FIRST / SPIRE / SMEC

MCU SOFTWARE

Prepared by :	Date :
Patrick Levacher Didier Ferrand	20-06-2000
Checked by :	**Date :**
Dominique Pouliquen	

Centre National de la Recherche Scientifique
Laboratoire Astrophysique de Marseille
BP 8 - 13376. Marseille Cedex 12.
Tel. : 33 .4.91.05.59.00 - Fax : 33 .4.91.66.18.62

Document change record

_____

Revision			Modified Pages
Issue - rev.	Date	Approved by	
01 - 00	25-04-00		All : first issue

Centre National de la Recherche Scientifique
Laboratoire Astrophysique de Marseille
BP 8 - 13376. Marseille Cedex 12.
Tel. : 33 .4.91.05.59.00 - Fax : 33 .4.91.66.18.62

Distribution list

———————

Jean Louis Auguères	CEA	X	
Armand Artinian	LAM	X	
Christophe Cara	CEA	X	
Claude Colin	LAM	X	
Colin Cunningham	ROE	X	
Didier Ferrand	LAM	X	
Michel Jevaud	LAM	X	
Ken King	RAL	X	
Patrick Levacher	LAM	X	
Dominique Pouliquen	LAM	X	

Acronyms

AD	Applicable Document
AVM	Avionics Model
BOL	Begin Of Life
BSM	Beam Steering Mirror
CQM	Cryogenic Qualification Model
EGSE	Electrical Ground Support Equipment
EOL	End of Life
ESA	European Space Agency
FIRST	Far InfraRed ans Sub-millimetre Telescope
FM	Flight Model
FPU	Focal Plane Unit
FTS	Fourrier Transform Spectrometer
FTSE	FTS warm Electronics
FTSP	FTS Preamplifier for the position encoder signals
H/K	House Keeping
H/W	Hardware
I/F	Interface
LAM	Laboratoire Astrophysique de Marseille
MAC	Multi Axes Controller
N/A	Not Applicable
RD	Reference Document
ROE	Royal Observatory of Edinburgh
S/C	Spacecraft
SM	Spare Model
SMEC	Spectrograph MEChanism
S/W	Software
TBC	To Be Confirmed
TBD	To Be Define
TBW	To Be Written
TC	Tele-Command
TM	TeleMetry
WE	Warm Electronics
…	

Table of contents

1.1 Control Algorithms

1.1.1 DSP Software

The DSP software shall be written in 21020 assembly language without the use of a specific off-the-shelf real time operating kernel. The main tasks to be realized shall be called by a Master scheduler which is a routine interrupted by a software interrupt generated by the inner timer of the DSP.
Mainly, the software shall not use other interrupt, excepted for the the following functions :

- watchdog interupt to recover from a loose of control of the DSP,
- interupt when the DSP does'nt answer to a query from the LINK command/parameter

Task name	Call type	duration	Sampling rate	Task function
MAC Master Scheduler	on DSP internal timer event		100 us	call for other MAC tasks depending on the timer value
SMEC Servo Loop	on Master Scheduler request		100 us	servo loop on encoder signal
BSM Chopping axis Servo loop	*on Master Scheduler request*		*100 us*	*servo loop on magnetoresistive sensors*
BSM Jiggle axis Servo loop	*on Master Scheduler request*		*100 us*	*servo loop on magnetoresistive sensors*
Read_command	polling sampling time		100 us	Read the buffer of commands and takes dedicated actions
Motor current acquisition	TBD polling		TBD	Read the motor current value and build the HK telemetry packet
MAC Status control	on Master Scheduler + IT for emergency action		TBD	Update the state of the DSP software + error word

1.1.2 Mac Master Scheduler Algorithm

The principle of the MMS Algorithm is the management of the various tasks to be performed in the Mac DSP. Each task has a 100us sampling time value. The dedicated function is called when its specific task counter reaches the sampling time value, as a multiple number of the base master scheduler sampling time.
For a 20MHz DSP clock, there are 400 DSP cycles during one base Master Scheduler sampling time (20us)

1.1.3 General considerations

For ADCs we use AD9221 with their own clock, that is the clock of DSP divided by 8 (400ns). The datas are always available on their output register so we don't order a start of conversion with theses devices. There is a 4 level pipeline in theses ADCs so we read a data that is the 4th oldest sample. This gives a delay of about 2us for the control loop, that is not a problem for our frequency control.
The following estimations of number of cycles are given by Analog Device application notes, for example the number of cycles for a FIR Filter is given by : number of cycles=7+ number of taps used for filtering.
The duration was computed with a 20MHz clock, so 1 DSP cycle=50ns

1.1.4 SMEC Servo Loop algorithm

step	function	Nb cycles DSP	Duration	observation
1	Read sine & cosine	1	50ns	One cycle only for read because of AD9221, and because we read in the same word the 2 ADCs
2	Filtering sine	40	2us	Ref Analog Device applications : the number of cycles for a FIR filter is given by : 7+number of taps used for filtering
3	Filtering cosine	40	2us	
4	Compute arctangent	120	6us	Ref ADSP-21000 Family Application Handbook : The compute of **Arctangent** is given for **82 cycles** maximum from Analog Devices (ref ADSP-21000 Family Application Handbook), but we must reserve about 40 cycles for overflow and error tests.
5	Compute current trajectory	40	2us	That is the same that a speed filtering, so the filter estimation is correct
6	Compute current error	40	2us	Another thing to do for this step « do somethig » if error>max error
7	Compute correction and the output DAC for FTS motor amplifier	20	2us	the compute of PID is given in 10 cycles (ref: DSP-Based Motor Controller Seminar of Analog Devices) These numbers don't include error test , overflow…
8	Write output DAC	1	50ns	

TOTAL 302 cycles 16.1us

1.1.5 BSM Chopping axis Servo loop

step	function	Nb cycles DSP	Duration	observation
1	Set anolog Multiplexer	1	50ns	set analog multiplexer to BSM chop position magneto-resistive sensor
2	read position	1	50ns	
3	Filtering position	40	2us	
4	compute current error	40	6us	Another thing to do for this step is « do something » if error>max error
5	Compute correction and the output DAC for BSM chopping	40	2us	
6	Write output DAC	1	50ns	

