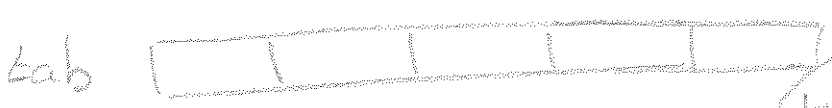
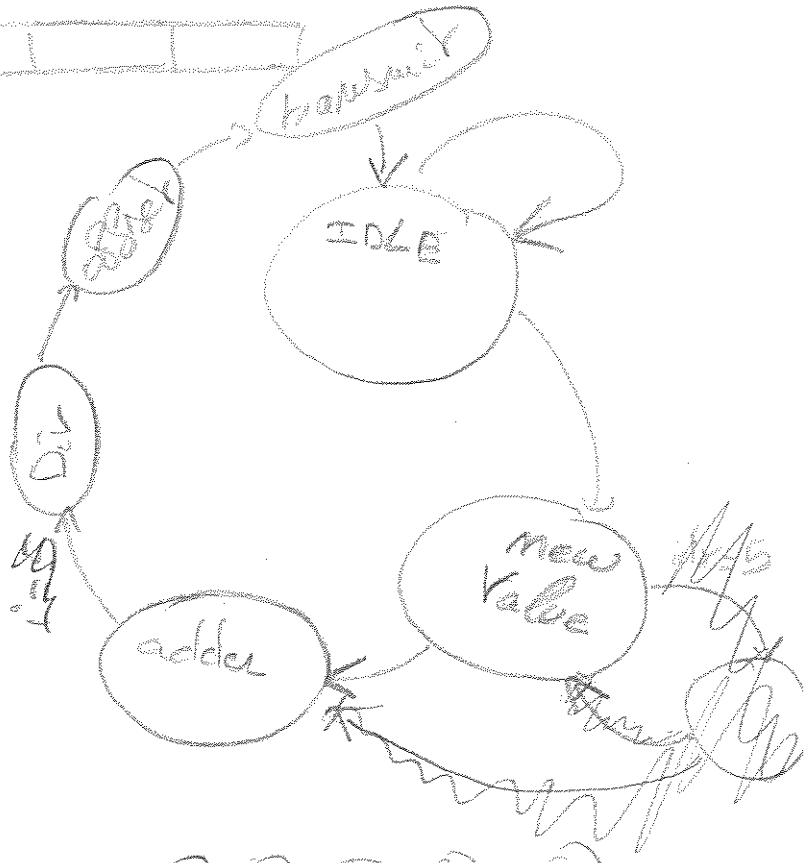
Speed & Position

Echantillonnage

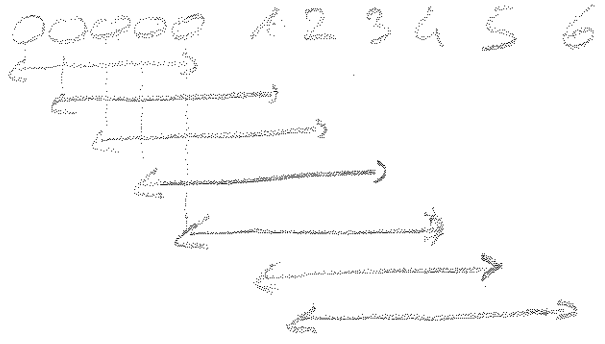


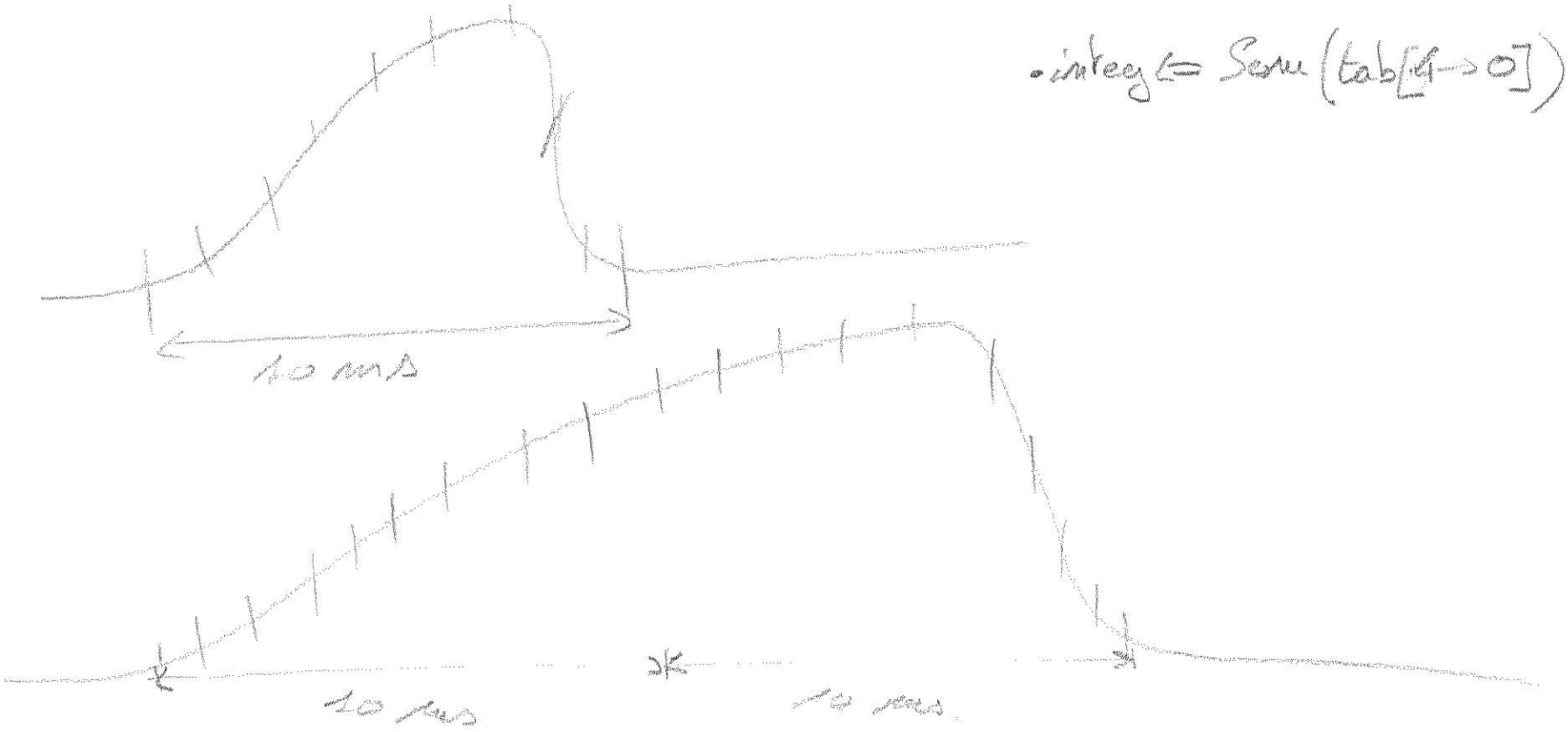

```
if(mval) sg)
  tab[i] <= mVal;
for tab[i] <= 0; i++
i <= i + 1
```



- adder <= sum[tab(0)->tab(9)]
- Div <= adder / adder'length
- offset <= Div + offset
- SpdData <= offset
  SpdOut <= -11.

0	0	0	0	0
1	0	0	0	0
2	1	0	0	0
3	2	1	0	0
4	3	2	1	0
5	4	3	2	1
6	5	4	3	2





• integ <= Sum(tab[4->0])

• On sait que la période minimale entre deux répétitions de trame est de 10 ms.
On choisi ainsi d'intégrer sur 5 valeurs, afin d'être sûr de bien n'voir passer notre signal, et d'avoir un temps de réaction correcte.

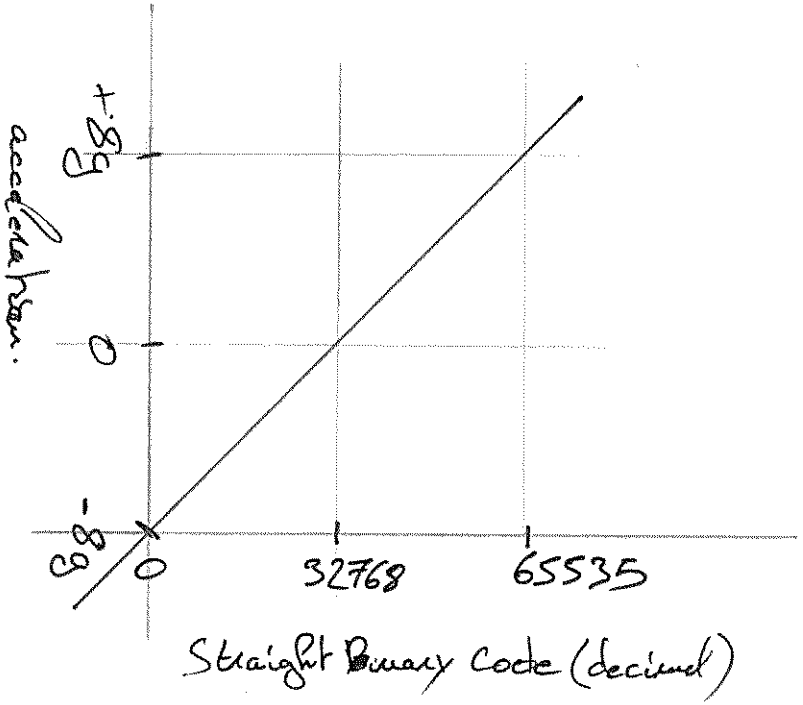
• Intégrer nos valeurs d'accélération et de vitesse, revient à effectuer une moyenne glissante, que l'on a fixer sur 5 valeurs.

