132,72,48,0
BLOCK
g_tri,gtri1,1
100.,abs
36,228,72,48,0
BLOCK
scope,,1
500,1,gsin0,gtri1,
adds1,,,,,
378,216,72,48,0
BLOCK
da1,,-1

288,144,72,48,0
WIRE
gsin0,,7,adds1,in1,0,FRACT,WORD
3,19,36,35
WIRE
gtri1,,7,adds1,in2,2,FRACT,WORD
3,21,-36,33
WIRE
adds1,,7,da1,,1,FRACT,WORD
3,33,-36,21
TEXT
123,96,276,27,1
Demo Add with Saturation

TEXT
35,292,431,19,2
Sum of a Sine and a Triangle: Output i
limited to [-1 .. +1[
```

1.3.6 Les objets graphiques: le bloc fonctionnel …

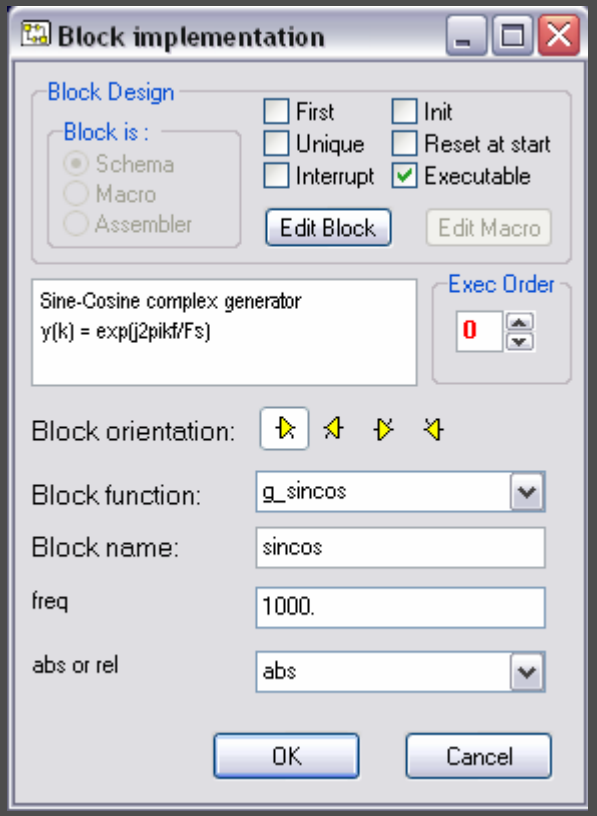


1.3.6 Les objets graphiques: le bloc fonctionnel …

Catalogue de blocs



Clic droit = édition des paramètres



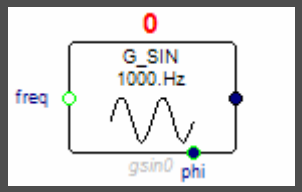
1.3.6 Les objets graphiques: le bloc fonctionnel …
Fichier .fib de description d'un bloc

```
MACRO
g_sin
OPT_INPUT
freq,FRACT,1
OUTPUT
,FRACT,7
OPT_OUTPUT
phi,FRACTD,5
PARAMS
name,frequency,1,1000.,abs=Hz rel=*Fs/2,2,abs,rel
DESCRIPTION
Sine wave generator
Optional input: variable frequency 0..1 -> 0..Fs/2
Optional input if connected overrides parameter

GRAPH
re,0,0,1,1
f,0.5,0.15
sin,0.5,0.7,0.5,0.2,2
a,0
af,1
tw,0.5,0.35
```

Code

Bornes

Paramètres



Description de la fonction

Graphisme

NB: Le fichier .fib n'est pas dépendant du processeur cible.

1.3.6 Les objets graphiques: la donnée

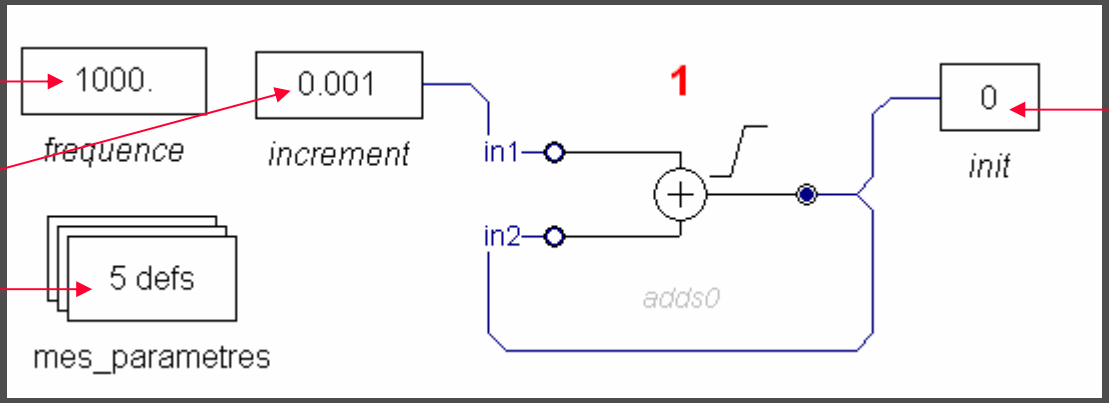
Bouton Donnée

Bouton Liste

Définition

Constante

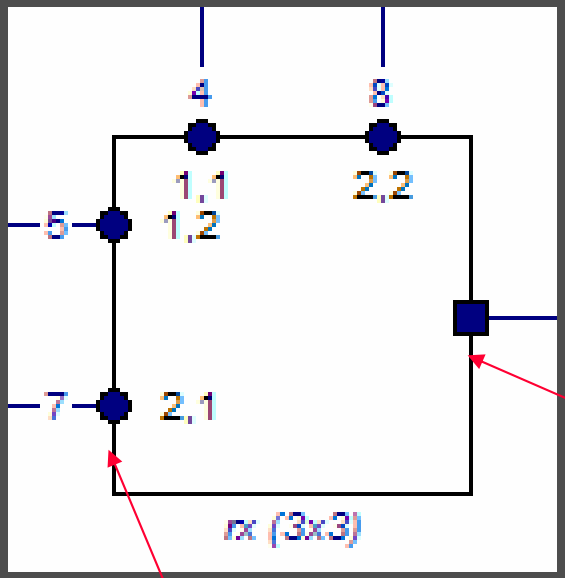
Liste de définitions



Valeur initiale

- Une donnée peut avoir 3 usages:
  - Non connectée: définition (ident. ⇔ valeur)
  - Connectée à une borne d'entrée: constante
  - Connectée à borne de sortie: valeur initiale.
- Ses bornes ne sont pas visibles
- On peut regrouper n définitions dans une liste

1.3.6 Les objets graphiques: la matrice



1.0000000	0.0000000	0.0000000
0.0000000	1.0000000	0.0000000
0.0000000	0.0000000	1.0000000

Fichier .dat de valeurs initiales (optionnel)

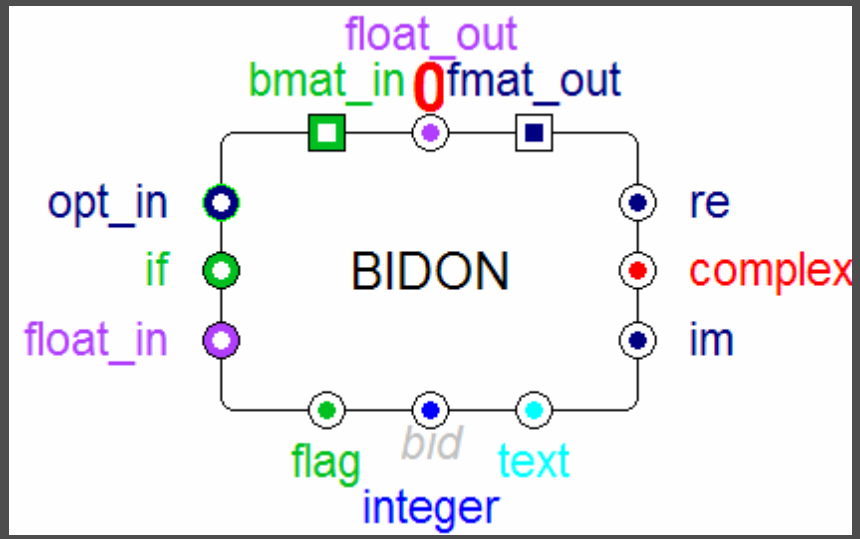
Connexion à la matrice globale (data struct = matrix)
Couleur marine= type fractionnaire

Connexion à un élément particulier de la matrice (data struct = word, dword ou bit)

La forme de la matrice représente ses dimensions:
matrice carrée = carré
ln > col = rectangle haut
ln < col = rectangle large

1.3.6 Les objets graphiques: les bornes

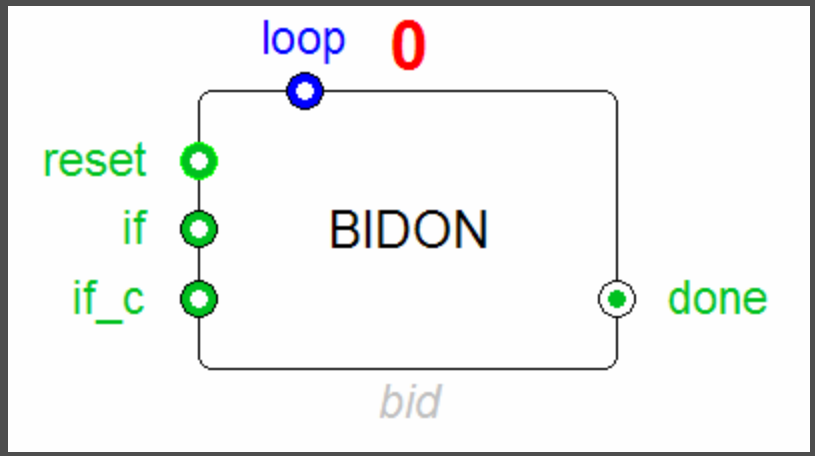
- Les bornes de sortie représentent les variables.
- Leur couleur définit le type de la variable.
- Leur graphisme dépend de leurs propriétés
- On ne peut connecter que Entrée ⇔ Sortie
- Les bornes carrées sont des pointeurs



1.3.6 Les objets graphiques: Les bornes de contrôle de l'exécution

- Les bornes de contrôle sont à rajouter manuellement:

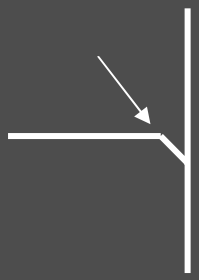
- Loop= exécution N fois
- Reset= activer l'initialisation
- If = exécuter si bool TRUE
- If_c = exéc. si TRUE, puis mettre à FALSE
- Done = fin d'exécution



1.3.6 Les objets graphiques: la connexion

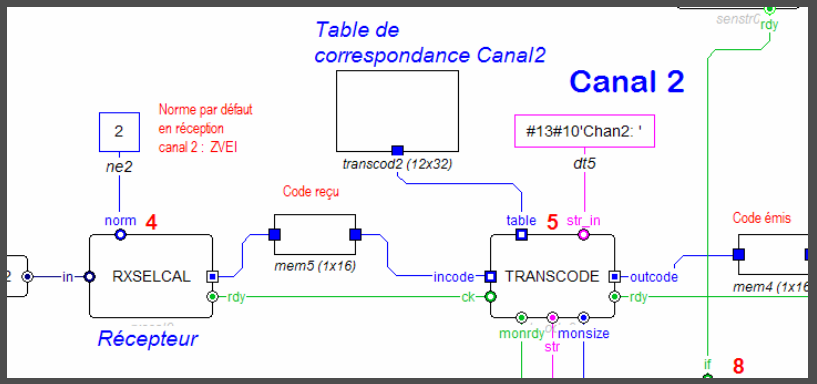
- On crée une connexion entre 2 bornes compatibles (sortie → entrée, même type, même structure de données) en cliquant ce bouton:

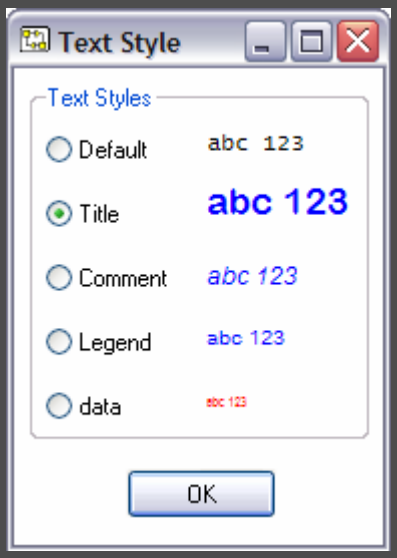
- Puis, la borne 1, puis la borne 2. (Possibilité de points intermédiaires).
- Le logiciel interdit la connexion entre 2 sorties ou entre 2 entrées.