TOTAL 83 cycles 10.1us

1.1.6 BSM Jiggle axis Servo loop

step	function	Nb cycles DSP	Duration	observation
1	Set anolog Multiplexer	1	50ns	set analog multiplexer to BSM jiggle position magneto-resistive sensor
2	read position	1	50ns	
3	Filtering position	40	2us	
4	compute current error	40	6us	Another thing to do for this step is « do something » if error>max error
5	Compute correction and the output DAC for BSM jiggle	40	2us	
6	Write output DAC	1	50ns	

TOTAL 83 cycles 10.1us

1.1.7 Watchdog mechanism

1.1.7.1 DIGITAL I/O POLLING

1.1.7.2 RESET

1.2 PID Control algorithm (ref Analog Device AN401-13, AN 401-33)

1.2.1 Discretize the linear PID

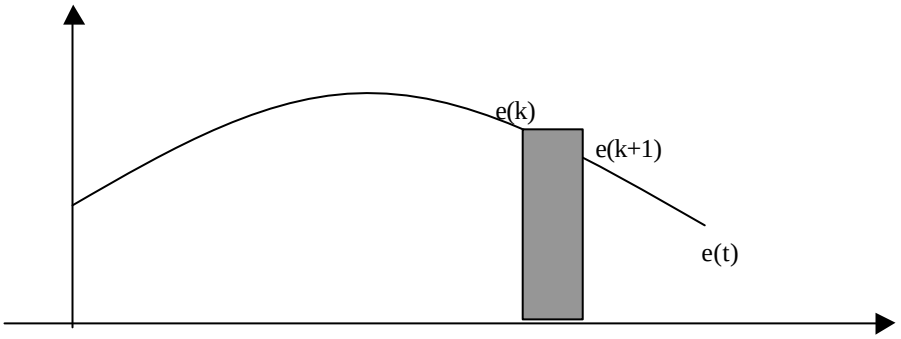
One basis form of linear PID is PID with filtering derivative :

$$\frac{Y(s)}{X(s)} = K(1+\frac{1}{T_i s}+\frac{T_d s}{1+\frac{T_d}{n}s})$$

Usually, we use 2<n<10.To implement it on 21020 , we need to discretize this equation and there are at least 2 methods to do this , corresponding with 2 methods to do a numerical approximation of the integral. Theses 2 methods are implemented in MATLAB/SIMULINK. Under MATLAB, if we want to compute a Z transform from a Laplace transfert function, we need to notify the 'ZOH' or 'FOH'method

1.2.1.1 1ST METHOD : ZERO ORDER HOLD (ZOH , OR EULER'S METHOD)

It's the simplest method. We approximate the integral $\int_0^t e(t)dt$ with the oldest value :

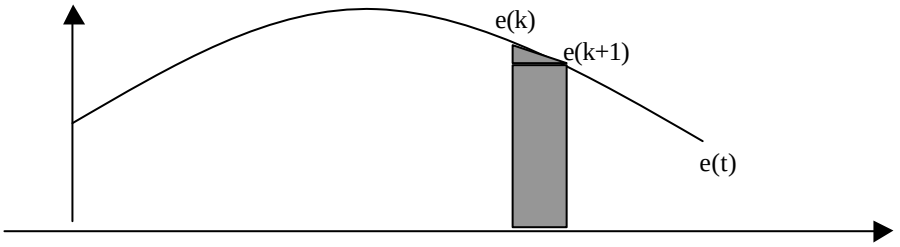


In this case we discretize the integral $\int_0^t e(t)dt$ by accumulating the rectangular areas wich corresponds to the following difference equation : $\sum_{k+1} = \sum_k + T_s e(k)$ where $\sum_k$ = value of the approximative integral for t=k*T_s and T_s is the sampling time. If we considere the Z transform, we have the following equation :

$\sum(z) = \frac{T_s}{z-1}E(z)$ and we can find the Z transform from a Laplace transfert function by replacing the Laplace operator s by $\frac{z-1}{T_s}$

We sometimes see (for example in Analog Device seminar control motor) the approximation of the rectangular areas with the newest value. Similarly, we replace in this case the Laplace operator s by $\frac{z-1}{zT_s}$

1.2.1.2 2ND METHOD : TRAPEZOIDAL APPROXIMATION OR FIRST ORDER HOLD (FOH)



In this case we discretize the integral $\int_0^t e(t)dt$ by accumulating the trapezoidal areas wich corresponds to the following difference equation : $\sum_{k+1} = \sum_k + T_s \frac{e(k+1)+e(k)}{2}$ .If we now considere the Z transform, we have the following equation : $\sum(z) = \frac{T_s}{2}\frac{z+1}{z-1}E(z)$ and we can find the Z transform from a Laplace transfert function by replacing the Laplace operator s by $\frac{2}{Ts}\frac{z-1}{z+1}$, sometimes named TRUSTIN formula

1.2.1.3 APPLICATION TO THE PID WITH FILTERING DERIVATIVE

We can mixe the methods. For example, if we come back to the linear PID formula given in 1.2.1, we can take the ZOH method for the integral effect and the TRUSTIN formula for the approximate derivative (ref : « commandes des systemes dynamiques », Thelliez/Vergé/Jaume) and we have :

$$\frac{Y(z)}{X(z)} = K\left(1 + \frac{T_s}{T_i(z-1)} + \frac{2\frac{T_d}{T_s}\frac{z-1}{z+1}}{1+2\frac{T_d}{nT_s}\frac{z-1}{z+1}}\right)$$

after ordonning, the expression becomes :

$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$, so a second order IIR with the coefficients given by :

$$b_0 = K\left(\frac{2nT_d}{nT_s + 2T_d} + 1\right) \qquad b_1 = K\left(\frac{T_s}{T_i} + \frac{nT_s - 2T_d - 4nT_d}{nT_s + 2T_d} - 1\right)$$

$$b_2 = K\left(\frac{T_s}{T_i}\frac{nT_s - 2T_d}{nT_s + 2T_d} + \frac{2nT_d - nT_s + 2T_d}{nT_s + 2T_d}\right)$$

$$a_1 = 1 - \frac{nT_s - 2T_d}{nT_s + 2T_d} \qquad a_2 = \frac{nT_s - 2T_d}{nT_s + 2T_d}$$