-8g	0	+8g
-32768	0	+32767
+offset		
0	+32768	+65535

-8g <=> 0

0g <=> 32768

+8g <=> 65535



ASP_filter.vhd

SPI 3-wire-adaptation SPI_MASTER CPHA = 0
↳ kx224. CPOL = 0

dwidth ⇐ 4|16

- last_bit_rx ⇐ dwidth × 2 + CPHA − 1
  - ⇐ 4|16 × 2 + 0 − 1
  - ⇐ 8|32 − 1
  - ⇐ 7|31

- Bit_READ = 1
- Bit_Write = 0

- assert_data ⇐ not-CPHA
  - ⇐ 1

clk_toggle

bit_Data	clk_toggle	<9			Bit counter
1	0	<9	Tx	xxxx XXX@	0
0	1	<9	Rx	xxxx xxxU	
1	2	<9	Tx	xxxx XX@@	1
0	3	<9	Rx	xxxx xxUU	
1	4	<9	Tx	xxxX @@@@	2
0	5	<9	Rx	xxxx xUUU	
1	6	<9	Tx	xxx@ @@@@	3
0	7	<9	Rx	xxxx UUUU	
1	8	<9	Tx	xxx@ @@@@	4
0	9	~~<9~~	Rx	xxxU UUUU	
1	10	~~<9~~	Tx	xx@@ @@@@	5
0	11		Rx	xxUU UUUU	
1	12		Tx	x@@@ @@@@	6
0	13		Rx	xUUU UUUU	
1	14		Tx	@@@@ @@@@	7
0	15		Rx	UUUU UUUU	
1	16		Tx	xxxx xxxZ/R	8
0	17		Rx	xxxx xxxD	

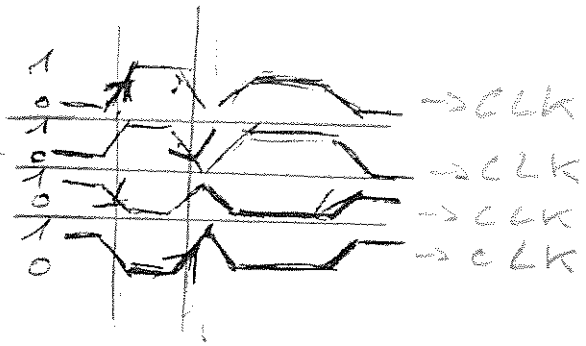
18	Tx	0000 00ZZ/RR	9
19	Rx	xxxx xxDD	
20	Tx	0000 0ZZZ/RRR	10
21	Rx	xxxx xDDD	
22	Tx	0000 ZZZZ/RRRR	11
23	Rx	xxxx DDDD	
24	Tx	000Z/R ZZZZ/RRRR	12
25	Rx	xxxD DDDD	
26	Tx	00ZZ/RR ZZZZ/RRRR	13
27	Rx	xxDD DDDD	
28	Tx	0ZZZ/RRR ZZZZ/RRRR	14
29	Rx	xDDD DDDD	
30	Tx	ZZZZ/RRRR ZZZZ/RRRR	15
31	Rx	DDDD DDDD	
32	Tx	ZZZZ ZZZZ	16
33	Rx	DDDD DDDD	
~~34~~	~~Tx~~		
~~35~~	~~Rx~~		
~~36~~	~~Tx~~		

CPOL (Clock Polarity)
CPHA (Clock Phase)

Ces options à ces 2 paramètres, on détermine les fronts où les données sont transmises (acquisition)
et des moments où elles peuvent être modifiées.

- CPOL → détermine si au repos l'horloge est au niveau bas = 0
- CPHA → détermine à quel front de l'horloge les haut = 1 données sont transmises.
  - données valide 1er front d'horloge = 0
  - " " 2e " " = 1

SPI MODE	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



Accéléromètres CPOL - CPHA

ADXL355 : 0 / 0
AIS3624 : 1 / 1
KX224 : 0 / 0

Adaptation du SPI-Master pour la prise en compte des trames de l'accéléromètre du Kionise - KX224.

⚠ Le problème majeur nécessitant cette "mise à niveau" étant que le KX224 possède une trame de données "atypique".

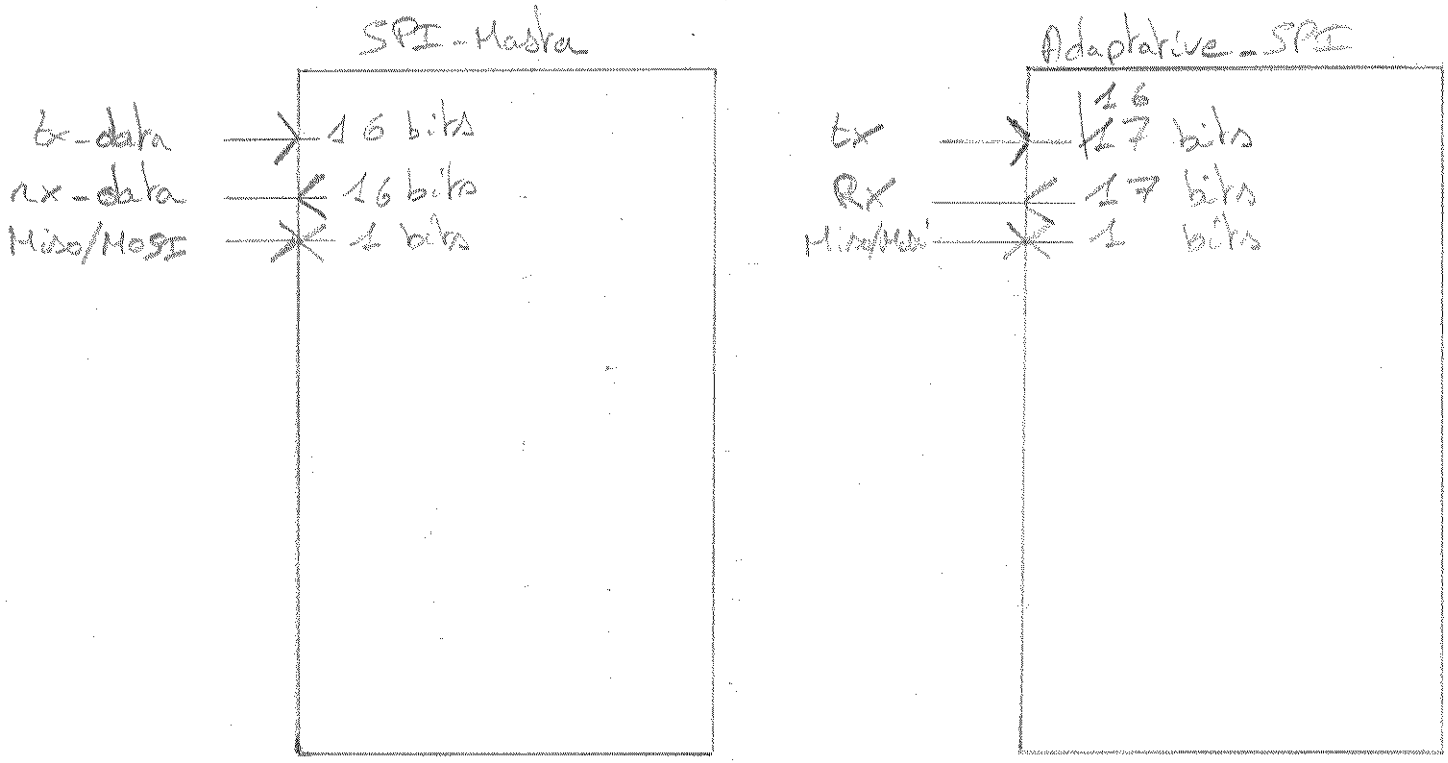
Pour l'envoie, nous avons une trame classique sur 16 bits

W + @ + Data
(1 bit) (7 bits) (8 bits) = (16 bits)

Pour la réception, nous avons une trame qui se voit rallongée de 1 bit.