- La couleur représente le type de donnée transportée
- Les angles coupés à 45° indiquent la direction de la connexion lorsque 2 fils se superposent



1.3.6 Les objets graphiques: les textes

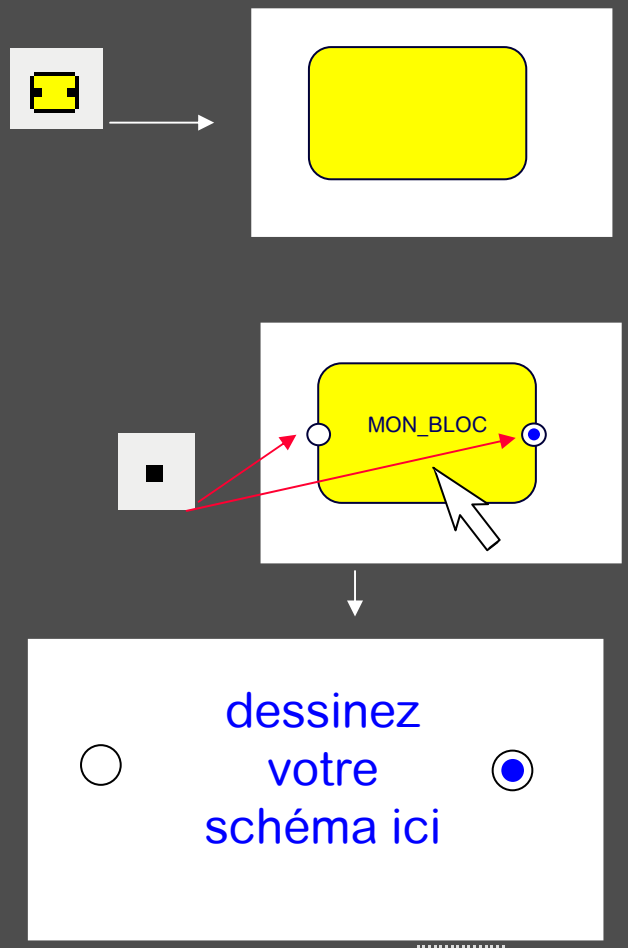
- Les textes servent à commenter un schéma
- On peut définir 5 styles pour les textes.





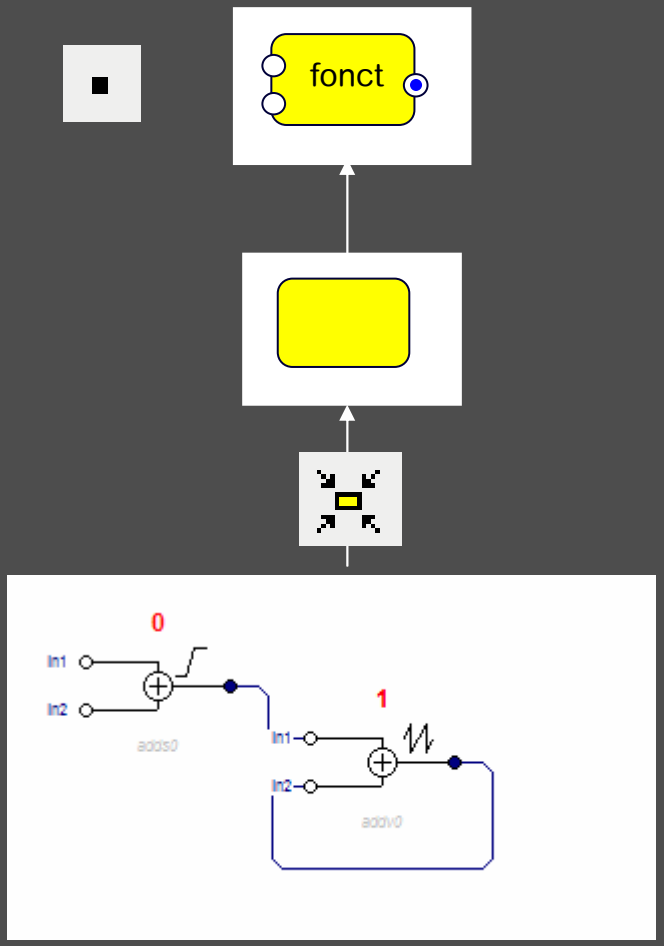
1.3.7 Conception de blocs en hiérarchie descendante

- Instancier un bloc vide:
  - Donner un nom de fonction
  - Placer et paramétrer des bornes
- Double clic ouvre le schéma interne (à créer) où apparaissent les bornes.
- Saisir le schéma
- Ajouter en mode texte
  - Les paramètres
  - Le descriptif
  - Le graphisme
- Sauver le nouveau bloc



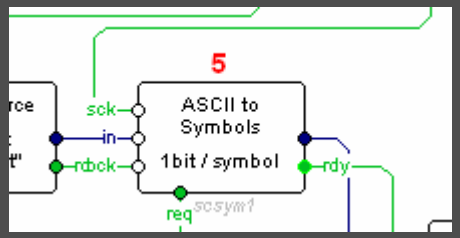
1.3.7 Conception de blocs en hiérarchie montante

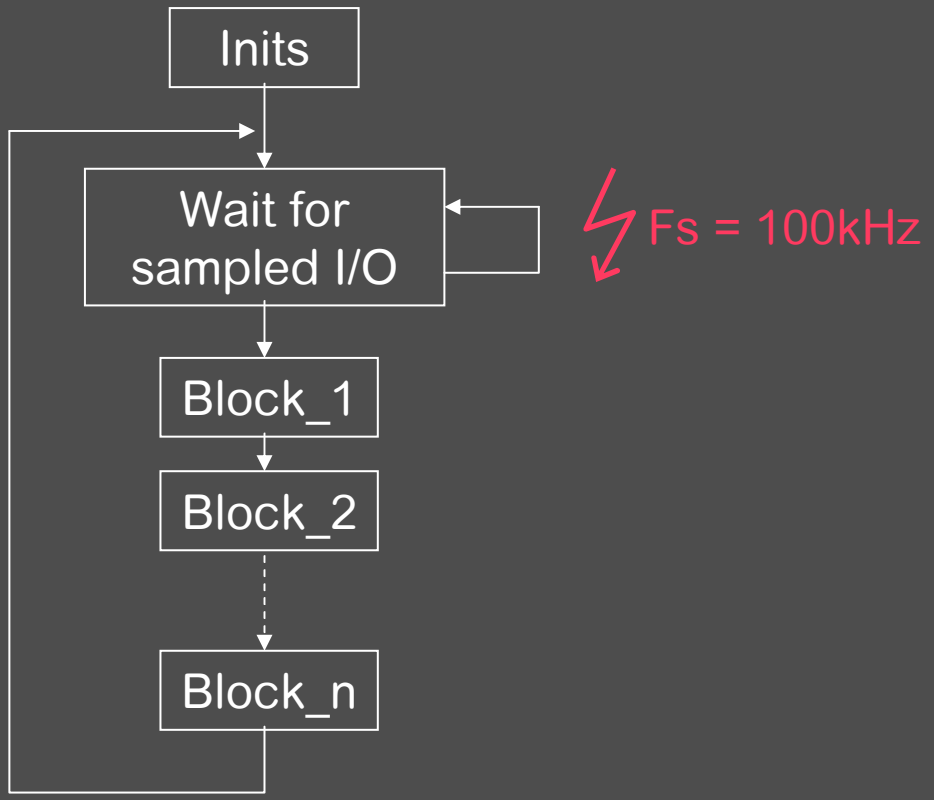
- Créer un schéma
- Réduire ce schéma à un bloc
- Ajouter, puis connecter les bornes
- En mode texte, ajouter au schéma:
  - Les paramètres
  - Le descriptif
  - Le graphisme



1.3.8 Propriétés d'un système à exécution isochrone …

- Dans la plupart des applications, chaque bloc est exécuté 1 fois à chaque instant d'échantillonnage des entrées / sorties.
- L'ordre dans lequel les blocs s'exécutent est défini par le chiffre surmontant chaque bloc.

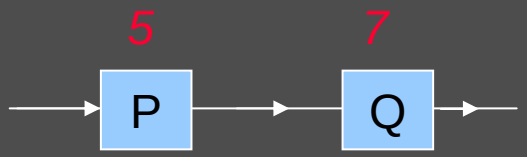




1.3.8 Propriétés d'un système à exécution isochrone …

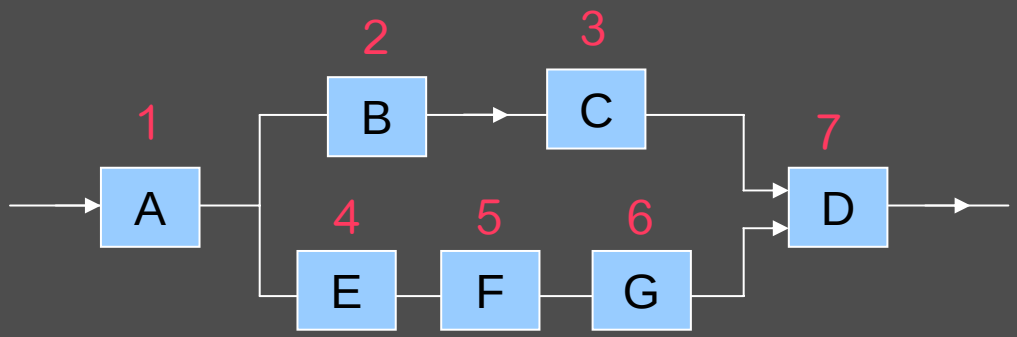
- Tous les blocs sont exécutés séquentiellement au sein d'une même boucle cadencée par la macro ada.
  Il suffit de les interconnecter comme des composants analogiques.

- Habituellement, on choisit l'ordre dans lequel les blocs s'exécutent de manière à ce que les données produites par le bloc P soient consommées par le bloc Q, où Q s'exécute après P.



1.3.8 Propriétés d'un système à exécution isochrone …

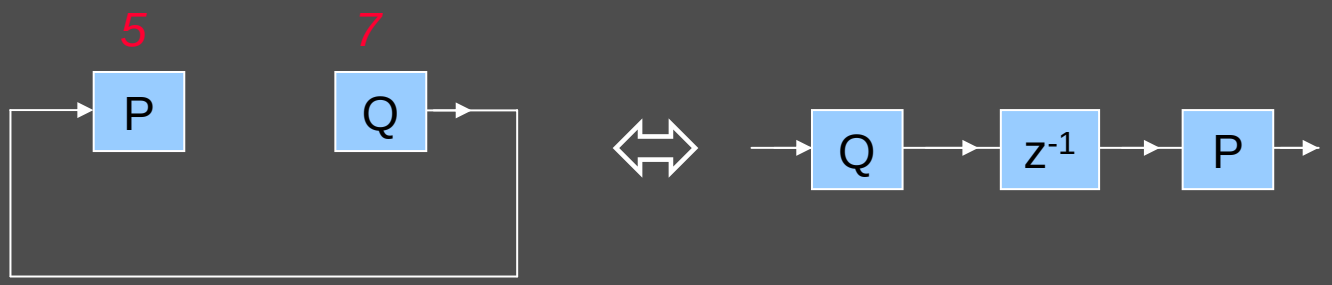
- Le respect de cette règle permet de créer des graphes contenant des branches concurrentes sans introduire de délai.