1.2.2 Difference equation

In order to implement this in 21020, we must find the difference equation from the Z transfom formula. We have seen that the Z transform of a PID gives a second order IIR (the PID always gives a second order IIR for all the expressions and all the used numerical methods). The direct form of the difference equation of this filter is immediatly given by the Z transform :

$$y(n) = b_0 x(n) + b_1 x(n-1) + b2x(n-2) + a_1 y(n-1) + a_2 y(n-2)$$

However, this direct form is not commonly used in IIR filter design and Analog Device (ref : Application Handbook 21020) recommands to use the cannonical form for the buffers optimization. We find this cannonical form by introducing an intermediate variable W(z) :

$$X(z) = (1 - a_1 z^{-1} - a_2 z^{-2})W(z) \quad \text{and} \quad Y(z) = (1 + b_1 z^{-1} + b_2 z^{-2})W(z)$$

Notice that we have nomalized the transfert function with the gain 1 (b_0=1). If this gain is not 1, we must multiply the sample by b_0 before filtering. With this intermediate variable, we can find the difference equation that corresponds to the cannonical form:

$$w(n) = x(n) + a_1 w(n-1) + a_2 w(n-2)$$
$$y(n) = w(n) + b_1 w(n-1) + b_2 w(n-2)$$

With this form, we have now only one buffer of samples to implement w(i)
Don't forget to beware of coefficient's signs from the Z transfert function (a_1, a_2)