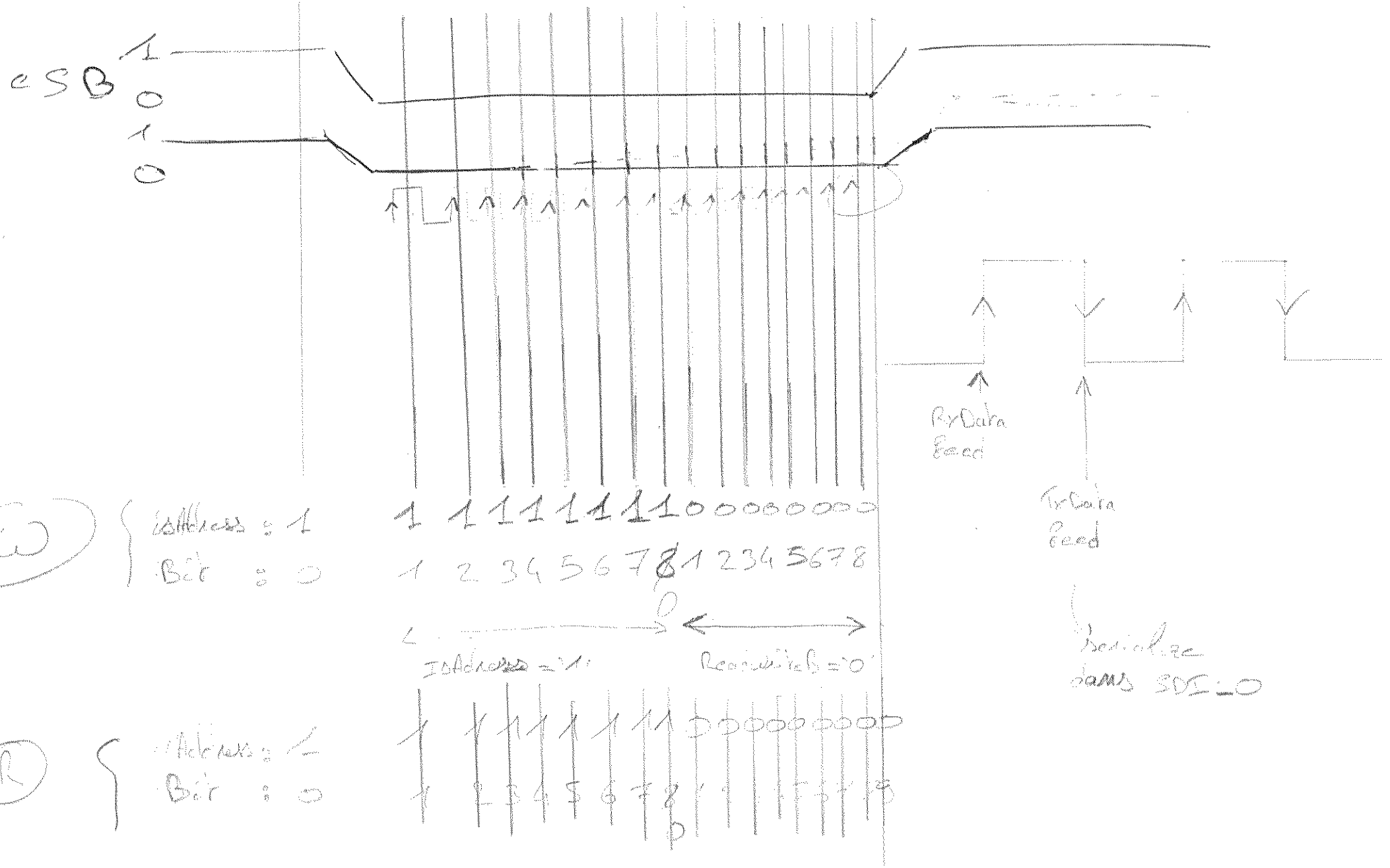
R + @ + EXTRA-BIT + DATA
(1 bit) (7 bits) (1 bit) (8 bits) = (17 bits)

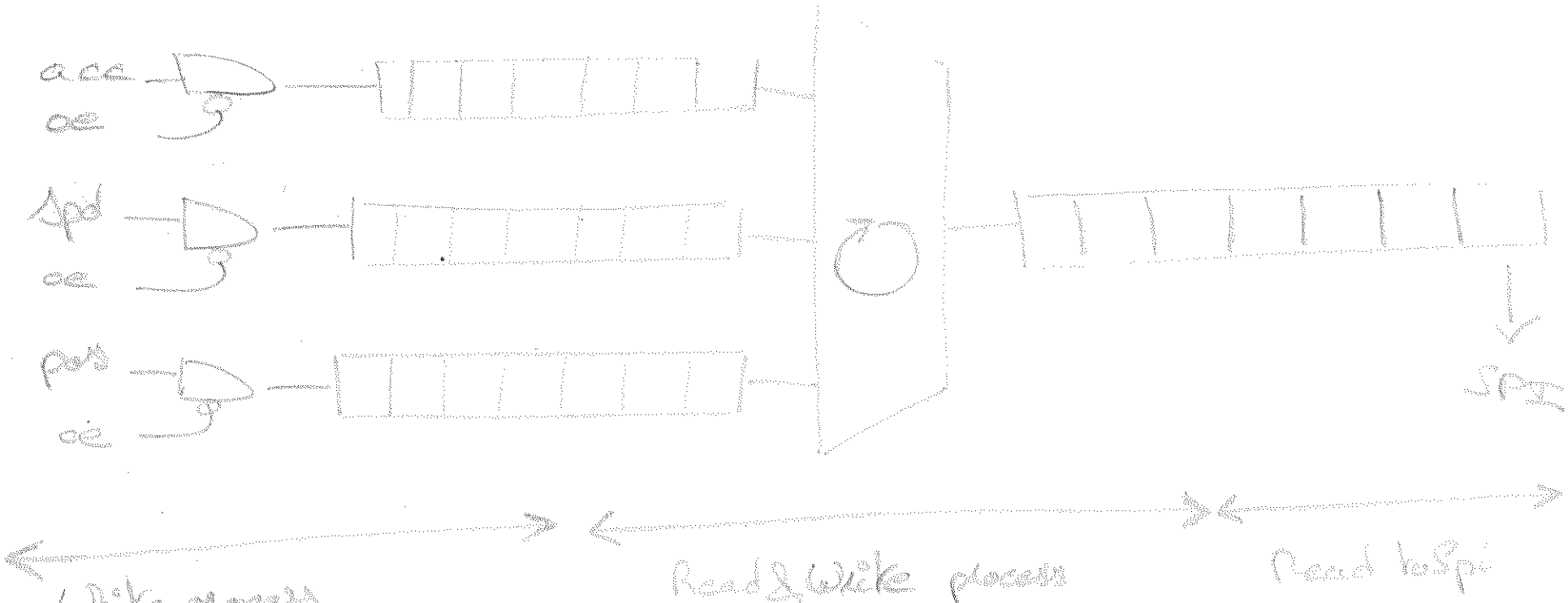
SPI 3-wire
Adaptation protocol SPI
accéléromètre KX224.



On se retrouve avec 2 entitées différentes, une gérant les envois classiques (même nbr de bits pour les trame ~~TX~~ et RX), et une deuxième permettant de s'adapter au cas particulier du Kionix KX224, cette dernière ayant une trame écriture (W) sur 16 bits, et une trame de lecture (R) sur 17 bits.

Schéma protocole SPI, tel que défini dans le fichier "SPI_Master" et adapté au KX224.





⚠ 1 process par ligne de données.