Les deux entrées du bloc D reçoivent bien deux échantillons synchrones

1.3.8 Propriétés d'un système à exécution isochrone …

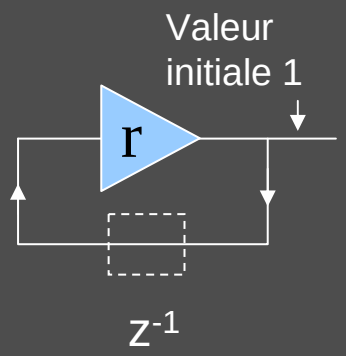
- Non adéquation de l'ordre des blocs à la propagation des données:
  - Si un bloc P reçoit une donnée produite par un bloc Q exécuté ultérieurement, cela introduit un retard élémentaire entre les 2 blocs:



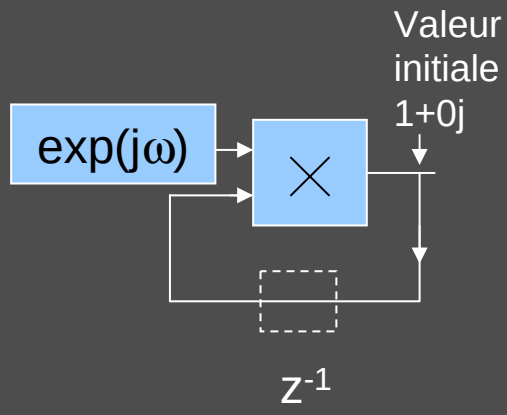
1.3.8 Propriétés d'un système à exécution isochrone

- Exploitation du retard élémentaire

Le retard élémentaire peut être exploité dans de nombreux cas, par exemple boucle entre sortie et entrée d'un même bloc:



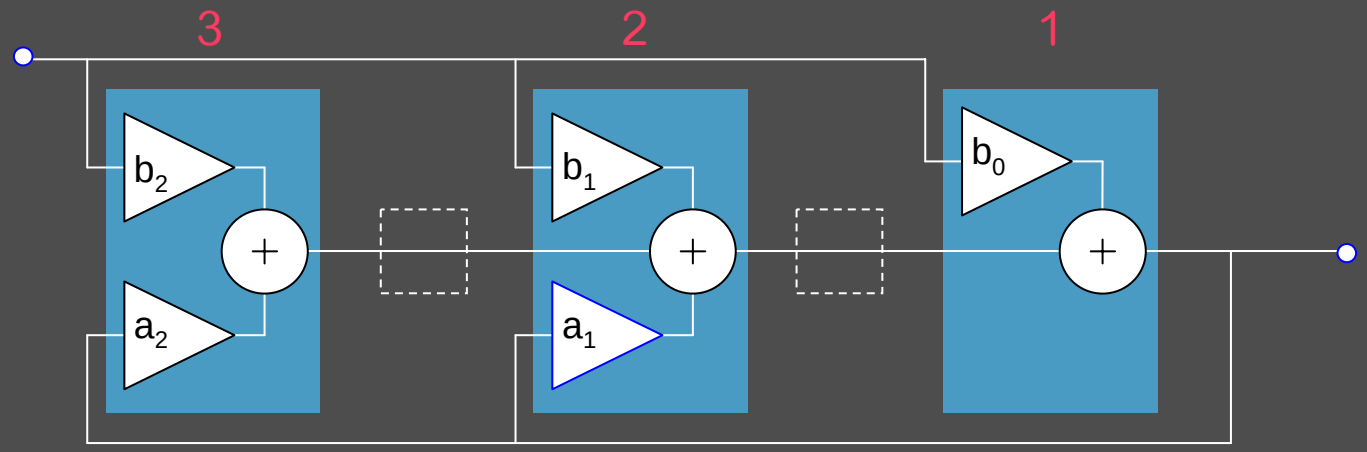
Exponentielle r^n



Oscillateur complexe
$\exp(jn\omega)$

1.3.8 Propriétés d'un système à exécution isochrone …

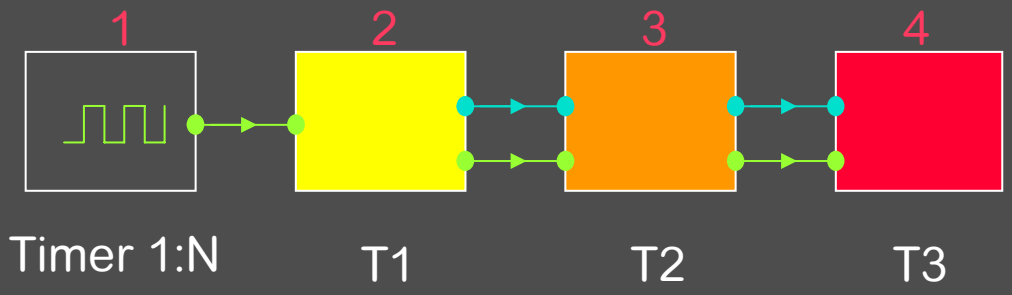
- Dans cet exemple, les retards introduits par l'inversion de l'ordre d'exécution de 3 additions pondérées constituent les variables d'état d'un filtre récursif du second ordre (forme canonique transposée)

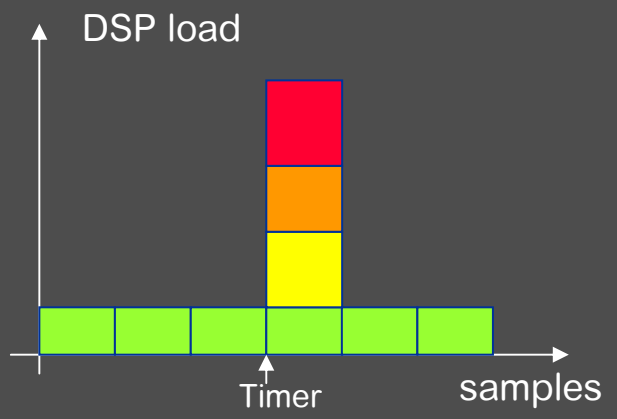


$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

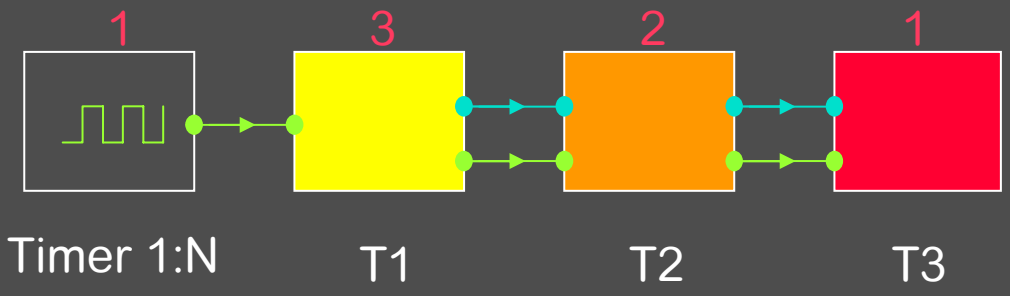
1.3.8 Propriétés d'un système à exécution isochrone
Technique d'étalement des tâches synchronisées

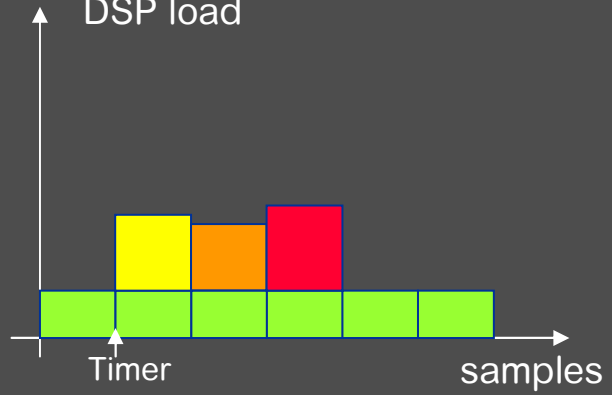
Chaque tâche attend l'achèvement de la précédente pour s'exécuter. Lorsqu'on active la 1ère, les autres s'exécutent en rafale





En inversant l'ordre des exécutions, on n'a qu'une seule tâche par échantillon

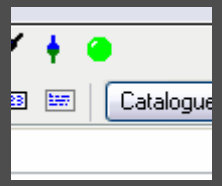




1.4 "Getting started"

1.4.1 **FIBULA-G**: Vérification de la connexion

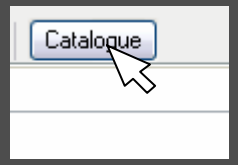
- Lancez FIBULA-G
- Connectez le câble RS232
- Mettez la carte DSP sous tension
- La led d'état doit passer au vert



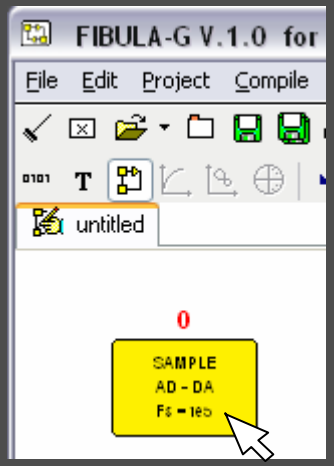
- Sinon changer la connexion ou exécuter la commande
  Settings | Serial Port | Port | COM2

1.4.2 **FIBULA-G**: mode opératoire …

- Étape 1:
  - A Ouvrir le catalogue,
  - B Choisir un bloc (clic G)
  - C Cliquer pour déposer le bloc sur le plan

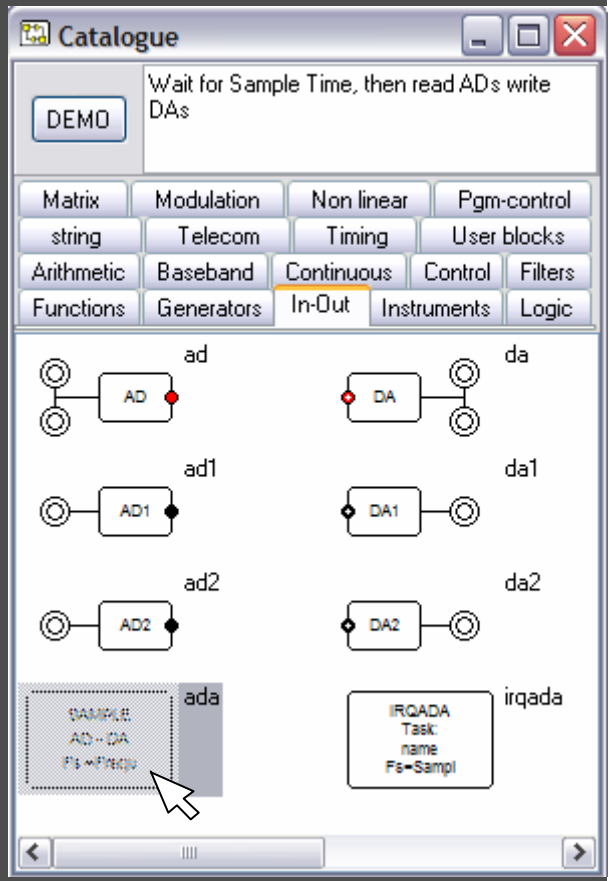


A



C

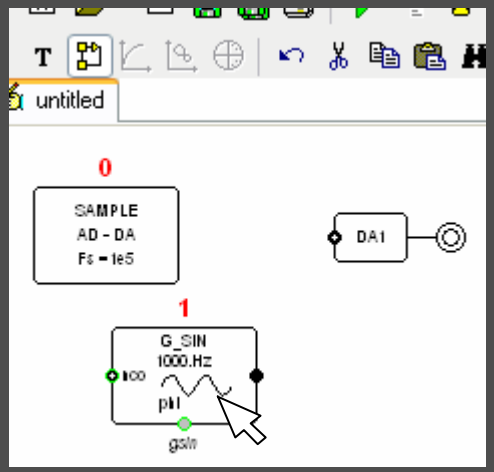
B



1.4.2 **FIBULA-G**: mode opératoire …

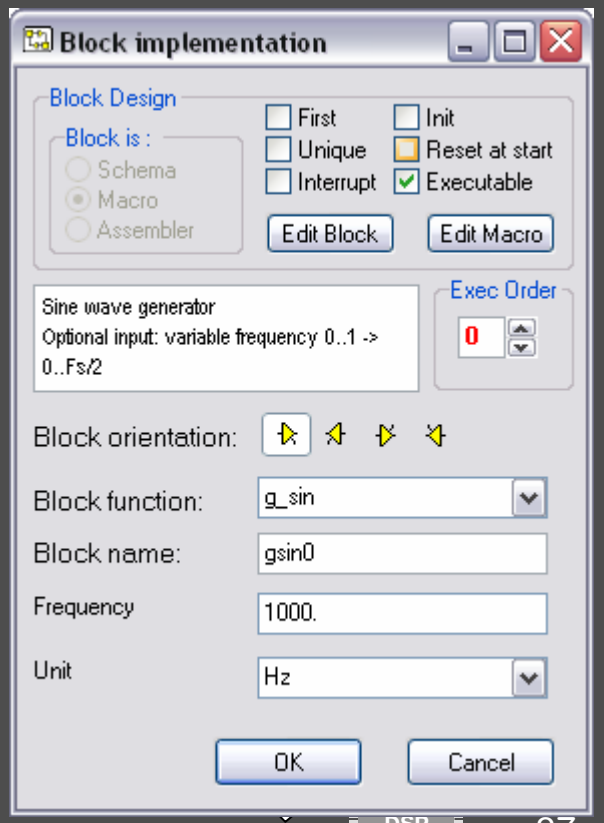
- Etape 2:
  - **A** Déposer tous les blocs nécessaires sur le plan
  - **B** Éditer les propriétés de chaque bloc (clic D sur bloc)
  - **C** Modifier éventuellement les paramètres, valider par OK

A



B

C

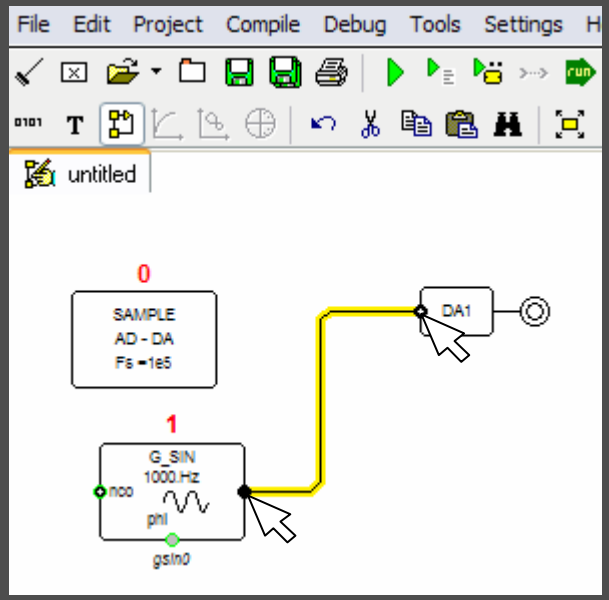


1.4.2 **FIBULA-G**: mode opératoire …

- Etape 3
  - **A** Activer le bouton Connexion
  - **B** Interconnecter les blocs: clic borne sortie (pleine), clic borne d'entrée (creuse) de même couleur (type de donnée)

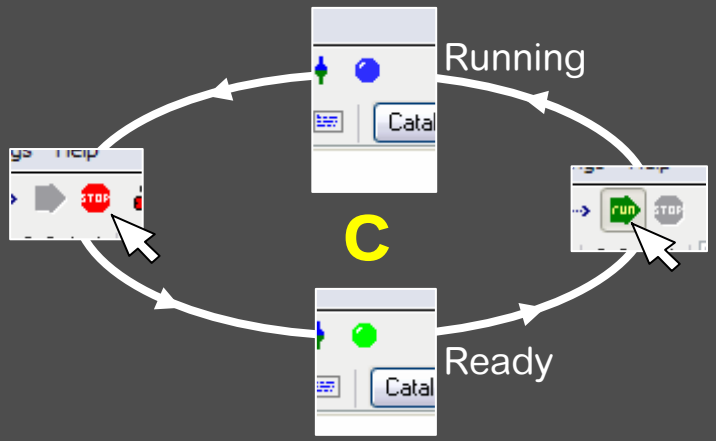
A

B



1.4.2 **FIBULA-G**: mode opératoire …

- Etape 4
  - **A** Compiler – Télécharger – Lancer le programme
  - **B** La led d'état passe au bleu indiquant que le programme s'exécute
  - **C** On peut alors arrêter / redémarrer le programme à volonté



1.4.2 **FIBULA**: mode opératoire.