1.2.3 Implementation in DSP : PID1.asm

The algorithm principle is the monitoring of 2 buffers, one in Data memory (delay_line=[w(n-2),w(n-1)]), one in program memory (coefs=[a2,a1,b2,b1]). Each time we read in a buffer, the associated pointer (« DAG register ») is automatically incremented by 1, so we point on the next data for a next read.
We must read delay line buffer in two pass : one for the computation of w(n), and one for the computation of y(n) so we point delay line buffer with two different DAG registers : (b0,i0) and (b1,i1).
Each time we can, we try to use the DSP with the maximum of capacity. For example, the next instruction :
`f12=f2*f4, f8=f8+f12, f3=dm(i0,m1), f4=pm(i8,m8);`
is executed in 1 cycle : (one multiply, one addition, one memory data read, one program memory read). We can so write a very compact and optimized code.
Don't be afraid with the code length : there is 4 pages of comments for only 18 lines interpreted by assembler (noted in bold). It's only to detail this first code we present

```
/*************************************************************************/
/*                     PID1.ASM  Michel JEVAUD 08/2000                   */
/*                                                                       */
/* origin:   "cascade.asm"                                               */
/* Analog Devices, Inc.  DSP Applications  P.O.Box 9106  Norwood, MA 02062  */
/* Christoph D. Cavigioli ... 25-Apr-1991                                */
/*                                                                       */
/* Difference Equations : w(n) = x(n) + a1*w(n-1) + a2*w(n-2)            */
/*                        y(n) = w(n) + b1*w(n-1) + b2*w(n-2)            */
/*                      (single biquad structure)                        */
/* Can be used for a more generally second order IIR:                    */
```

```
/*                    1 + b1z' + b2z"                                          */
/*              F(z)= ---------------   with z'=z^(-1)   z"=z^(-2)             */
/*                    1 - a1z' - a2z"                                          */
/* Be carreful: b0, a0 normalised to 1. if not,  multiply the sample x(n)     */
/* by b0/a0 before each call of PID1_Control                                  */
/*                                                                            */
/****************************************************************************/


/*  PARAMETERS TO SEND TO THE ROUTINE PID1_Control:                           */
/****************************************************************************/
/*  l0, l1, l8 = 0 (buffers delay_line and coefficients are'nt circular       */
/*  m1, m8 = 1                                                                */
/*  f8 = input sample x(n) just read by ADC                                   */
/*  b0 = address of delay line buffer (in Data Memory)                        */
/*      buffer delay_line: ----------                                         */
/*                         | w(n-2) | <----- (b0,i0)                         */
/*                         | w(n-1) |                                        */
/*                         ----------                                        */
/* after the first instruction of PID1_Control,the delay_line buffer uses a   */
/* 2nd DAG register b1,i1:                                                    */
/*                         ----------                                        */
/*          (b1,i1)------->| w(n-2) | <----- (b0,i0)                         */
/*                         | w(n-1) |                                        */
/*                         ----------                                        */
/*                                                                            */
/*  b8 = address of coefficient buffer(in Program memory)                     */
/* buffer coefficients:    ----------                                        */
/*                         |   a2   | <----- (b8,i8)                         */
/*                         |   a1   |                                        */
/*                         |   b2   |                                        */
/*                         |   b1   |                                        */
/*                         ----------                                        */
/****************************************************************************/
/*  PARAMETERS TO SEND TO THE ROUTINE PID1_Init                               */
/*  wich perfoms the RAZ delay line buffer w(n-1)=w(n-2)=0                    */
/****************************************************************************/
/*  l0 = 0                                                                    */
/*  b0 = address of delay line buffer                                         */
/****************************************************************************/
/*             RETURNED VALUES OF PID1_Control                                */
/*  f8 = last y(n) is the result of the compuutation for sample x(n)          */
/*       and we must deposit this value immediatly on the DAC                 */
/****************************************************************************/
/*            BUFFERS STATES AT THE END OF THE ROUTINE PID1_Control:          */
/****************************************************************************/
/*      buffer delay_line: ----------                                         */
/*                         | w(n-1) |                                        */
/*                         | w(n)   |                                        */
/*                         ----------                                        */
/*             (b1,i1)---->          <----- (b0,i0)                          */
/*  the delay_line buffer is ready to used for the following sample,(w(n-1)   */
/*  will become w(n-2) and w(n) will become w(n-1)). However, we must         */
/*  initialize DAG register (b0,i0): ad the end of the PID1_Control routine,  */
/*  DAGs register (b0,i0) and (b1,i1) point out of the buffer delay_line.     */
/*  We have not to initialize (b1,i1) before coming in routine because        */
/*  (b1,i1) adjust its pointing on (b0,i0) in th first instruction of         */
/*  PID1_Control                                                              */
/*                                                                            */
/*  buffer coefficients:   ----------                                         */
/*                         |   a2   |                                        */
```

```
/*                          |   a1   |                                      */
/*                          |   b2   |                                      */
/*                          |   b1   |                                      */
/*                          ----------                                      */
/*                                    <----- (b8,i8)                        */
/* same remark: at the end of PID1_Control routine DAG register (b8,i8)     */
/* points out of the coefficients buffer, we must initialize at the first   */
/* coefficients buffer address before the next call of the routine          */
/***************************************************************************/
/*            RETURNED VALUES OF PID1_Init                                 */
/*   delay line zeroed: w(n-1)=w(n-2)=0.0                                  */
/***************************************************************************/
/*      buffer delay_line: ----------                                      */
/*                          |   0    |                                      */
/*                          |   0    |                                      */
/*                          ----------                                      */
/*                                    <----- (b0,i0)                        */
/***************************************************************************/
```

```
/***************************************************************************/
/*                          Register File Usage:                           */
/*  f2, f3, f4, f8, f12         PID1_Control                                */
/*  f2                          PID1_Init                                   */
/*                         Data Address Generator Usage                    */
/*  i0, b1, i1, i8, m1, m8      PID1_Control                                */
/*  i0                          PID1_Init                                   */
/***************************************************************************/
/*                          Benchmark                                      */
/*    6 cycles     PID1_Control                                            */
/*    5 cycles     PID1_Init                                               */
/*    (incl. non-delayed rts)                                              */
/***************************************************************************/
/*                          Memory Usage                                   */
/*-------------------- PID1_Control ---------------------------------------*/
/*  10 words        instructions in PM                                     */
/*  4  words        coefficients in PM                                     */
/*  2  words        delay line storage in DM                               */
/*------------------------- PID1_Init -------------------------------------*/
/*  5 words         instructions in PM                                     */
/*  2 words         delay line storage in DM                               */
/***************************************************************************/

.GLOBAL PID1_Control, PID1_Init;
.EXTERN coefs, dline;

.SEGMENT /PM    pm_sram;
/***************************ROUTINE PID1_Init*******************************/
/* --- call this once to set up initial conditions -------------------------*/
PID1_Init           :       f2=0; /* clear f2                              */
                            dm(i0,1)=f2; /* clear w(n-2) storage          */
                            dm(i0,1)=f2; /* clear w(n-1) storage          */
                            rts;
/***************************************************************************/


/***************************ROUTINE PID1_Control****************************/
/* --- call this for every sample to be filtered --------------------------*/
```

```
/* --- comments for subroutine PID1_Control---------------------------------*/
/* we can see what happens to the buffers for each step                     */
/* #0 adjust DAG reg (b1,i1) to (b0,i0) ,i.e first address of delay line    */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/* i1--->| w(n-2) | <--- i0               |   a2   |<--i8                    */
/*       | w(n-1) |                       |   a1   |                         */
/*       ----------                      |   b2   |                         */
/*                                       |   b1   |                         */
/*                                       ----------                         */
/* #1 clear f12,                      rd w(n-2)-->f2, rd a2-->f4 prologue  */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/* i1--->| w(n-2) |                       |   a2   |                         */
/*       | w(n-1) | <---i0                |   a1   |<---i8                   */
/*       ----------                      |   b2   |                         */
/*                                       |   b1   |                         */
/*                                       ----------                         */
/* #2  f12=a2w(n-2), f8=x(n)+0,       rd w(n-1)-->f3, rd a1-->f4           */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/* i1--->| w(n-2) |                       |   a2   |                         */
/*       | w(n-1) |                       |   a1   |                         */
/*       ----------                      |   b2   |<---i8                   */
/*                 <---i0                |   b1   |                         */
/*                                       ----------                         */
/* #3  f12=a1w(n-1), f8=x(n)+a2w(n-2), wr w(n-2)<--f3=w(n-1), rd b2-->f4    */
/*     we haven't need to w(n-2) now ==> we replace it by w(n-1) in order   */
/*     to prepare the delay line buffer for the next sample                 */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/*       | w(n-1) |                      |   a2   |                         */
/* i1--->| w(n-1) |                      |   a1   |                         */
/*       ----------                      |   b2   |                         */
/*                 <---i0                |   b1   |<---i8                   */
/*                                       ----------                         */
/*************************************************************************/
/*************************************************************************/
/* #4   f12=b2w(n-2), f8=x(n)+a2w(n-2)+a1w(n-1)=w(n)  , rd b1-->f4          */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/*       | w(n-1) |                      |   a2   |                         */
/* i1--->| w(n-1) |                      |   a1   |                         */
/*       ----------                      |   b2   |                         */
/*                 <---i0                |   b1   |                         */
/*                                       ----------                         */
/*                                                 <---i8                   */
/* #5    f12=b1w(n-1), f8=w(n)+b2w(n-2), wr w(-1)<--f8=w(n),                */
/*     we haven't need to w(n-1) now ==> we replace it by w(n) in order     */
/*     to prepare the delay line buffer for the next sample                 */
/*       delay line                      coefficients                       */
/*       ----------                      ----------                         */
/*       | w(n-1) |                      |   a2   |                         */
/*       | w(n-1) |                      |   a1   |                         */
/*       ----------                      |   b2   |                         */
/* i1--->          <---i0                |   b1   |                         */
/*                                       ----------                         */
/*                                                 <---i8                   */
/* #6    f8=w(n)+b2w(n-2)+b1w(n-1)                                epilogue  */
/*************************************************************************/
```

```
PID1_Control:
    b1=b0;                                                  /* #0 */
        f12=f12-f12,          f2=dm(i0,m1), f4=pm(i8,m8);  /* #1 */
        f12=f2*f4, f8=f8+f12, f3=dm(i0,m1), f4=pm(i8,m8);  /* #2 */
        f12=f3*f4, f8=f8+f12, dm(i1,m1)=f3, f4=pm(i8,m8);  /* #3 */
        f12=f2*f4, f8=f8+f12,               f4=pm(i8,m8);  /* #4 */
        f12=f3*f4, f8=f8+f12, dm(i1,m1)=f8              ;  /* #5 */
    rts (db),      f8=f8+f12;                               /* #6 */
    nop;
    nop;
.ENDSEG;
```

1.2.4 PIDirq.asm : main program for calling PID1.asm

We have tested the algorithm on DSP 21020 evaluation board with the ADCs and DACs that are installed (AD7769). Theses ADCs and DACs are very poor (8bits, 3us) in comparison to the AD9220 that we'll have in the final design. However, we have obtained good results and we have seen that the algorithm principle was O.K., with the limitation developped in next sections.