⚠ Chaque FIFO sera un mot de 24 bits, composé comme suivant :

command ⊕ ⊕ ⊕ Data.

acc dpd pos

Template.com

Write process (clk50)

Process Acquisition/ipad/fbis

IDLEst

if acc_ce = '1'

IP_FA.wire_ce <= '1'

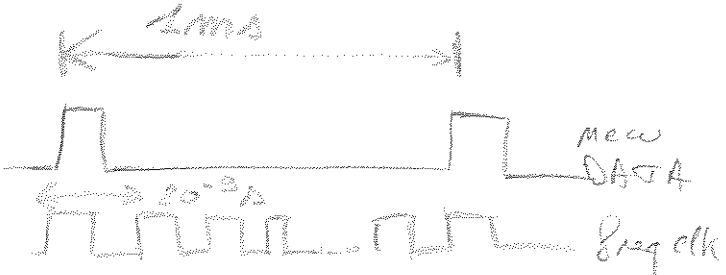
eState <= WRITEst;

WRITEst

fifo_write <= com + @ + data;

IP_FA.wire_ce <= '0';

eState <= IDLEst;



×3

+Full control

Process R&W (clk50)

IDLEst

for I in 0 to 2

if FIFO(I).empty = !empty

fifo_ce_N(I) <= '1';

DACFIFO.ceWire <= '1';

CPDATAst

DACFIFO.ceWire <= '0';

DACFIFO.wire <= FIFO(I).read;

fifo_ce_N(I) <= '0';

INCREMENTst

I := I + 1

if I <= 2

fifo_ce_N(I) <= '1' & DACFIFO.ceWire <= '1';

CPDATAst

else

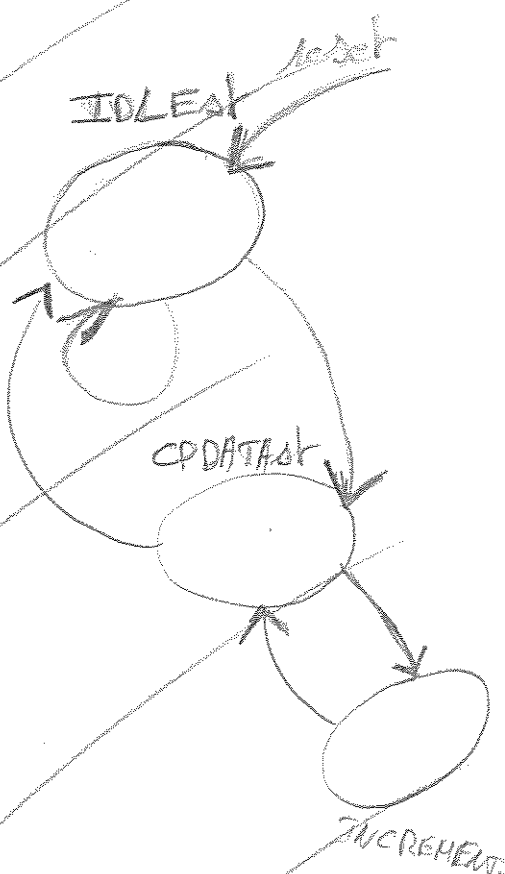
IDLEst

+if I < 2

INCREMENTst

else

IDLEst

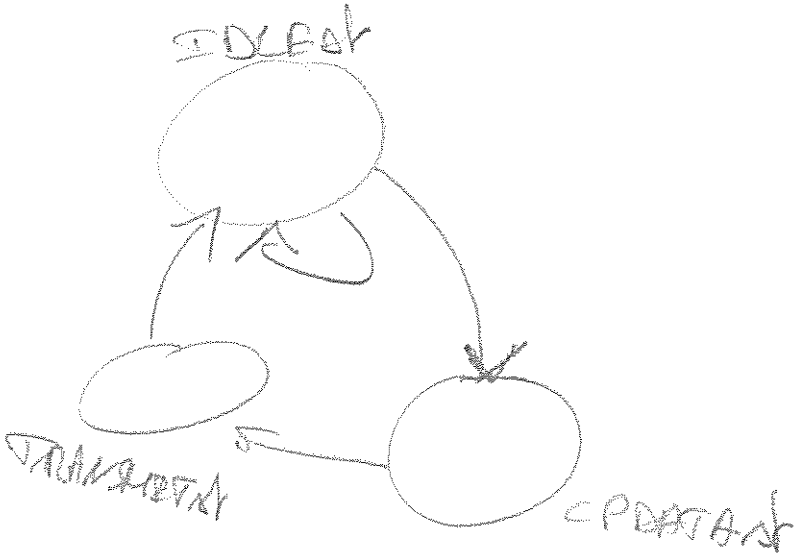


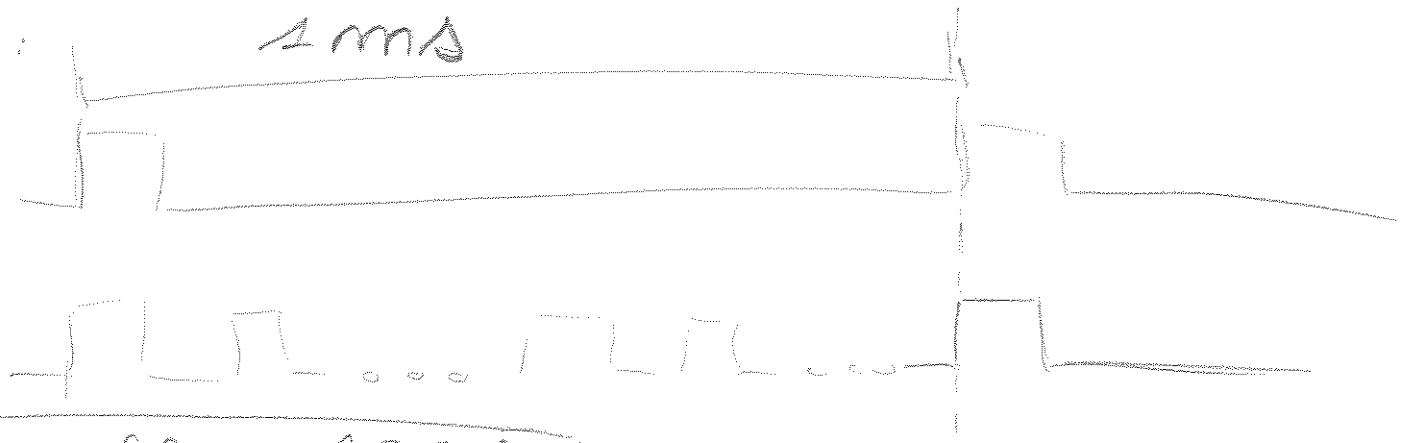
Process ReadtoSpi

IDLEst
- if DACFIFO.empty = !empty
  - DACFIFO.oe.R <= '1'; @spi.enable = '1';
  - CPUSt; / @DAC_oe.output = '1';
- else
  - IDLEst;

CPUSt
- spi_tx(0:5 -> 0) <= DAC_FIFO.Read();
- m_tx(19 -> 16) <= 2;
- "1" & "0" (23 -> 20) <= command;
- TRANSMITSt;

TRANSMITSt
- DAC_FIFO.oe.Read;
- spi.enable <= '0';
- DAC_OE.output <= '0';
- IDLEst;





Process RR (clock 50)

IDLEst

if (fifo(idx).isempty = !empty)

fifo_re_n(idx) <= '1';

DAC_FIFO_we_w <= '1';

EPOATAst

else

*

EPOATAst

DAC_FIFO_w <= FIFO(idx).read;

fifo_re_n(idx) <= '0';

DAC_FIFO_we_w <= '0'

if idx <= 2

INCREMTst

else

IDLEst

INCREMTst

idx = idx + 1;

if idx =< 2

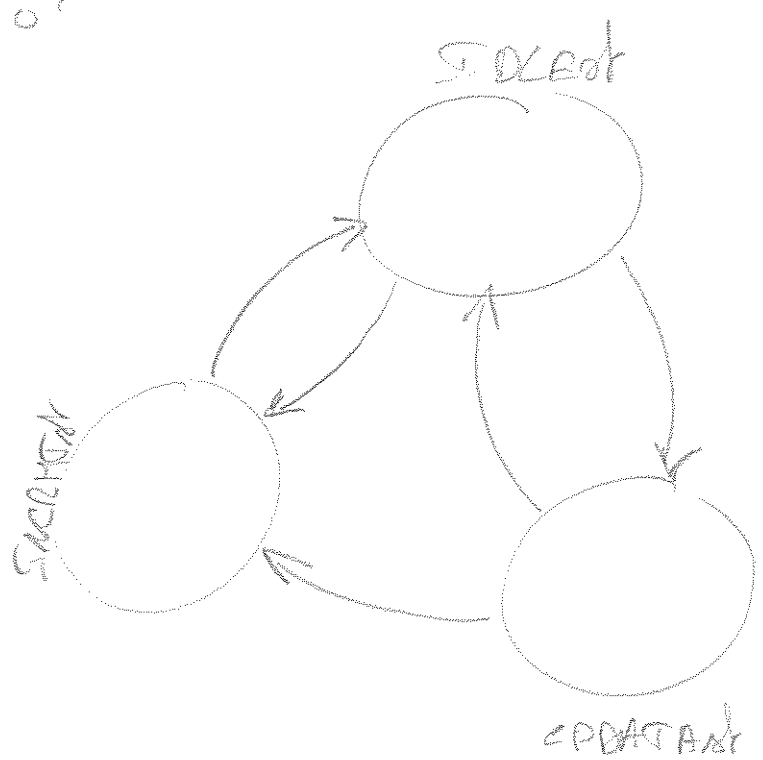
IDLEst

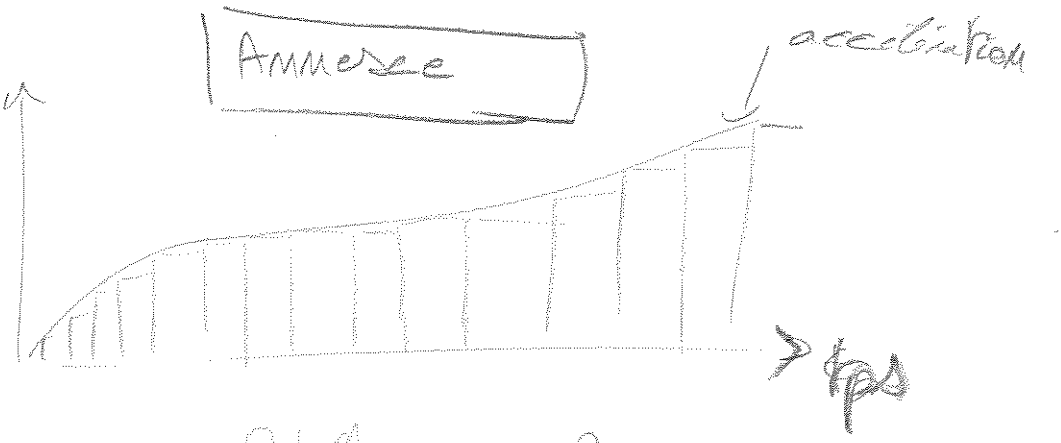
else

idx = 0;