- Edition graphique
  - Déplacer un bloc sur le plan: Clic G, glisser
  - Déplacer tout le plan: Clic D sur blanc, glisser
  - Zoomer le plan: tourner la molette
  - Sélectionner une partie du plan Clic G, glisser, lâcher
  - Sélectionner tout: double-clic sur blanc
  - Copier, coller, couper, supprimer la sélection, défaire: Ctrl-C Ctrl-V Ctrl-X, DEL, Ctrl-Z
  - Fermer toutes les pages sans sauver (kill) Ctrl K

1.4.3 Acquisition d'un signal Matériel

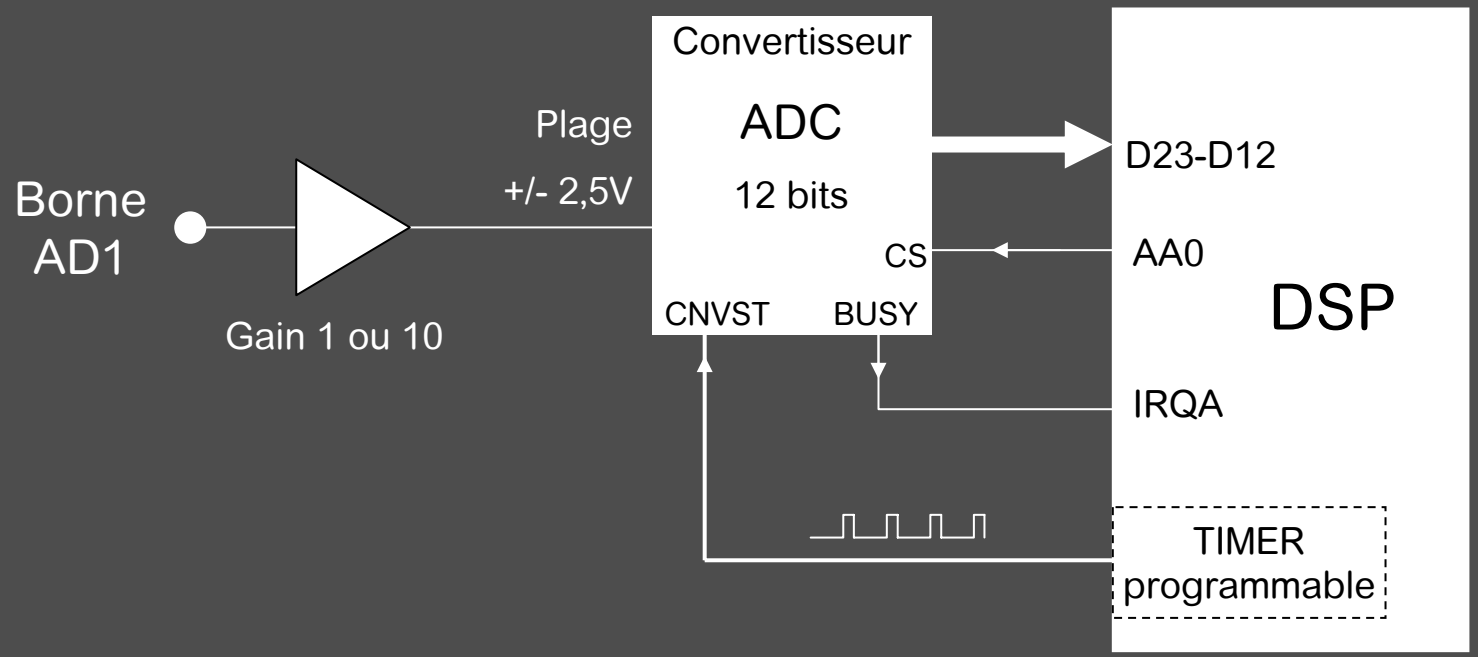


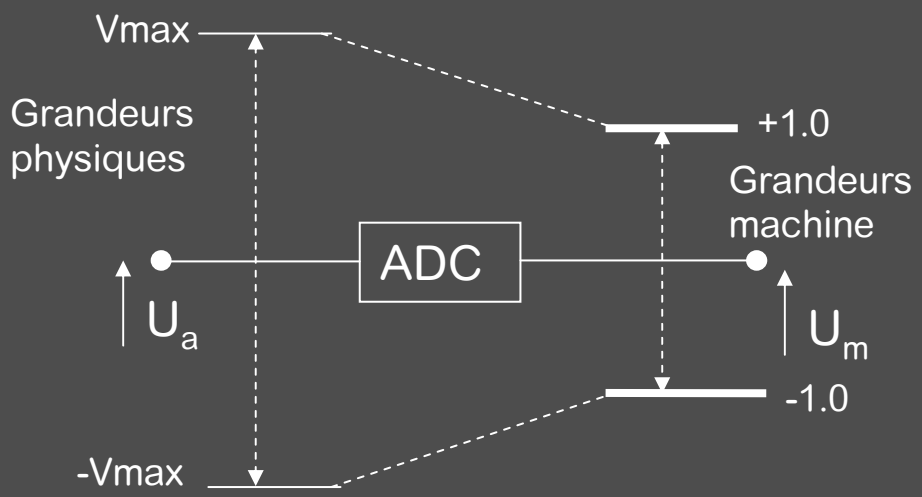
Schéma simplifié du port analogique AD1

1.4.3 Acquisition d'un signal : codage

CAN bipolaire à sortie codée Complément à 2

Entrée CAN Volts	Code Sortie CAN (binaire)	Code Lu par le DSP (hexadécimal)	Valeur du signal en Unité Machine
>= 2,4988	011111111111	7FF000	0,9995
1,25	010000000000	400000	0,5
0,00122	000000000001	001000	0,000488
0	000000000000	000000	0
-0,00122	111111111111	FFF000	-0,000488
-1,25	110000000000	C00000	-0,5
-2,5	100000000000	800000	-1,0

1.4.3 Acquisition d'un signal
facteur d'échelle (processeur à virgule fixe)



Calibrage:

toute grandeur physique x est représentée par

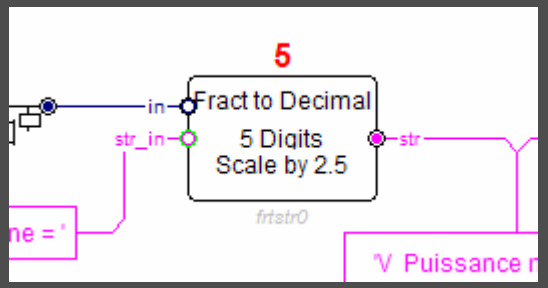
$x / |x_{max}|$

dans la machine

Um = Ua / Vmax

Ua = Vmax Um

Le bloc FRTOSTR convertit une variable fractionnaire en chaîne de caractères telle que à la valeur machine 1.0 corresponde le nombre affiché Vmax (=scale)

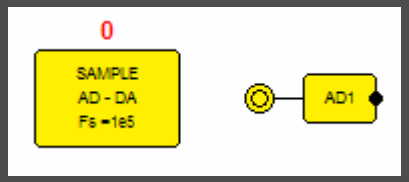


1.4.3 Acquisition d'un signal : code logiciel

Version textuelle: macro ada
Ex:

ada 1e5

Version graphique:



Segment d'initialisations:

Programmer le Timer pour impulsions à la fréquence Fe

Initialiser le DSP pour traiter les interruptions IRQA

Service de l'interruption IRQA:

Copier conversion vers variable ad1

Drapeau=1

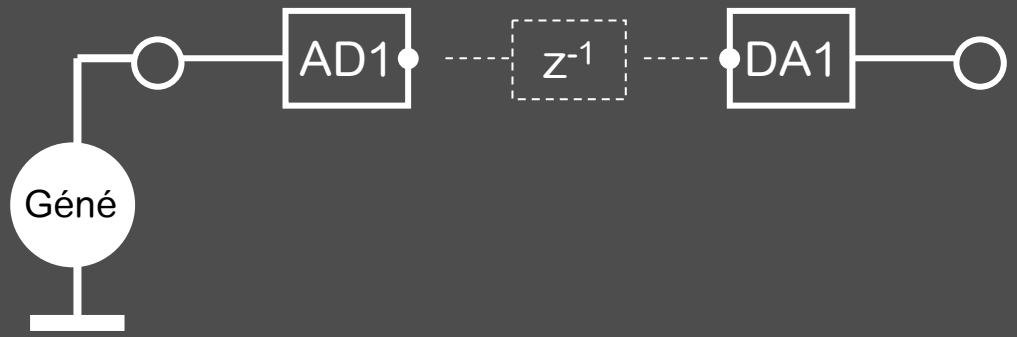
Programme principal:

Initialisations

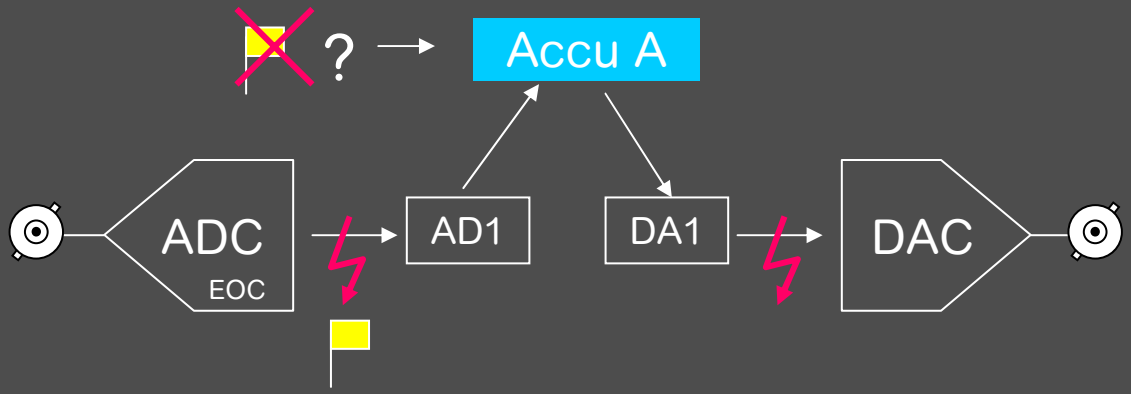
Attente et RAZ drapeau

Traitement d'un échantillon

1.4.3 Acquisition d'un signal
Programmation en Assembleur

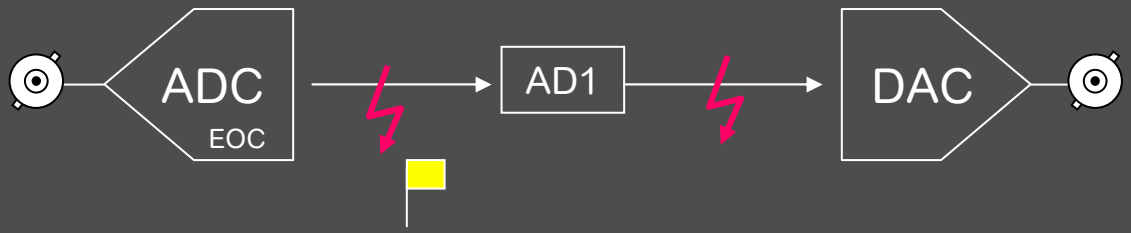



```
            org     x:
ad1         ds      1
da1         ds      1

            org     p:

loop        ada     1e5
            move    x:ad1,a
            move    a,x:da1
            jmp     loop
```

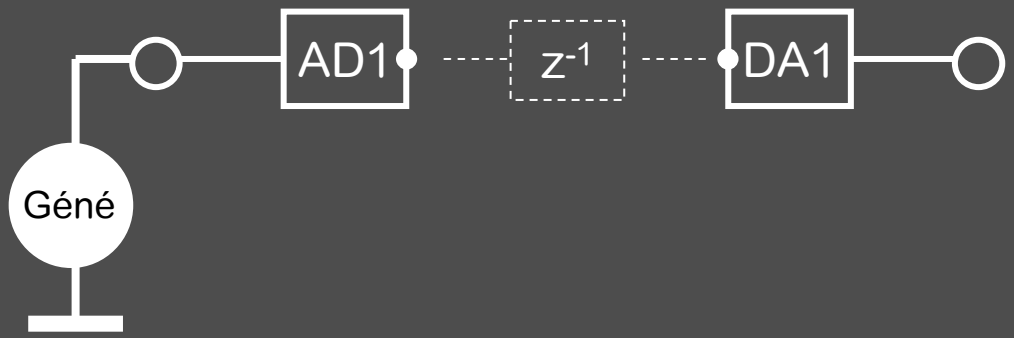


1.4.3 Acquisition d'un signal
Programmation en Fibula Textuel

- AD1 est connecté à DA1
- AD1 et DA1 constituent une variable unique mise à jour sous Interruption
- Le programme principal est une boucle vide



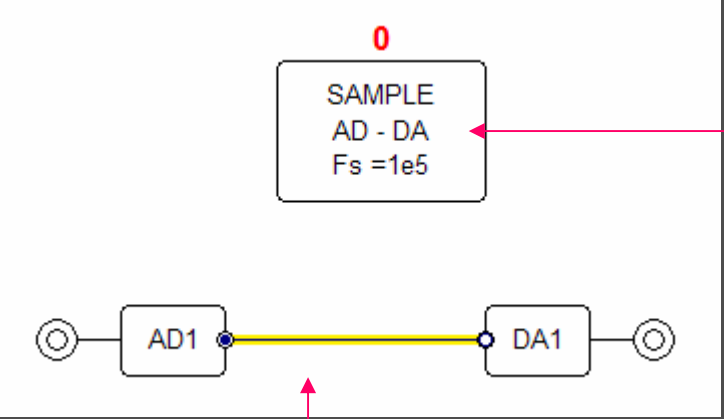

```
        cn      ad1,da1
loop    ada     1e5
        jmp     loop
```



Macro « cn » = connexion

Crée une variable unique constituant la sortie d'un bloc et l'entrée d'un ou plusieurs autres

1.4.3 Acquisition d'un signal
Programmation en fibula graphique



Ce bloc crée le segment d'initialisations, le segment d'interruption, et le segment Temps Réel.

Le segment TR consiste à attendre le drapeau puis le remettre à 0 (test-and-clear).