So, the evaluation board has 2 ADCs , adc_a ,adc_b and 2 DACs dac_a,dac_b in data memory mapped. At the sequence of init, we initialize the timer to deliver an interupt each 10us. So, most of the time, the DSP is waiting for this interupt (wait :). When the interupt is active, the DSP jump at « new_sample » where it :

1) read adc_a
2) write a copie of this value on dac_a
3) Convert the 8 bit value integer of the adc_a in a float value
4) call PID1_Control to filtering this value
5) convert the float result in a 8 bit integer value
6) write the filtering result converted on dac_b
7) start a conversion of the 2 ADCs for the next sample
8) return of interupt

Notice that we have a delay between start of conversion of dac_a and write of the filtering result on dac_b. this delay is about 10us (there is not a lot of instruction between). This delay is a nuisance only for the phase, we'll seen this in next sections . One more time, there is a lot of comments and only the bold caracters are understood by the assembler.

```
/*************************************************************************/
/*                  PID1irq.asm Michel JEVAUD 08/2000                    */
/* origin:                                                               */
/* 1) iirirq.asm                                                         */
/* Analog Devices, Inc.  DSP Applications  P.O.Box 9106  Norwood, MA 02062 */
/* Christoph D. Cavigioli ... 25-Apr-1991                                */
/* 2) ordre1.asm M.Jevaud june 2000                                      */
/*                                                                       */
/* main program for calling PID1.asm                                     */
/* input, output from DM ports using Timer to create sampling interval   */
/* follows example system described in ADSP-21020 Users Manual           */
/*                                                                       */
/*************** required files:*****************************************/
/* "def21020.h":   bit placements in ADSP-21020 system registers         */
/* "iircoefs.dat": PID coefficients in this order:a2,a1,b2,b1            */
/*                 for w(n) = x(n) + a1*w(n-1) + a2*w(n-2)               */
/*                     y(n) = w(n) + b1*w(n-1) + b2*w(n-2)               */
```

```
/*                iircoefs.dat is not used in this program. However, we   */
/*                notice that we can load the coefs buffer with the       */
/*                instruction: VAR    coefs[4]="iircoefs.dat";            */
/* "Ezlab21k.ach": architecture file for download to Evaluation board      */
/* "PID1.asm": PID1_Control and PID1_Init routines                         */
/*************************************************************************/
/* In order to compile,link and download to Evaluation board:              */
/* step 1: asm21k PID1irq.asm                                              */
/*         produce PID1irq.obj from PID1irq.asm,def21020.h,iircoefs.dat     */
/* step 2: asm21k PID1.asm                                                 */
/*         produce PID1.obj from PID1.asm                                  */
/* step 3: ld21k PID1irq PID1 -a Ezlab21k -m                               */
/*         produce PID1irq.exe from PIDIirq.obj,PID1.obj with the          */
/*         architecture of Ezlab21k.ach , and produce a map file           */
/* step 4: spl21k -a Ezlab21k -pm -dm -ram -f B PID1irq                    */
/*         with prom splitter, produce PID1irq.stk (that can be downloaded */
/*         on evaluation board) from PID1irq.exe with the architecture of  */
/*         Ezlab21k.ach                                                    */
/* step 5: put PID1irq.stk into the c:\21kezlab directory                  */
/* step 6: launch lab21k.exe from c:\21kezlab directory and downlaod       */
/*         PID1irq.stk into the evaluation board with menu 0               */
/*************************************************************************/

#include "def21020.h"

.EXTERN PID1_Control,PID1_Init;
.GLOBAL coefs, dline;
.PRECISION = 40;
.ROUND_NEAREST;

/*************************************************************************/
/*             PORTS NOTIFICATION                                        */
/* like the architecture file  Ezlab21k.ach                              */
/* we only notify AD7769 ports, CODEC is out....                         */
/*************************************************************************/
.SEGMENT/DM   adc_a; /* adc-a address=0x40000020  */
.PORT         adc_a;
.ENDSEG;

.SEGMENT/DM   adc_b; /* adc-b address=0x40000028  */
.PORT         adc_b;
.ENDSEG;

.SEGMENT/DM   dac_a; /* dac_a address=0x40000030  */
.PORT         dac_a;
.ENDSEG;

.SEGMENT/DM   dac_b; /* dac-b address=0x40000038  */
.PORT         dac_b;
.ENDSEG;
/*************************************************************************/
```

```
/************************************************************************/
/*             VARIABLES NOTIFICATION                                   */
/************************************************************************/
.SEGMENT /DM    dm_sram;
.VAR    dline[2];                       /* filter delay line thus:      */
.ENDSEG;                                /* [w(n-2),w(n-1)]              */

/************************************************************************/

/************************************************************************/
/*                     INTERUPT TABLE                                   */
/* only processor reset service rst_svc and interupt timer high priority.  */
/* reset service call init_21k, bank init,registers,interupts... then we   */
/* at begin , i.e nothing to do except wait for timer interupt          */
/* the interupt timer=0 high priority jump to new_sample                */
/* new sample must be terminated with a RTI                             */
/************************************************************************/
.SEGMENT /PM    rst_svc;                /* processor RESET service      */
        call init_21k;
        jump begin;
.ENDSEG;

.SEGMENT /PM    tmzh_svc;               /* Interval Timer interrupt service*/
        jump new_sample;
.ENDSEG;

.SEGMENT /PM    pm_sram;                /* selected by PMS0~            */
/* "iircoefs.dat"     [a2,a1,b2,b1]                                     */
/* PID1: -0.967245,1.967245,0.991629,-1.967245                          */
/* PID1 with filtering integral: (freq1=1Hz)                            */
/*                  -0.96715536,1.9671533,0.99235814,-1.99235719        */
/* PID1 with filtering integral: (freq1=10Hz)                           */
/* -0.954774587875452,1.95452911713819,0.978746364190107,-1.97858271703193 */
/* delay advance phase correction 1 (2/08/2000):                        */
/* -0.90577878575296,1.90535593758009,0.95288590630004,-1.95246305812718   */
/* delay advance phase correction 2 (3/08/2000):                        */
/* -0.96793143697924,1.96789502274384,0.98763609512009,-1.98759968088469   */
.VAR    coefs[4]=-0.96793143697924,1.96789502274384,
                 0.98763609512009,-1.98759968088469;

begin:  r2=0;
        r3=0;
        dm(adc_a)=r3;           /* write to adc_a to start conversion       */
        dm(adc_b)=r3;           /* write to adc_b to start conversion       */
        bit set mode2 TIMEN;    /* turn on the interval timer               */
        bit set mode1 IRPTEN;/* allow interrupts                            */

/*--- main processing loop ----------------------------------------------*/
wait:   idle;                   /* wait for interrupts indefinitely         */
        jump wait;              /* after rti, go wait for more              */
```

```
/*****************************************************************************/
/*                  INIT DSP ROUTINE                                         */
/*  only called when DSP reset                                               */
/*  1) wait states initialization: 0 for all the PM, 0 for DM bank0,2 for DM bank1*/
/*     2 for DM bank2, 0 for DM bank3. Note that ADCs/DACs 7769 is on bank1   */
/*     wait state are all internal wait state only because we don't use DMACK */
/*     PMACK extaern signals ,DRAM MEMORY PAGE SIZE=32k (not used in this config  */
/*     because we have satic RAM                                             */
/*  2) load tperiod and tcount to initialize timer (one interupt each 10us)  */
/*  3) we only unmask timer high priority interupt                           */
/*  4) init FLAG1,FLAG2,FLAG3 like output (leds)                             */
/*  5) FLAG1=FLAG2=FLAG3=1 (LEDS=on)                                         */
/*****************************************************************************/
init_21k:
        pmwait  =0x1c21;     /* pgsz=32K,pmwtstates=0,sw.wtstates only        */
        dmwait  =0x70a521;   /* pgsz=32K,bank1&2_dmwtstates=1,sw.wtstates only    */
        tperiod =0x0149;     /* 10us, 4c4640=timer interrupt at 5 Hz          */
        tcount  = tperiod;
        irptl   =0;         /* clear any pending interrupts                  */
       bit set imask TMZHI;  /* only use interupt timer high priority         */
       bit set mode2 FLG1O|FLG2O|FLG3O; /* flag 1,2,3, used like outputs      */
        l0=0; l1=0; l8=0;    /* DAG registers initialisation                  */
        m1=1; m8=1;
        call PID1_Init(db);  /* zero the delay line                           */
        b0=dline;
        nop;
        bit set astat FLG1|FLG2|FLG3; /* turn on all LEDs                     */
       rts;
       nop;
       nop;

/*-------------- interrupt service routine which does the filtering ---------------*/
/*****************************************************************************/
/*                   TIMER INTERRUPT                                         */
/* toggle on FLAG2 led, copy from adc_b on dac_a and filtering adc_b to deliver the*/
/* result on dac_b                                                           */
/*****************************************************************************/
new_sample:
        bit tgl astat FLG2;       /* toggle the flag 1 LED
*/
        r0=dm(adc_b);   /* read from adc_b                                   */
        dm(dac_a)=r0;  /* copy to dac_a to see                              */
        r0=lshift r0 by -24;/* shift 24 because dac is on bits(24..31)       */
        f1=float r0; /* conversion float                                    */
        f2=127.0; /* centering value: ADCs and DACs aren't centered , it isn't     */
        f1=f1-f2; /* necassary if we use AC filtering but better for computation   */
        f2=1.0;   /*f2=b0                                                   */
        f8=f1*f2; /* multiply by b0                                         */
        call PID1_Control(db); /* input=f8, output=f8                       */
```

```
        b0=dline;
        b8=coefs;
       f2=127.0; /* re scaling uncentered value for DACs                    */
       f8=f8+f2;
       r0=fix f8;    /* we put result in fixed point register r0           */
       r0=lshift r0 by 24;  /*shift 24 because dac is on bits(24..31)       */
       dm(dac_b)=r0;  /*write read value to dac_b                          */
       dm(adc_a)=r3;  /* write to adc_a to start conversion                 */
       dm(adc_b)=r3;  /* write to adc_b to start conversion                 */
       rti;
     nop;
     nop;

.ENDSEG;
```