IDLEst

* INCREMTst





- Integ = somme de petits carrés

$$\begin{cases} acc = a \\ v = ax + v_0 \\ p = \frac{1}{2}ax^2 + vx + p_0 \end{cases}$$

- Algorithme :

> speed = acc × Δt + speed

acc data

time between 2 measures (constant)

Previous speed value calculated

> pos = 1/2 acc × (Δt)² + speed × Δt + pos

speed previously calculated

acc data

time between 2 measures

pos calculated

$300\,000 = \underbrace{0,3}_{\text{Mantisse}} \times 10^{6}$ ← exposant

- notation scientifique par binaire ⟺ format flottant

32 bits

S	E	M
1	8	23

exe :

	S	M	décimal complété à	E
• 12 ⇒	0	1100		2^0
sans virgule, on complète la partie décimale à droite	0	1,	1000	2^3 → 110 = 3+3 → +3 demandé par la norme pour les nbs à 8 bits

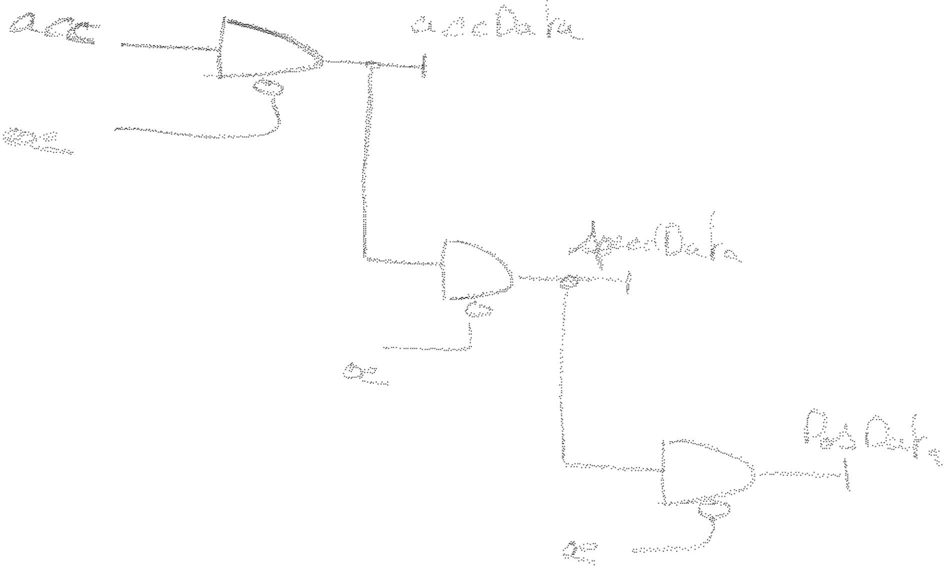
⇒ 0 110 1000 (S E M)

	S	M	déc	E
• 2,625	0	10	0,625 × 2 = 1,25 0,25 × 2 = 0,5 0,5 × 2 = 1 → 101	2^0
avec virgule, on décale les zéros de la mantisse à la partie décimale		1,0	→ 0101	2^1 → +3 0100

⇒ 0 100 0101 (S E M)

VHDL

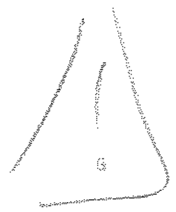
- real ⇒ pour constante en simulation, mais non valide en VHDL synthétisable
- virgule flottante ⇒ idem



Sur Architecture du Si..., ~~[illegible]~~

ajouter : > les données calculé : acc
speed
position

> les acc necessaire pour : speed
position



Comme observé, le calcul à virgule n'est pas réalisable ou négligeable.

Ainsi, par les div, il va nous falloir nous rapprocher des puissance de 2.

Par exemple,

> Multi par 0,001 revient à diviser par 1000.
On se rapprochant de la puissance de 2 la plus proche,
$2^{10} = 1024$

=> On préfèrera donc diviser par 1024

ex : $\frac{5100}{1000} = 5,1$ → erreur à 6%

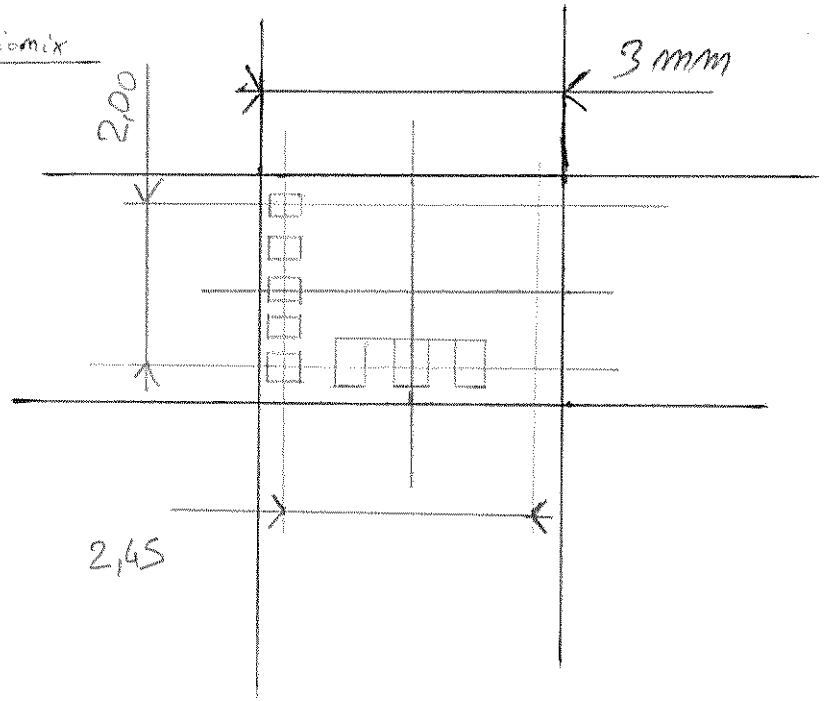
$\frac{5100}{1024} = 4,98$

=> Speed : a × Δt + v
⇔ a × 0,001 + v
⇔ a ÷ 1000 + v
⇔ a ÷ 1024 + v

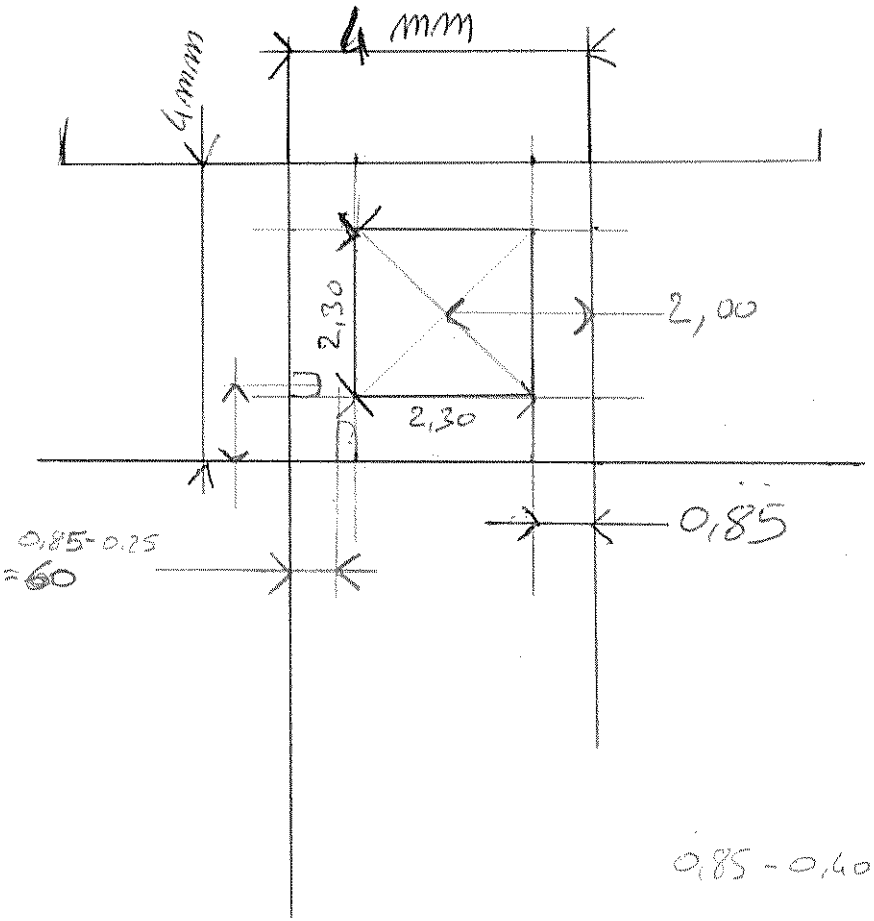
Div par 1024 ⇔ decalage à droite de 10.
ex :
5000
↳ 1001110001000
$\frac{5000}{1024} = 4,88$