Ce fil crée la variable commune de sortie du bloc AD1 et d'entrée du bloc DA1

1.4.4 Manipulation

Réalisez les traitements

$$DA1 = |AD1|$$
$$DA2 = (AD1)^2$$

- A) en mode assembleur
- B) en mode Fibula textuel
- C) en mode Fibula-G

1.4.4 Programmation en Assembleur

```
        org     x:                      ; Champ données
ad1     ds      1
da1     ds      1                       ; réservation mémoire
da2     ds      1
        org     p:                      ; Champ programme
loop    ada     1e5                     ; Attente Ts
        move    x:ad1,a
        abs     a           a,x0        ; NB: x0=ad1
        mpy     x0,x0,a     a,x:da1
        move    a,x:da2
        bra     loop
```

T = 8 cycles avec org x:

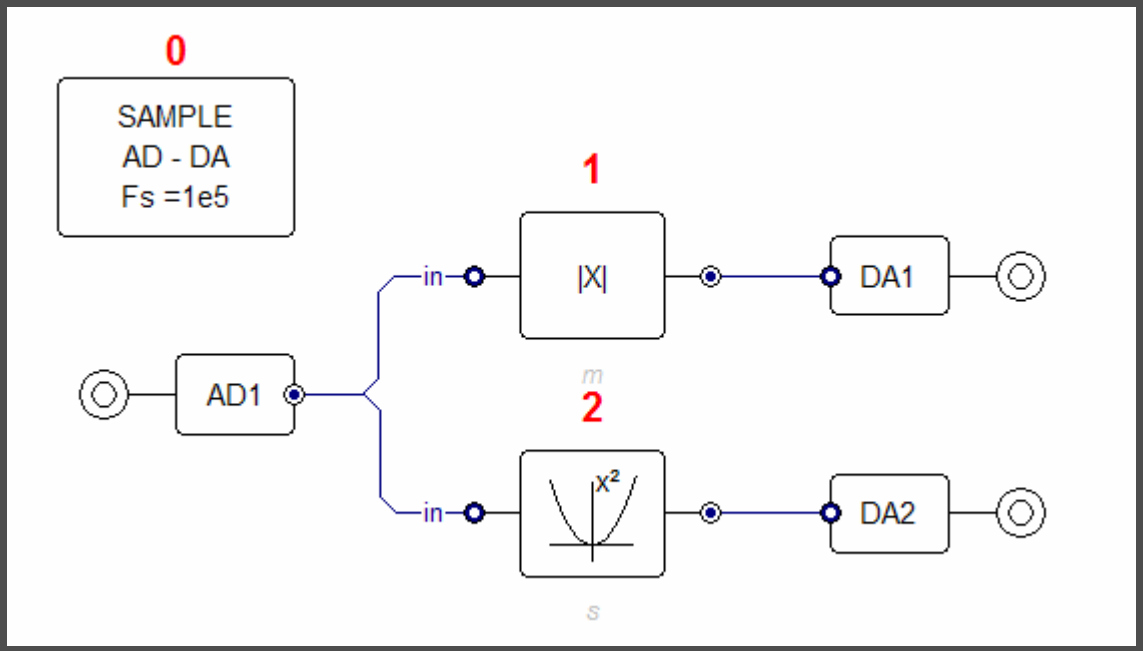
T = 6 cycles avec org x:0

1.4.4 Programmation en Fibula textuel

```
        cn      ad1,m_in
        cn      m,da1
        cn      ad1,s_in
        cn      s,da2
loop    ada     1e5
        magn    m
        square  s
        bra     loop
```

T = 8 cycles

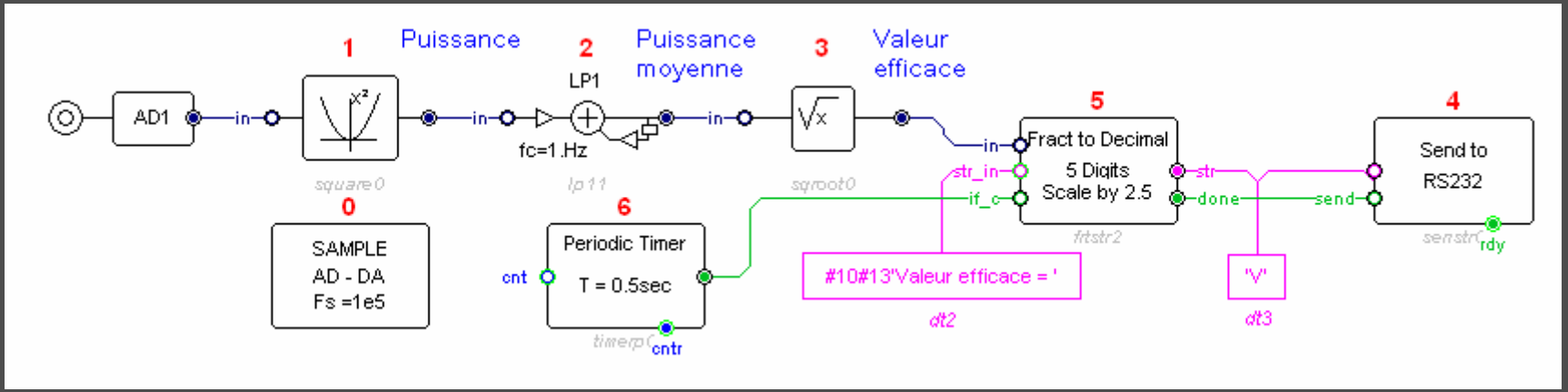
1.4.4 Programmation en Fibula Graphique



T = 8 cycles

Manip:
Mesures à l'aide du CAN de MuPsi

Affichage de la valeur efficace d'un signal

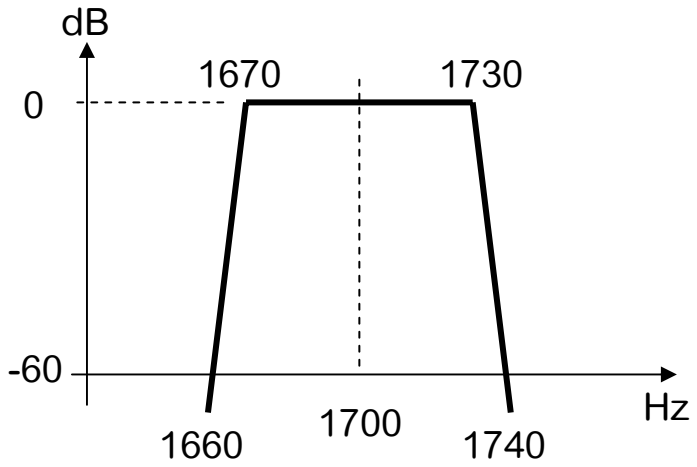


1.5
Quelques applications industrielles
Utilisation en pédagogie

Applications industrielles
Mesures embarquées état des signaux de signalisation ferroviaire

ARNATRONIC - SNCF

Banc de 8 filtres récursifs à spécifications rigoureuses



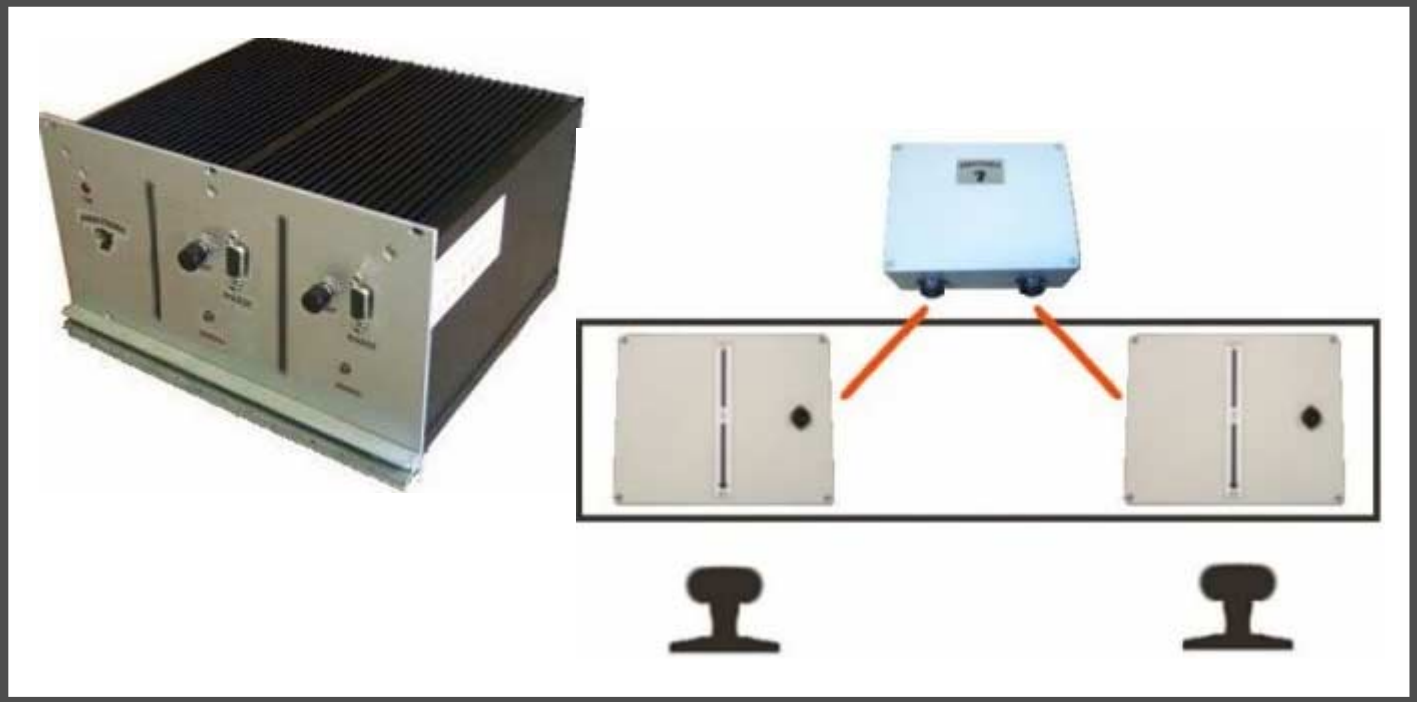
Chaque filtre est constitué d'une cascade de 7 sections du 2nd ordre

Applications industrielles

Mesures du courant de retour traction

ARNATRONIC - SNCF

Capteur de courant dans les rails par mesures différentielles de champs magnétiques alternatifs



Applications industrielles

Analyseur de fréquence pour bourreuse de ballast

ARNATRONIC - SNCF

A partir d'un accéléromètre l'appareil détermine la fréquence exacte du mode de vibration principal de la machine.

Applications industrielles

Détecteur d'impact de meule pour équilibrage de turbocompresseurs

ARNATRONIC - DATATECHNIC

Une analyse fine du signal d'un accéléromètre dans le domaine spectral permet de déterminer l'instant précis où la meule entre en contact avec la pièce.

Pilotage d'un poste de soudure TIG par la carte **mu.psi**

CRAN - Commercy Soudure

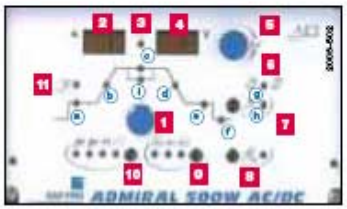
L'Air Liquide Welding

TIG DC or AC/DC welding up to 500 amps.

ADMIRAL 500 AC/DC is a high power AC/DC TIG and coated electrode manual welding machine providing excellent welding performances in all conditions and up to 400 amps at 100% duty cycle. Doted with the latest Digital Signal Processor (DSP) technology, AC noise reduction and enhance performance are provided through a range of wave-form setting options.

High power (400 A at 100% and 500 A maximum)

- Numerical control of the process with DSP (Digital Signal Processor)
- memorization of 15 programs,
- multi-process: DC or AC TIG, TIG spot, pulsed TIG, coated electrode DC+, DC- or AC,
- compact installation with integrated cooling unit,
- AC noise reduction and enhanced performance by a range of wave-form setting options,
- in TIG, single-electrode use possible, whether in DC or AC welding mode (cerium electrode with sharpened point) to enhance penetration power of arc.