This main program is in respect with the architecture described in the architecture file Ezklab.ach wich is linked with PID1irq1.asm and PID1.asm to produce the executable file.
Here is the architecture file Ezlab.ach:

```
.SYSTEM        EZ_LAB;
.PROCESSOR =    ADSP21020;

.SEGMENT /RAM  /BEGIN=0x000000 /END=0x000007 /PM     resrvd0;
.SEGMENT /RAM  /BEGIN=0x000008 /END=0x00000F /PM     rst_svc;
.SEGMENT /RAM  /BEGIN=0x000010 /END=0x000017 /PM     resrvd1;
.SEGMENT /RAM  /BEGIN=0x000018 /END=0x00001F /PM     sovf_svc;
.SEGMENT /RAM  /BEGIN=0x000020 /END=0x000027 /PM     tmzh_svc;
.SEGMENT /RAM  /BEGIN=0x000028 /END=0x00002F /PM     irq3_svc;
.SEGMENT /RAM  /BEGIN=0x000030 /END=0x000037 /PM     irq2_svc;
.SEGMENT /RAM  /BEGIN=0x000038 /END=0x00003F /PM     irq1_svc;
.SEGMENT /RAM  /BEGIN=0x000040 /END=0x000047 /PM     irq0_svc;
.SEGMENT /RAM  /BEGIN=0x000048 /END=0x00004F /PM     resrvd2;
.SEGMENT /RAM  /BEGIN=0x000050 /END=0x000057 /PM     resrvd3;
.SEGMENT /RAM  /BEGIN=0x000058 /END=0x00005F /PM     cb7_svc;
.SEGMENT /RAM  /BEGIN=0x000060 /END=0x000067 /PM     cb15_svc;
.SEGMENT /RAM  /BEGIN=0x000068 /END=0x00006F /PM     resrvd4;
.SEGMENT /RAM  /BEGIN=0x000070 /END=0x000077 /PM     tmzl_svc;
.SEGMENT /RAM  /BEGIN=0x000078 /END=0x00007F /PM     fix_svc;
.SEGMENT /RAM  /BEGIN=0x000080 /END=0x000087 /PM     flto_svc;
.SEGMENT /RAM  /BEGIN=0x000088 /END=0x00008F /PM     fltu_svc;
.SEGMENT /RAM  /BEGIN=0x000090 /END=0x000097 /PM     flti_svc;
.SEGMENT /RAM  /BEGIN=0x000098 /END=0x00009F /PM     resrvd5;
.SEGMENT /RAM  /BEGIN=0x0000A0 /END=0x0000A7 /PM     resrvd6;
.SEGMENT /RAM  /BEGIN=0x0000A8 /END=0x0000AF /PM     resrvd7;
.SEGMENT /RAM  /BEGIN=0x0000B0 /END=0x0000B7 /PM     resrvd8;
.SEGMENT /RAM  /BEGIN=0x0000B8 /END=0x0000BF /PM     resrvd9;
.SEGMENT /RAM  /BEGIN=0x0000C0 /END=0x0000C7 /PM     sft0_svc;
.SEGMENT /RAM  /BEGIN=0x0000C8 /END=0x0000CF /PM     sft1_svc;
.SEGMENT /RAM  /BEGIN=0x0000D0 /END=0x0000D7 /PM     sft2_svc;
.SEGMENT /RAM  /BEGIN=0x0000D8 /END=0x0000DF /PM     sft3_svc;
.SEGMENT /RAM  /BEGIN=0x0000E0 /END=0x0000E7 /PM     sft4_svc;
.SEGMENT /RAM  /BEGIN=0x0000E8 /END=0x0000EF /PM     sft5_svc;
.SEGMENT /RAM  /BEGIN=0x0000F0 /END=0x0000F7 /PM     sft6_svc;
.SEGMENT /RAM  /BEGIN=0x0000F8 /END=0x0000FF /PM     sft7_svc;

.SEGMENT /RAM  /BEGIN=0x000100 /END=0x007FFF  /PM     pm_sram;
.SEGMENT /RAM  /BEGIN=0x00000000 /END=0x00007FFF /DM    dm_sram;

.SEGMENT /PORT /BEGIN=0x40000000 /END=0x40000005 /DM    hip_regs;    /* do not used, only for  Codec   */
```