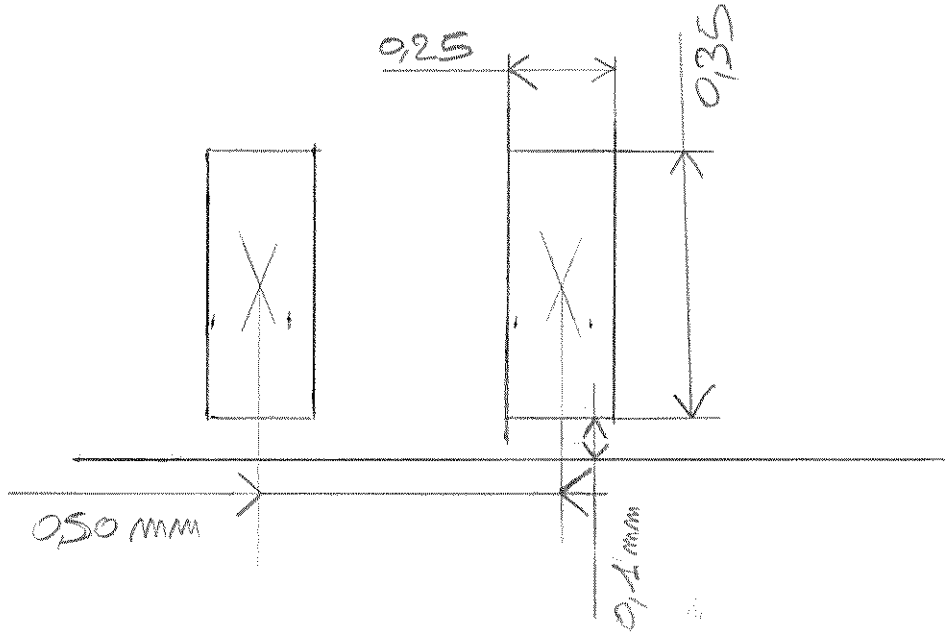
So, don't have to devide by 0,01 see next page.