1. Parameter selector
2. Current display
3. Memory display (last welding performed)
4. Voltage display
5. Parameter setting
6. Balance adjustment (cleaning time/penetration)
7. Electrode size setting
8. For recording andreading programmes
9. Choice of welding process
10. Choice of welding mode
11. Pulsed function

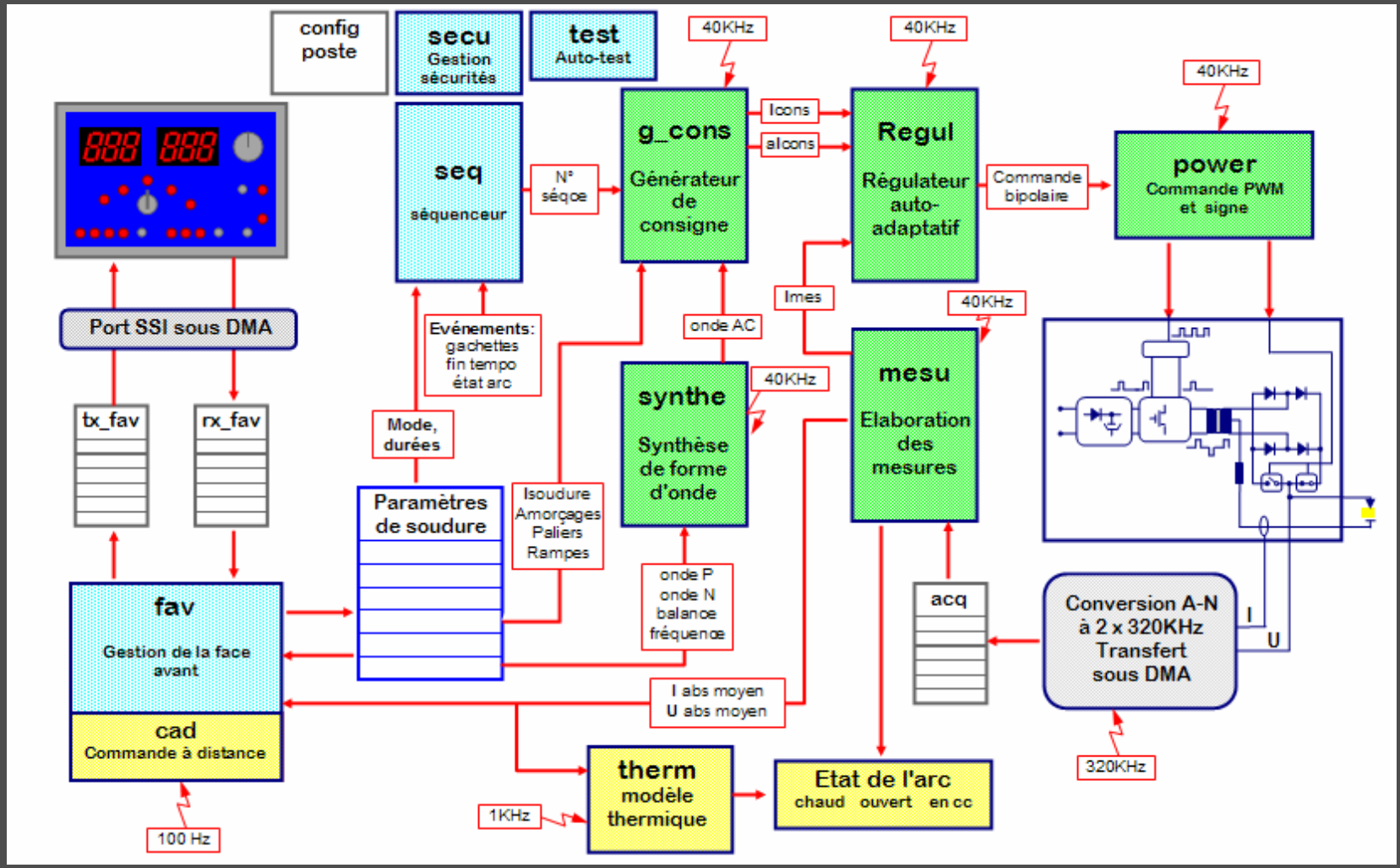
Description of the cycle

(a) Pre-weld current
(b) slope up
(c) Welding current
(f) Post-gas
(g) Balance
(h) Electrode diameter

Options

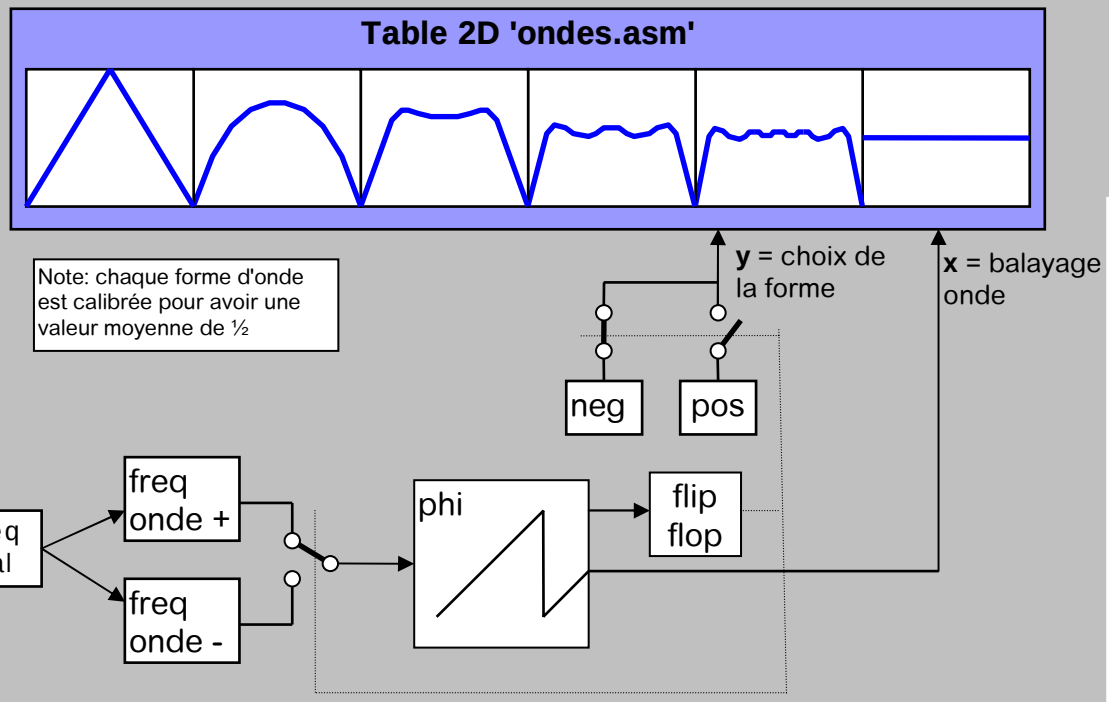
Pilotage d'un poste de soudure TIG (2000-2001)

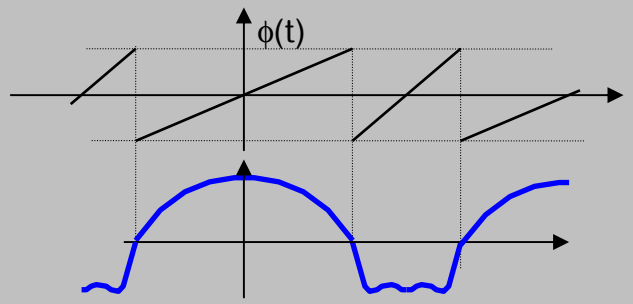
Schéma bloc de l'application

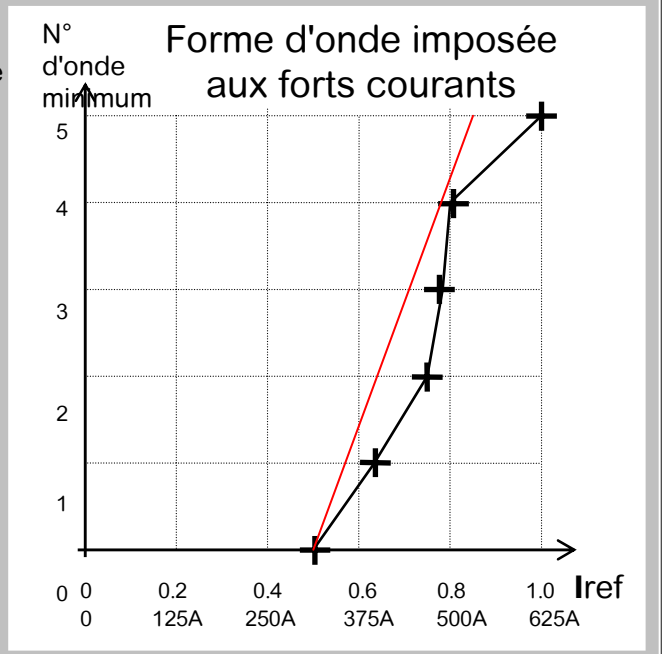


Pilotage d'un poste de soudure TIG (suite)
Synthèse de la forme d'onde consigne courant

Morphing de la forme d'onde par table 2D interpolée





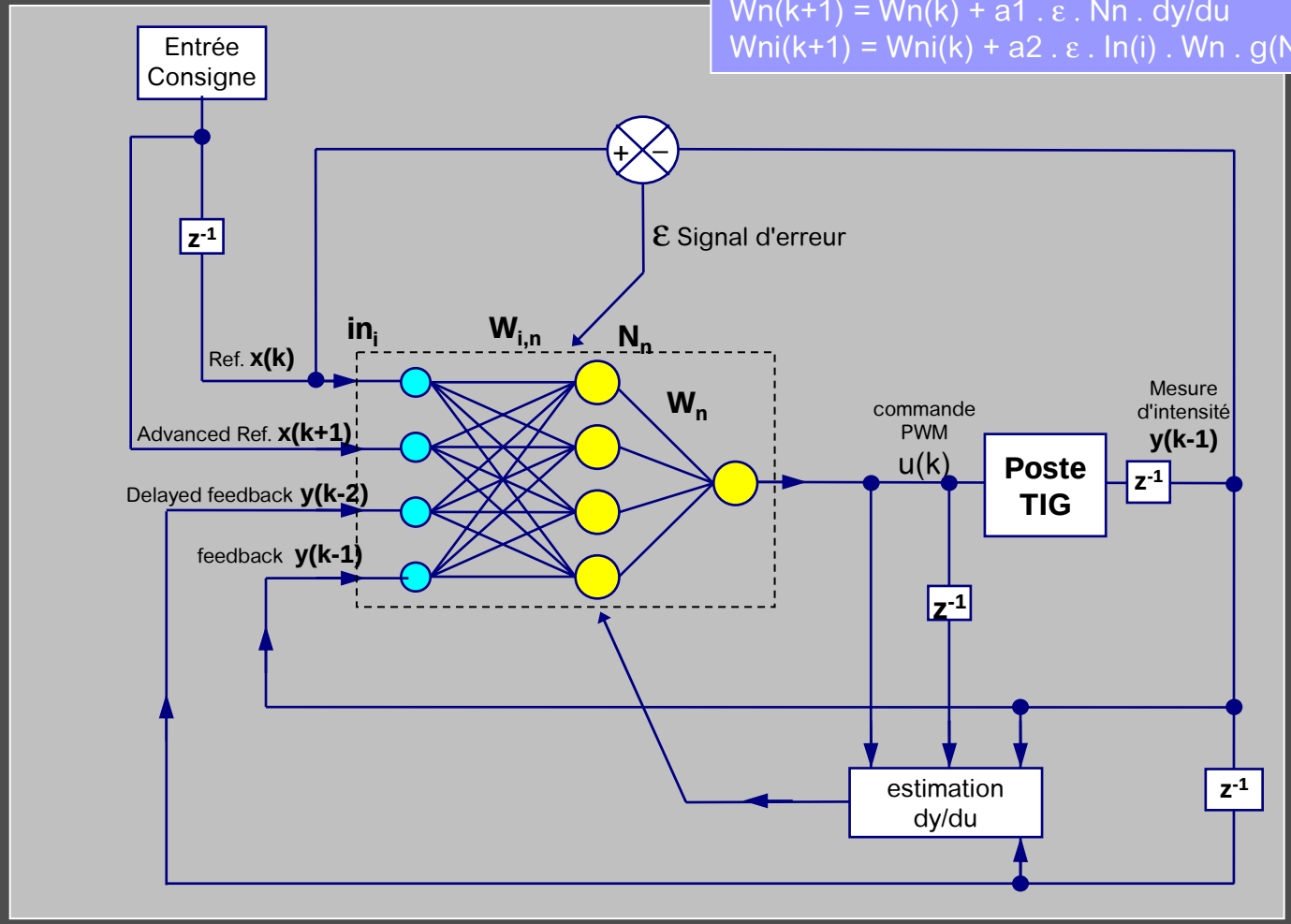


Poste de soudure TIG (suite)
Commande neuronale

Apprentissage par rétro propagation du gradient:

$Wn(k+1) = Wn(k) + a1 . \varepsilon . Nn . dy/du$

$Wni(k+1) = Wni(k) + a2 . \varepsilon . In(i) . Wn . g(Nn) . dy/du$



Poste de soudure TIG (fin)

Estimation de la puissance de calcul nécessaire

Tâche à exécuter	Puissance (DSP-MIPS)
Séquenceur	0,01
Générateur de consigne	3,50
Acquisition des mesures	4,40
Régulation adaptative	17,80
Gestion des paramètres	0,01
Total Puissance de calcul	**25,72**

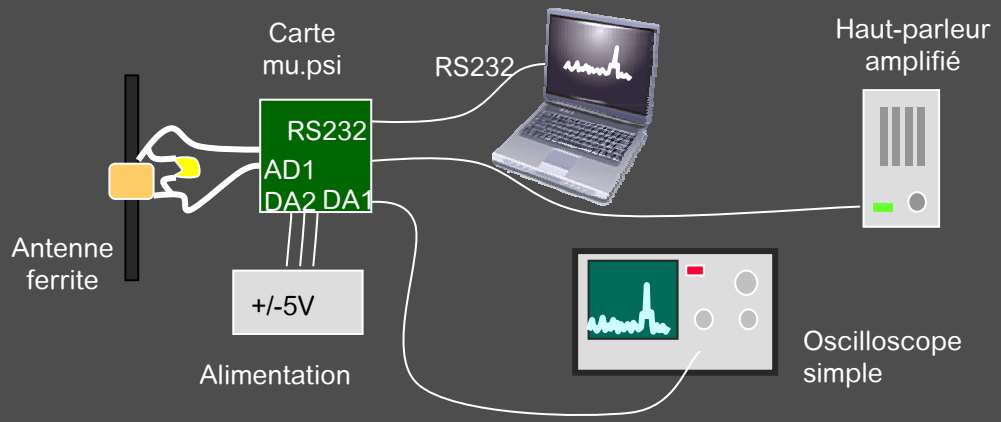
8 Enseignement

Signaux,
Transmissions Numériques,
Théorie de l'Information
Codage

Signaux TP "Modulation d'amplitude"

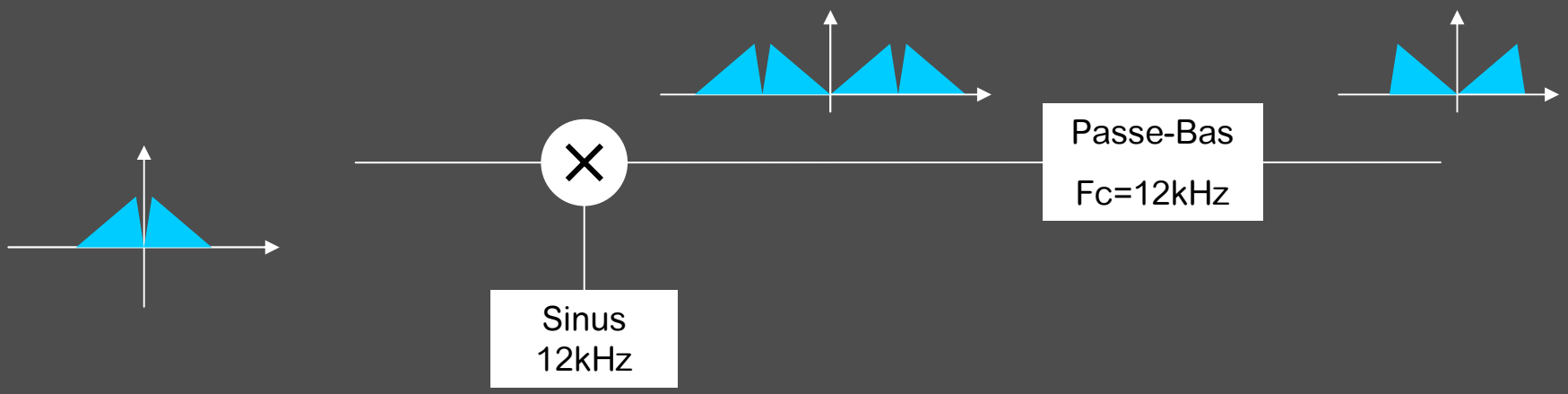
Spectre d'un émetteur radio, filtrage, démodulation

- Récepteur radio Ondes Longues (RTL, Europe1) avec visualisation du spectre en TR



Transformations du spectre audio

- Décalage de fréquence
- Multiplication de fréquence
- Inversion de bande (cryptage son Canal+):



DEMO

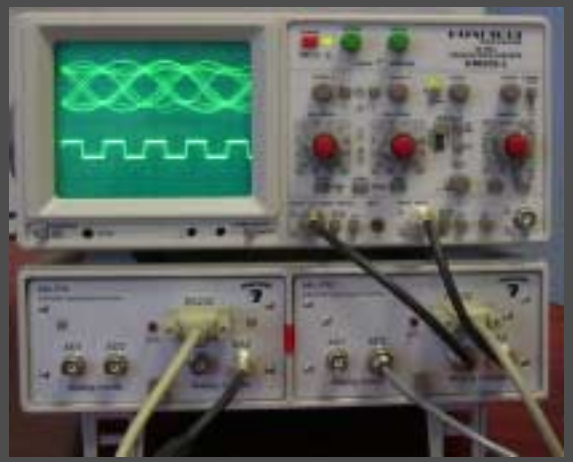
Transmissions Numériques
Bande de base

Canal à bande limitée:

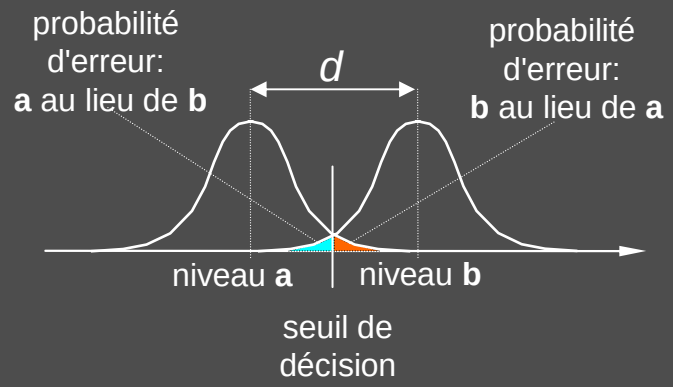
Diagramme de l'œil

Régénération de l'horloge

Pulse Shaping RRC

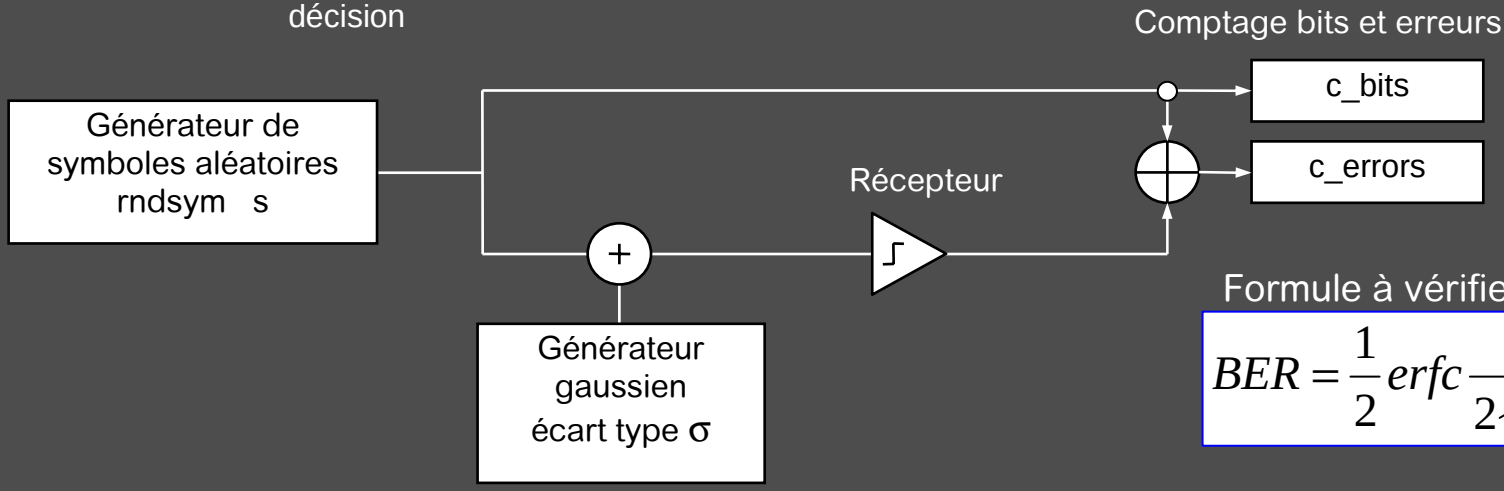


Transmissions Numériques
Mesure du taux d'erreur sur un canal gaussien (AWGN)



La rapidité du DSP permet d'obtenir rapidement des statistiques sur de très grands nombres (10^6)

Le générateur gaussien doit être très précis. On l'obtient par addition de 25 variables aléatoires uniformes.



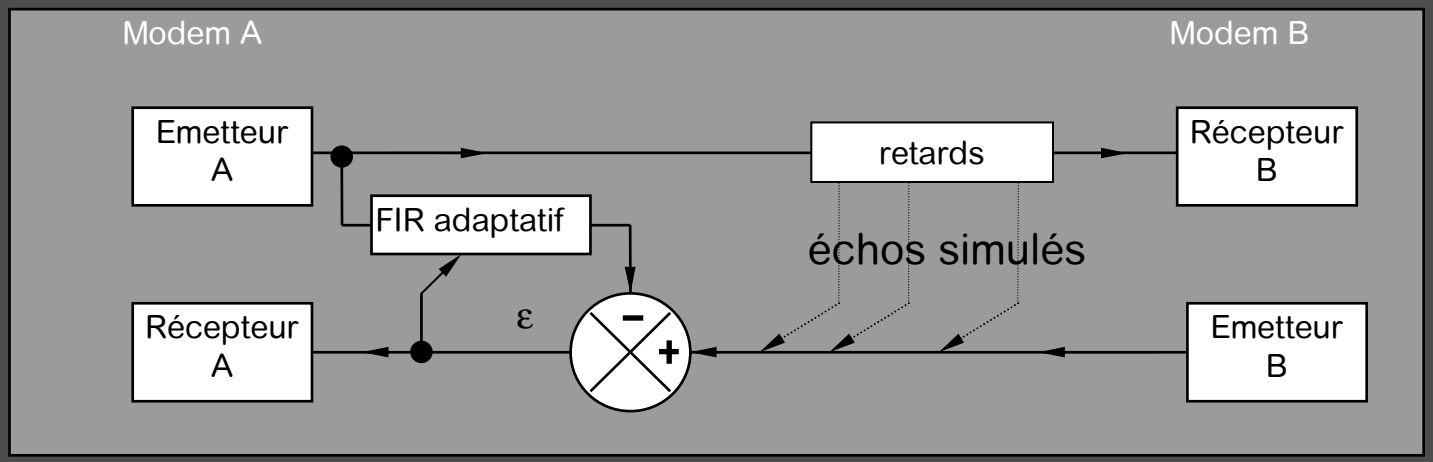
Formule à vérifier:

$$BER = \frac{1}{2} erfc \frac{d}{2\sqrt{2}\sigma}$$

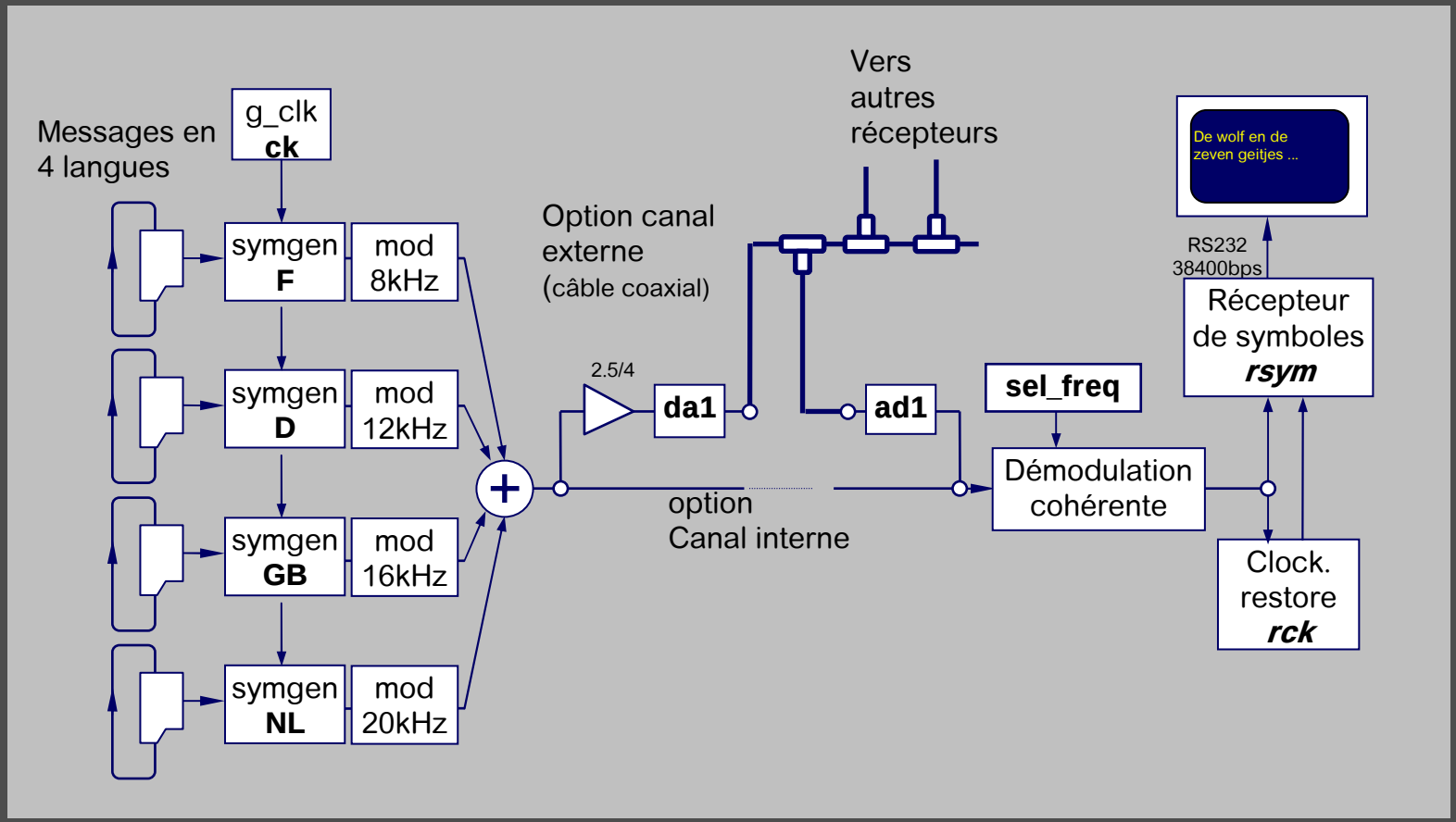
Transmissions Numériques (suite)

Suppression d'échos en téléphonie