Interupt Vector Table :
In fact, we only use 2 vectors in this table :

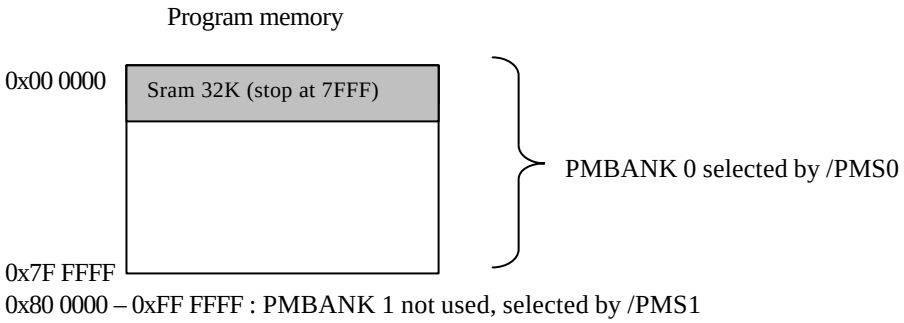
1) rst_svc vector wich is the interrupt vector for the interupt reset service routine (that will a jump to « begin : »)
2) tmzh.svc vector wich is the interupt vector for the timer=0 high priority service routine (that will a jump to « new sample : »

note : this interupt vector table is partitionned in 8 words areas. This is done to put in this 8 words of program the direct code of the interupt service routine in the case of this code is little, avoiding to put a jump to a label

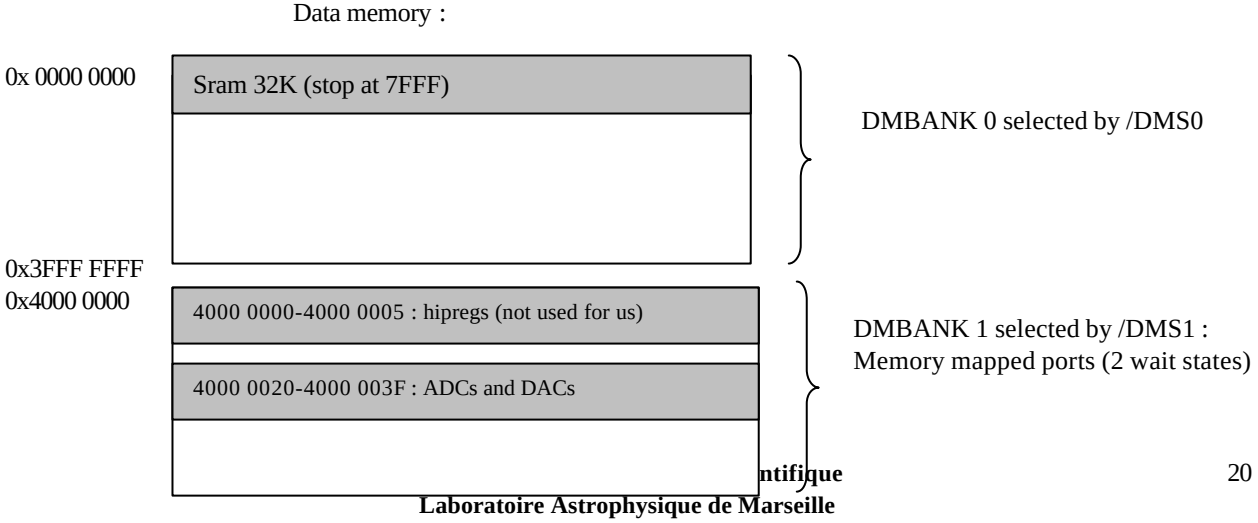
```
.SEGMENT /PORT /BEGIN=0x40000020 /END=0x40000020 /DM    adc_a;
.SEGMENT /PORT /BEGIN=0x40000028 /END=0x40000028 /DM    adc_b;
.SEGMENT /PORT /BEGIN=0x40000030 /END=0x40000030 /DM    dac_a;
.SEGMENT /PORT /BEGIN=0x40000038 /END=0x40000038 /DM    dac_b;

.ENDSYS;
```

1.2.5 Memory mapping

This architecture is in respect with the hard of the 21020 evaluation board following (the two SRAMs are 0 wait state and the ADCs and DACs are 2 wait sates) :



0x80 0000 – 0xFF FFFF : PMBANK 1 not used, selected by /PMS1



Laboratoire Astrophysique de Marseille
BP 8 - 13376. Marseille Cedex 12.
Tel. : 33 .4.91.05.59.00 - Fax : 33 .4.91.66.18.62

0x7FFF FFFF
0x8000 0000-0xBFFF FFFF : DMBANK 2 not used (selected by /DMS2)
0xC000 0000-0xFFFF FFFF : DMBANK 3 not used (selected by /DMS3)

After compiling and linking all the programs, we see on the map file the following mapping :

Program Memory :
0x00 0000-0x00 00FF : Interupt vector table :
0x00 0100-0x00 0103 : Buffer coefs (a2,a1,b2,b1)
0x00 0104-0x00 0109 : begin label
0x00 010A-0x00 010B: wait label
0x00 010C-0x00 011E : init_21k routine
0x00 011F-0x00 0134 : new_sample routine