KX224-kiomix

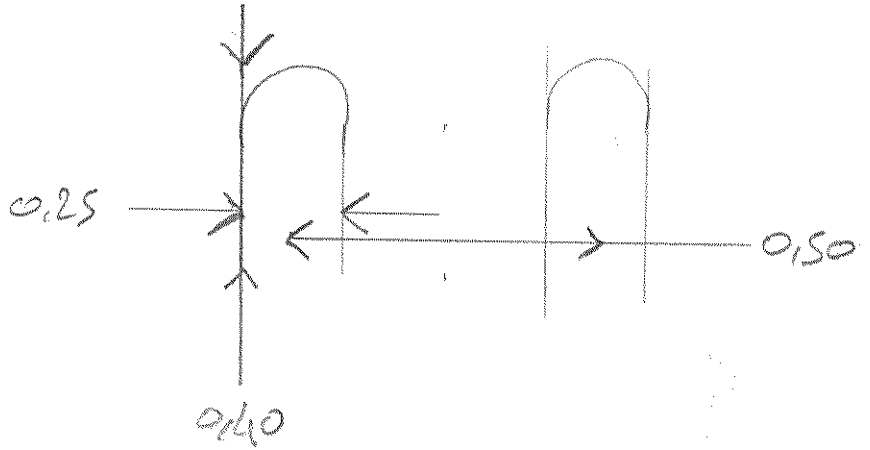


ST_AIS3624



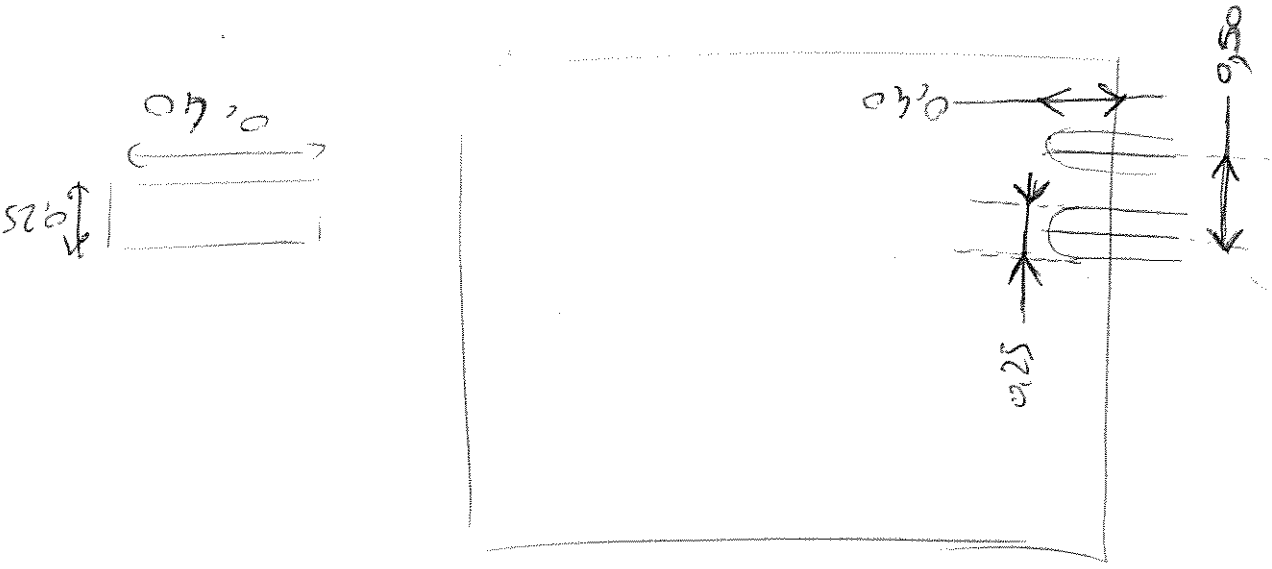


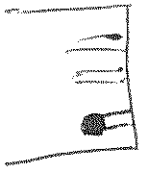
0,85 - 0,40 = 0,45

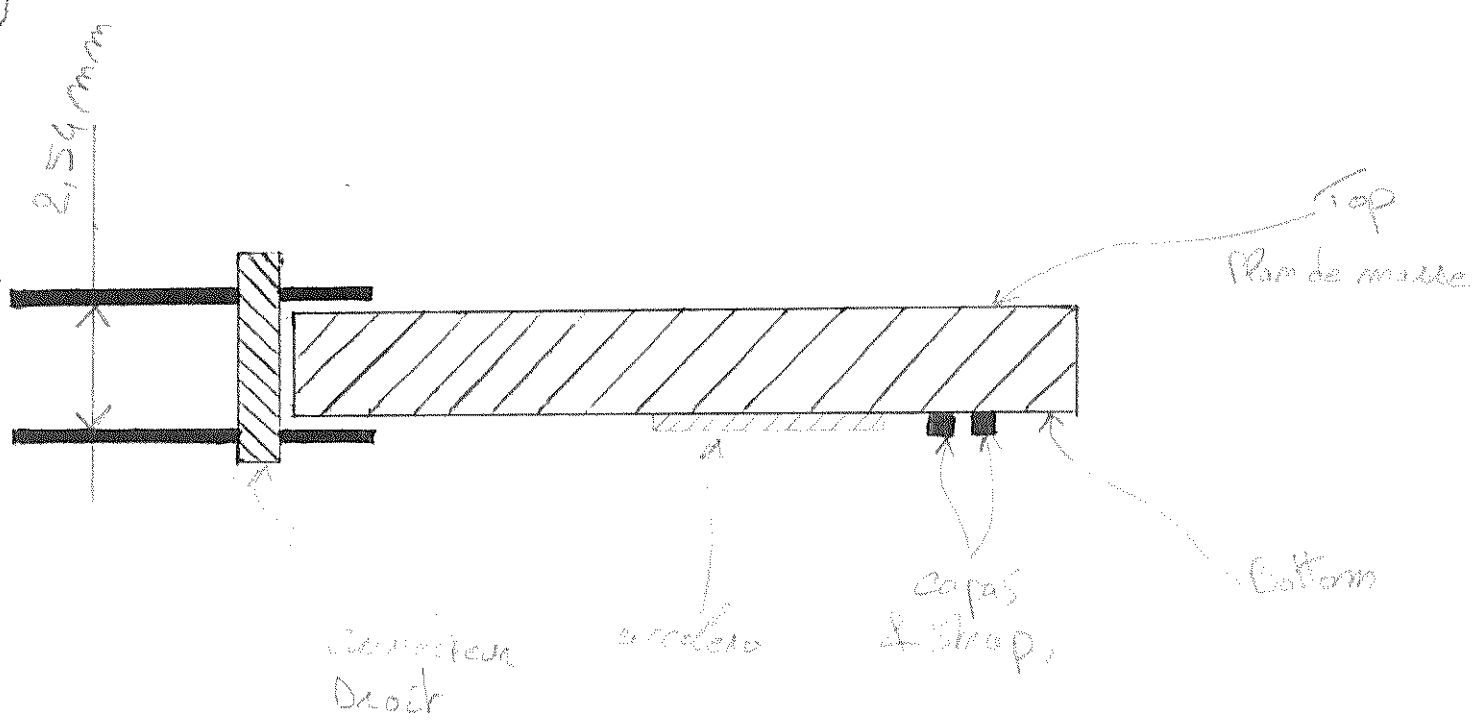


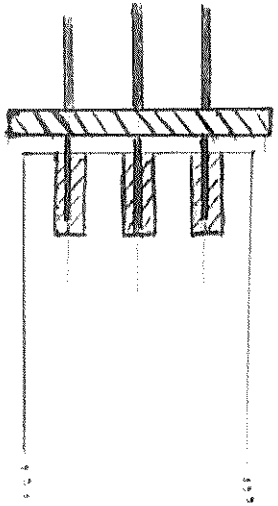
• didier gerard

dgerard@[illegible]europe.com





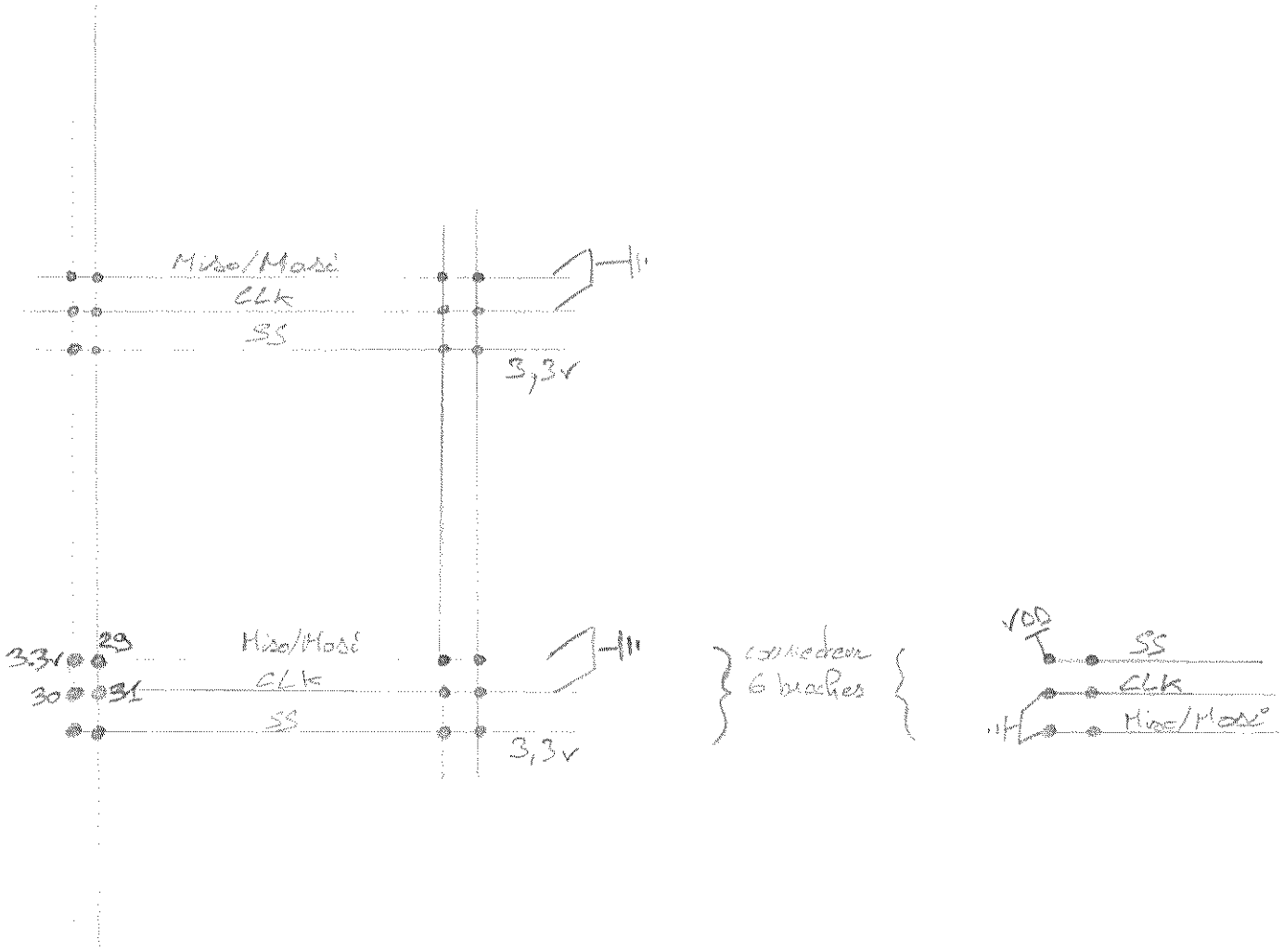




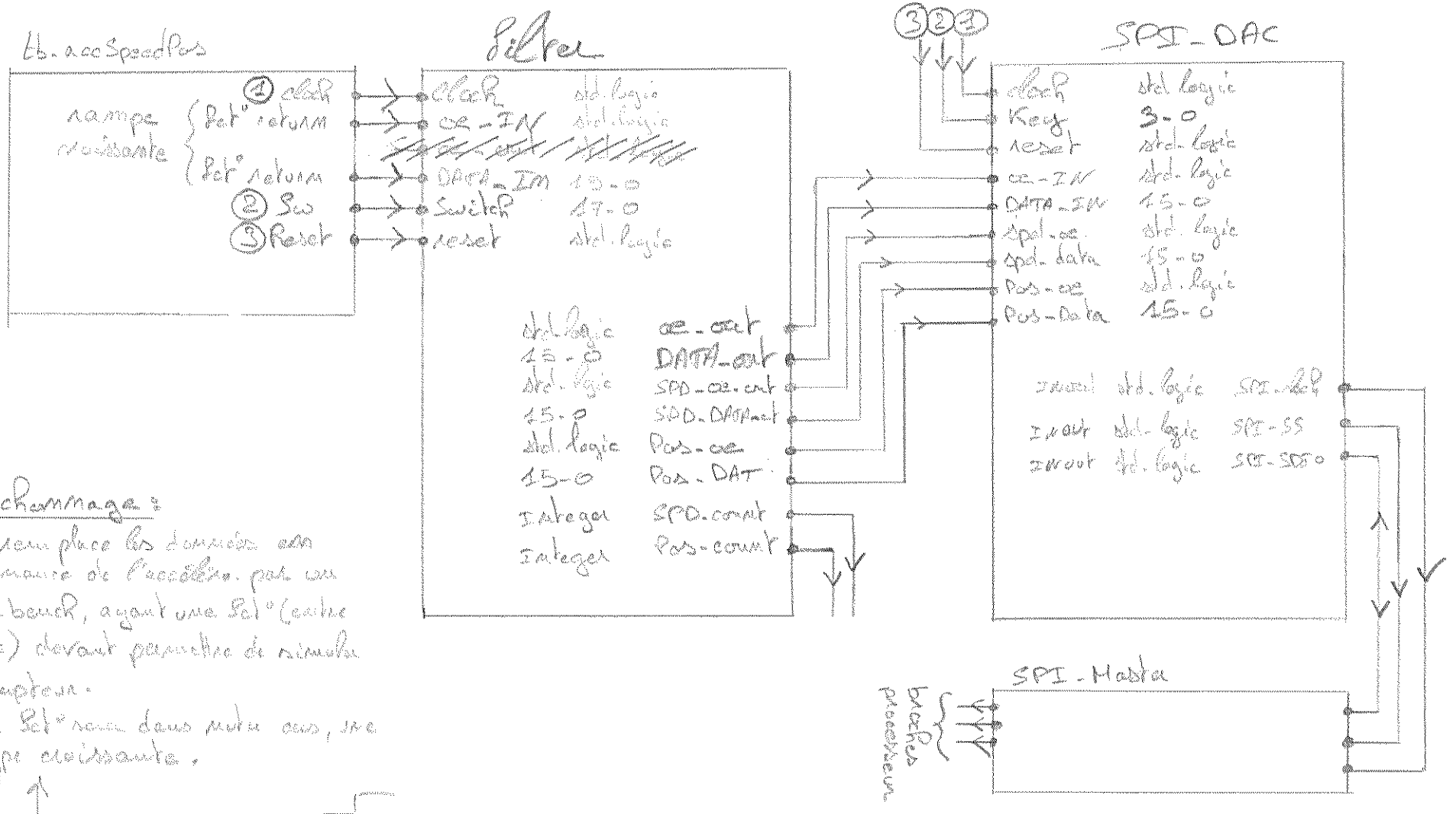
1) Suite à une réunion avec Philippe REPAIN, il est apparu que ce design ne peut être retenu pour cause des points suivants :

- obligé de rallonger le support imprimé → augmentation du poid de la pièce.
- impossibilité d'augmenter la longueur de la pièce, dû à l'espacement entre les touches du piano.