Filtre RIF auto adaptatif (algorithme LMS)



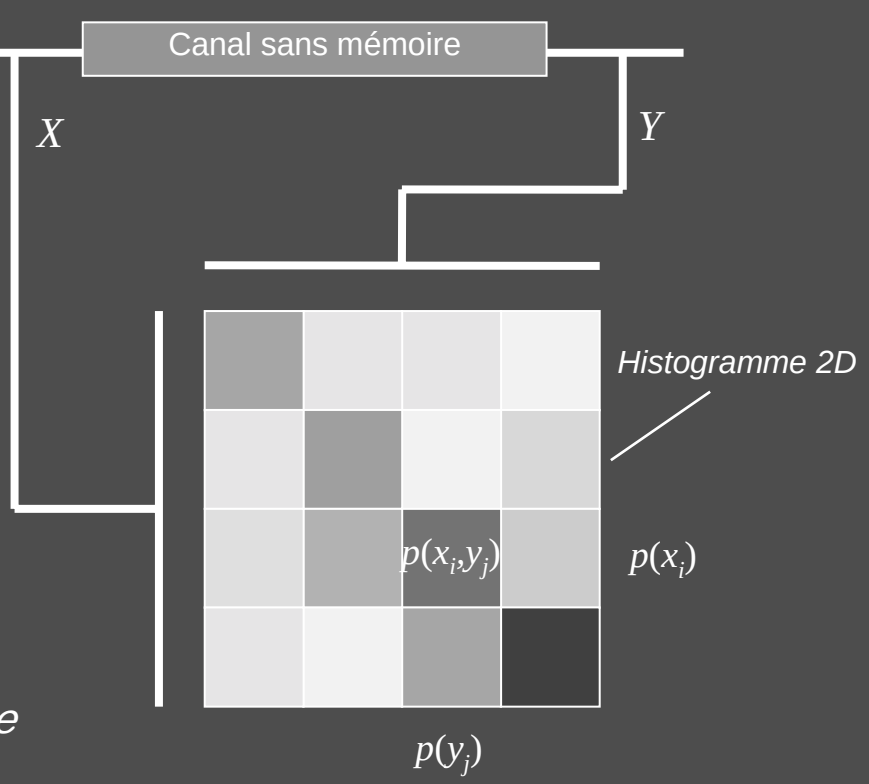
Transmissions Numériques
Multiplexage fréquentiel



Théorie de l'information

- Mesure de l'Information Mutuelle entre entrée et sortie d'un canal sans mémoire

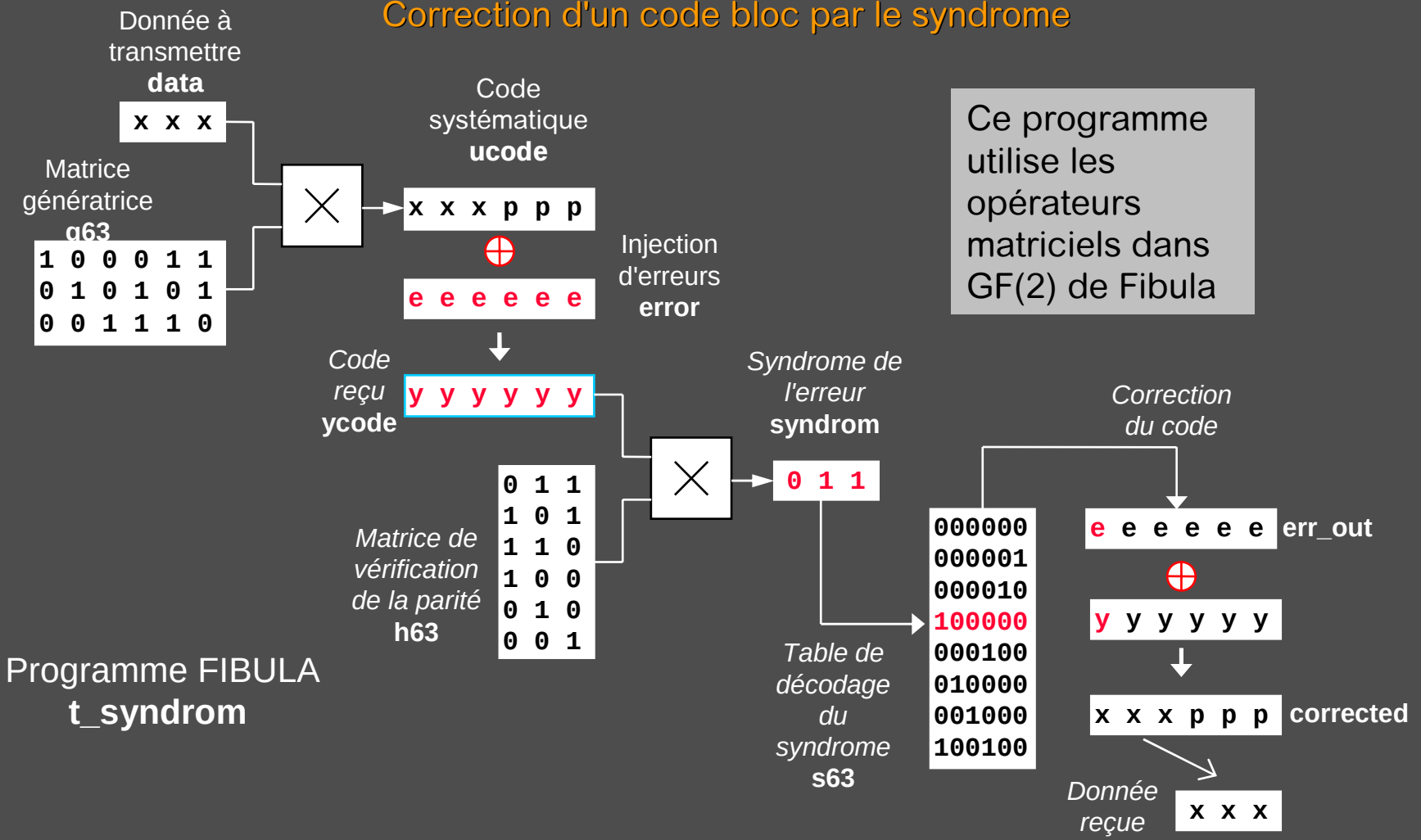
$$I = \sum_i \sum_j p(x_i, y_j) \log_2 \frac{p(x_i, y_j)}{p(x_i)\, p(y_j)}$$



La rapidité du processeur DSP permet d'obtenir rapidement des statistiques sur de très grands nombres d'échantillons.

Codage

Correction d'un code bloc par le syndrome

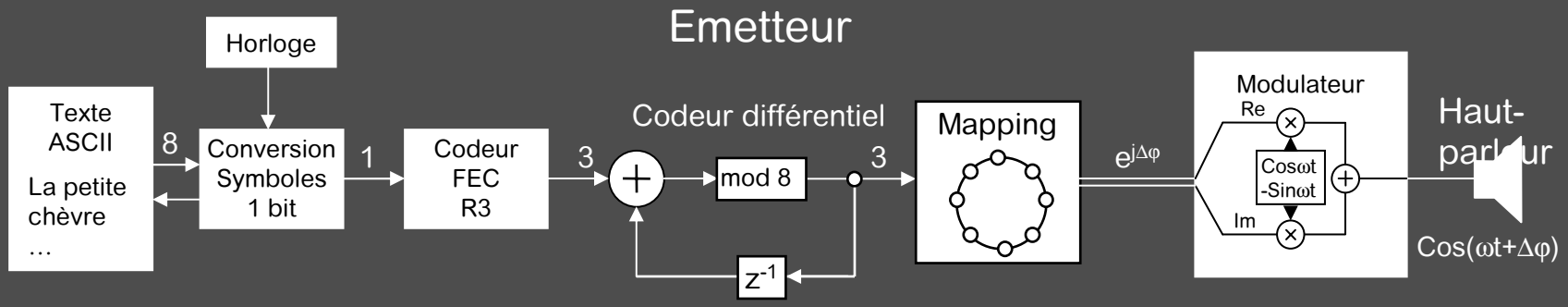


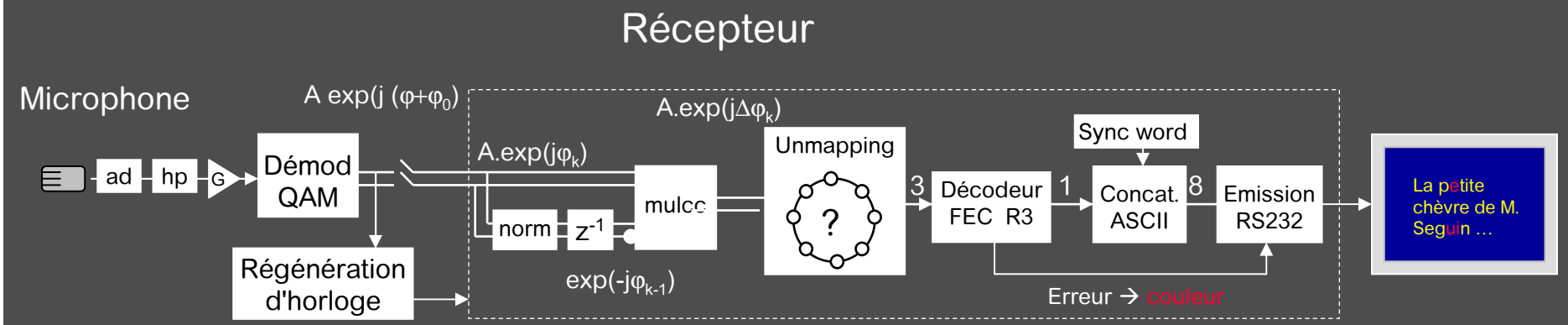
Ce programme utilise les opérateurs matriciels dans GF(2) de Fibula

Programme FIBULA
t_syndrom

Transmissions Numériques (fin)

- Chaîne complète de transmission: modem acoustique DPSK8





Conclusions

- Les DSP sont maintenant d'emploi très courant dans divers domaines de l'industrie.
- Pour de bonnes performances, il est important de bien choisir le type de DSP adapté à son besoin.
- La majeure partie du temps de développement est toujours dévolue au logiciel.
- Loin d'être obsolète, la programmation en assembleur, bien structurée à l'aide de blocs fonctionnels hiérarchiques, prouve son efficacité dans la langage Fibula.

Adresses utiles

ESSTIN
École Supérieure des Sciences et Technologies de l'Ingénieur de Nancy
2 rue Jean Lamour 54519 Vandœuvre lès Nancy
www.esstin.uhp-nancy.fr
JM Ory
Tél. 03 83 68 51 33

DIDALAB
Z.A. de la Clef Saint Pierre
5, rue du groupe Manoukian
Tel. : 01.30.66.59.63
Fax. : 01.30.66.72.20
Mail : ge@didalab.fr
Web : www.didalab.fr

IFETURA-EU département Arnatronic
2 rue du Mad 54530 Arnaville
www.arnatronic.com
Tél. 03 83 80 02 02

IFETURA - ARNATRONIC

Bibliographie 1

Signaux continus:

Frédéric de COULON Théorie et traitement des signaux.
Presses Polytechniques et Universitaires Romandes ISBN: 2-88074-319-2
Jean-Pierre DELMAS Eléments de théorie du signal: Signaux déterministes
Ellipses ISBN 2-7298-9142-0
Maurice CHARBIT Eléments de théorie du signal: Les signaux aléatoires
Ellipses ISBN 2-7298-9075-0
Dominique VENTRE Communications analogiques
Ellipses ISBN 2-7298-9161-7
B. PICINBONO Théorie des signaux et systèmes avec problèmes résolus
Dunod ISBN 2-04-018837-1
J. MAX Méthodes et techniques de traitement du signal. Applications aux mesures physiques.
Masson Tome 1 ISBN 2-225-80470-2 Tome 2 ISBN 2-225-80785-X

Signaux discrets:

E. TISSERAND, J-F. PAUTEX, P. SCHWEITZER
Analyse et traitement des signaux - Méthodes et applications au son et à l'image
Cours et exercices corrigés ISBN 2-10-007481-4 2004 Dunod
A.V. OPPENHEIM, R.W. SCHAFER Digital signal processing
Prentice-Hall 75