Cablage Accelero & Mezza

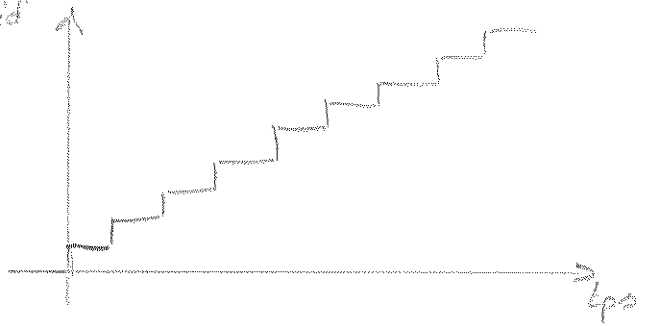


Architecture de haut niveau du projet

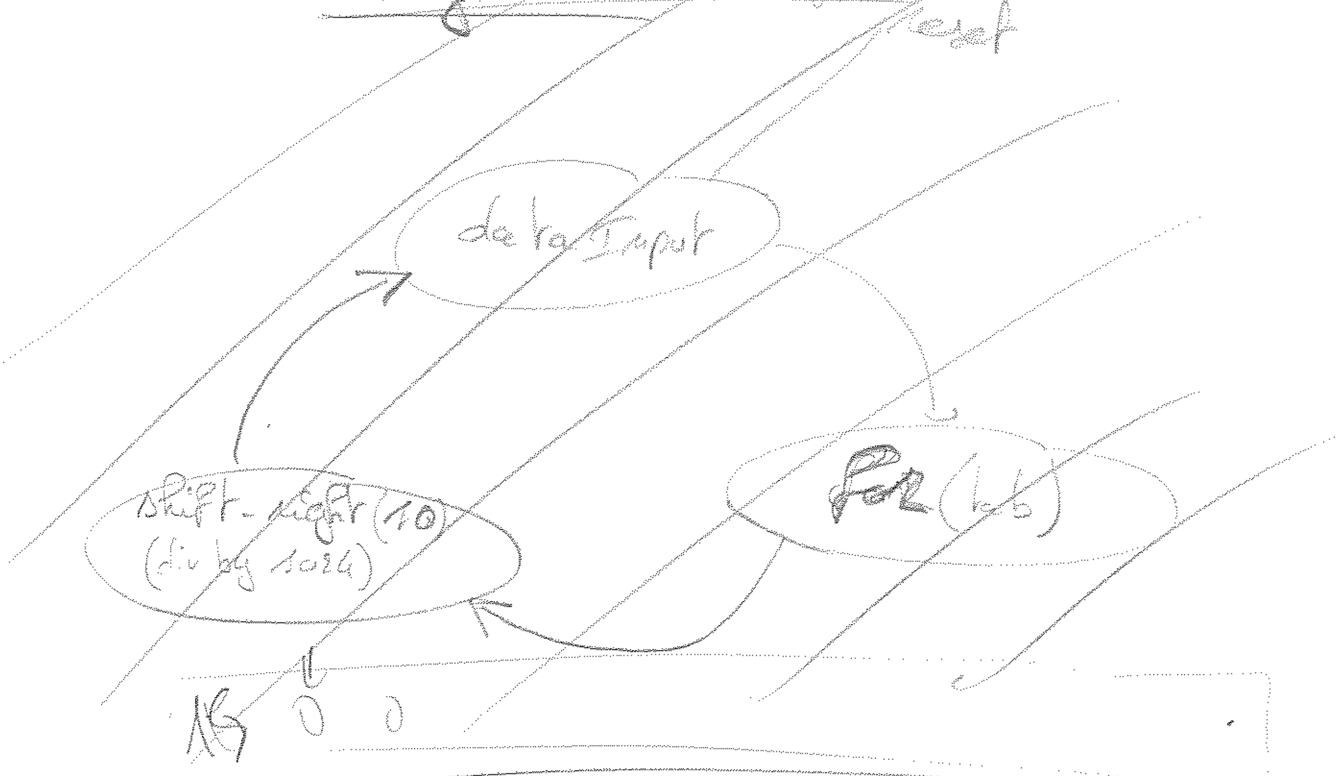


Bouchonnage :

On remplace les données en provenance de l'accéléro. par un test_bench, ayant une Sel° (entre autre) devant permettre de simuler le capteur.
Cette Sel° sera dans notre cas, une rampe croissante.



Moyenne classique



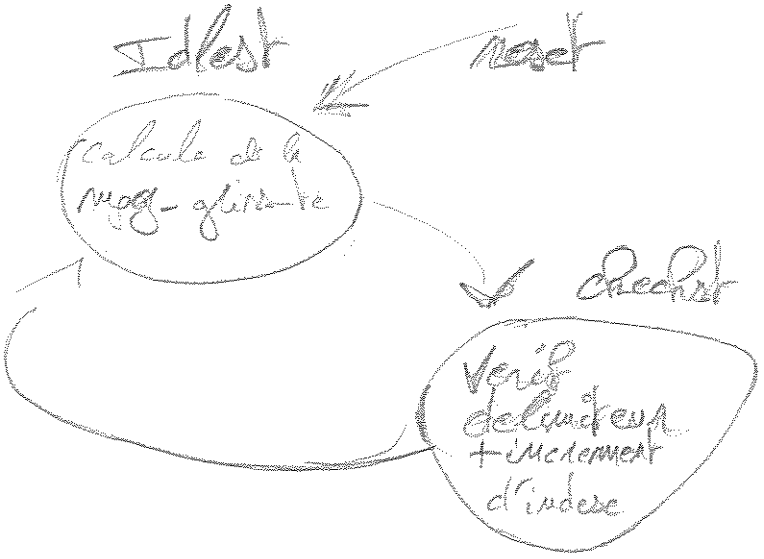
algo de Moy glissante

```
IRQ()
    sample = newData();

    avg <= avg + sample - hist[mostOldValue];
    hist[mov] <= sample;

    mov = mov + 1;
    if (mov == N)
        mov = 0;
```

⚠ Penser à l'init des data
- Tableau
- variable etc...

- 15
  avg = 0 + 15 - 0
  | 15 | 0 | 0 | 0 | 0 | 0 |
  mov = 1
- 22
  avg = 15 + 22 - 0
  | 15 | 22 | 0 | 0 | 0 | 0 |
  mov = 2
- 12
  avg = 37 + 12 - 0
  | 15 | 22 | 12 | 0 | 0 | 0 |
  mov = 3
- 7
  avg = 49 + 7 - 0
  | 15 | 22 | 12 | 7 | 0 | 0 |
  mov = 4
- 1
  avg = 56 + 1 - 0
  | 15 | 22 | 12 | 7 | 1 | 0 |
  mov = 5
- 20
  avg = 57 + 20 - 0
  | 15 | 22 | 12 | 7 | 1 | 20 |
  mov = 6
  mov = 0
- 19
  avg = 77 + 19 - 15
  | 19 | 22 | 12 | 7 | 1 | 20 |
  mov = 1
  ⋮

State Machine Moyenne Mobile




```
case cState is

  when reset =>
    for I in 0 to I loop
      Fifo[I] <= 0;
    end loop;
    cAvg <= 0;
    cMov <= 0;
    cState <= Idlest;

  when Idlest =>
    cAvg <= cAvg + NewData - Fifo[cMov];
    Fifo[cMov] <= NewData;
    cState <= checkst;

  when checkst =>
    if cMov >= TabMaxSize - 1 then
      cMov <= 0;
    else
      cMov <= cMov + 1;
    end if;
```

variable