M.BELLANGER Traitement numérique du signal. Théorie et pratique
Masson 87
Murat KUNT Traitement numérique des signaux
Presses Polytechniques et Universitaires Romandes ISBN: 2-88074-352-4
Robert MIQUEL Le filtrage numérique par microprocesseurs
Editests ISBN 2-86699-029-3
Roman KUC Introduction to Digital Signal Processing
McGraw-Hill ISBN 0-07-035570-3
A.W.M VAN DEN ENDEN N.A.M. VERHOECKX Traitement numérique du signal Une introduction.
Masson ISBN 2-225-82522-X (1992)
BOAZ PORAT A course in Digital Signal Processing
John Wiley ISBN 0-471-14961-6 (1996)

Bibliographie (2)

Processeurs DSP

Craig Marven & Gillian Ewers
A simple approach to Digital Signal Processing ISBN 0-904047-00-8 Texas Instruments

Mark McQuilken & James P. Leblanc
Digital Signal Processing and the Microcontroller Motorola

Patrice Nus
Traitement Numérique du Signal
Applications du processeur spécialisé DSP56002
Publitronic Elektor

Motorola Reference books (.pdf files)
DSP56300FM Family reference manual
DSP56309UM User Manual
DSPASM Assembler reference manual
DSPLNK Linker and utilities manual

Motorola Application notes (.pdf files)
Fractional and Integer Arithmetic using the DSP56000 family ...
Implementing IIR/FIR filters with Motorola's DSP5600x Digital Signal Processors
FFT Implementation on Motorola's Digital Signal Processors
Application optimization for the DSP56300/DSP56600 Digital Signal Processors

Bibliographie (3)

Transmissions numériques

A.B. CARLSON *Communication systems*
Mc Graw-Hill ISBN 0-07-100560-9
A. SPATARU *Fondements de la théorie de la transmission de l'information*
Presses Polytechniques Romandes ISBN 2-88074-133-0
P. LECOY *Technologie des Télécoms* Hermes ISBN 2-7462-0002-3

Théorie de l'Information, codage

C.E. SHANNON A mathematical Theory of Communication The Bell System Technical Journal Jul-Oct. 1948
CALDERBANK *The Art of Signaling: Fifty years of Coding Theory* IEEE Transactions Vol 44 Oct. 98 [history_50years.pdf]
G. BATAIL *Théorie de l'Information, Application aux techniques de communication*. Masson 1997
M. URO Polycopiés de l'Institut National des Télécommunications [sur le Web]
L.WEHENKEL *Théorie de l'Information et du Codage* (Poly de Cours de l'Université de Liège) [sur le Web]
V. IPATOV *Coding and Encryption in Telecommunication* (Poly de cours de l'Université de Turku, Finlande) [sur le Web]
M. PURSER *Introduction to error correcting codes* Artech House
J.L. MASSEY Applied Digital Information Theory (Poly de l'ETH Zurich) [sur le Web]
D.J.C. MACKAY *Information Theory, Inference and learning algorithms* [sur le Web]
J. ZIV, A.LEMPEL *A Universal Algorithm for Sequential Data Compression* IEEE 1977 [ziv77.pdf]