0x00 0135-0x00 0138 : PID1_init routine (4 words)
0x00 0139-0x00 0141 : PID1_Control routine (9 words)

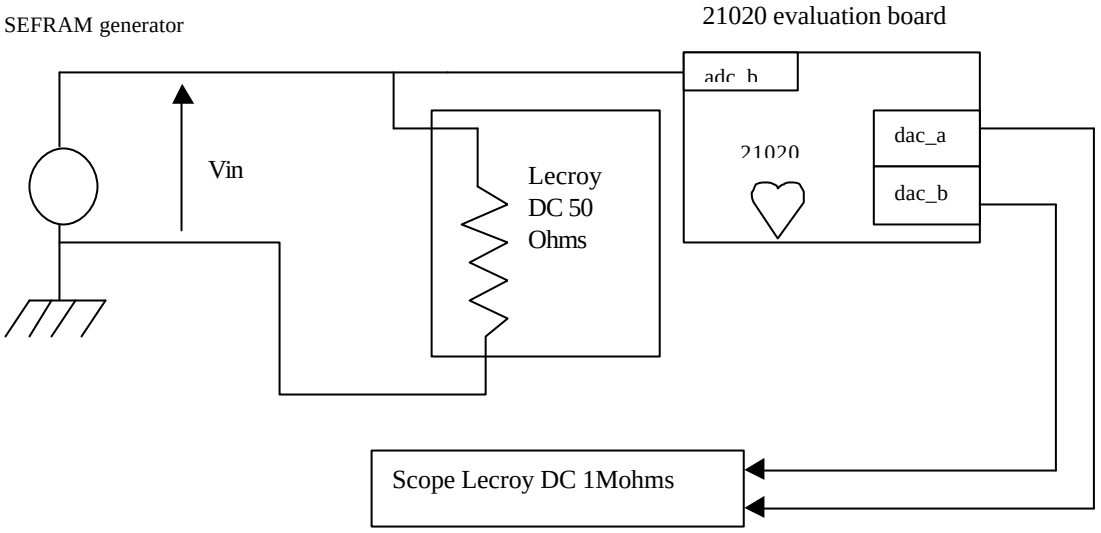
Data Memory :
0x0000 0000-0x0000 0001 : delay_line buffer [w(n-2),w(n-1)]

1.2.6 Measurement

1.2.6.1 HARD SETUP

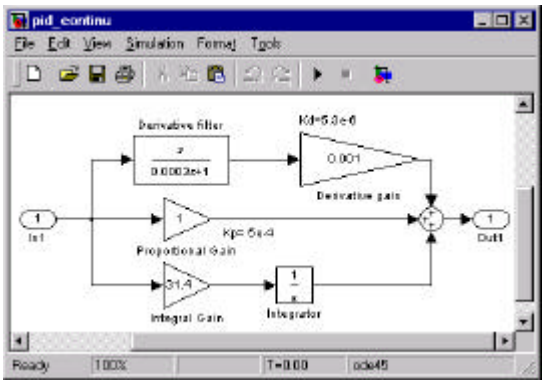
To make measurement on DSP running on evaluation board, we have perfomed Bode diagram with the help of :

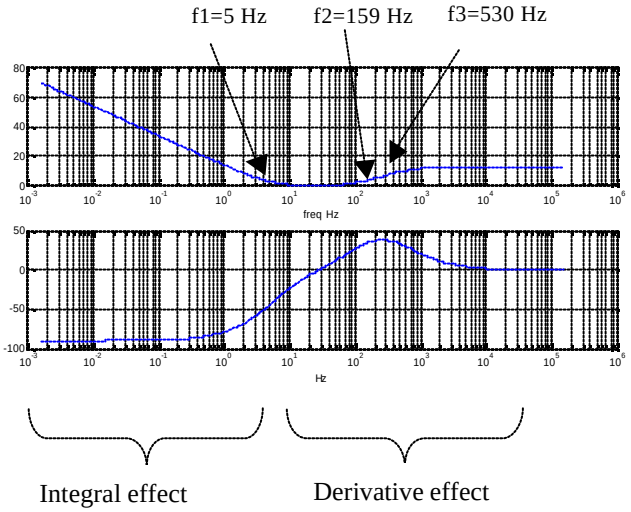
1) Lecroy Oscilloscope LC334 (500MHz,25MHz band limited, ADC 8 bits)
2) Sefram 4410 function generator 20Hz-10KHz



We have choosen Vin to cover the maximum span of adc_b, so we have Vin= 4V pp (the purpose of the experiment is not to measure the noise of the adcs 7769, but to verify the algorithm implementation)

1.2.6.2 FIRST TEST : PID CORRECTION WITH A PURE INTEGRATOR

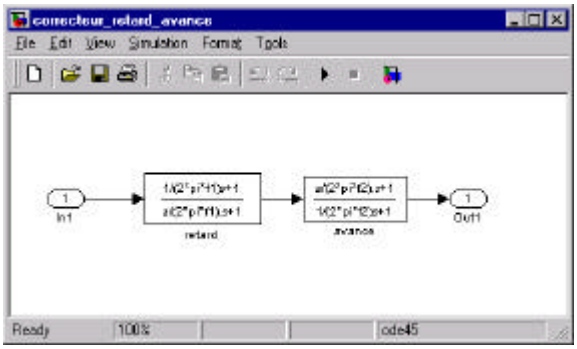


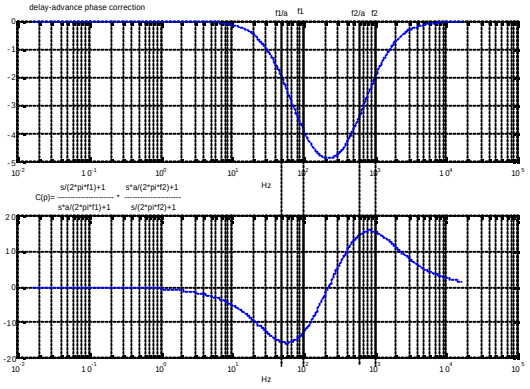


The problem with this correction is that there is a pure integrator and we are coupling in DC for the input. So, the smaller offset at the input is integrated by this correction. So, we see a important drift at the output, wich quickly induce the DACs saturation. We have not noted measurement for this correction because of this drift is very noisy to make measurement.

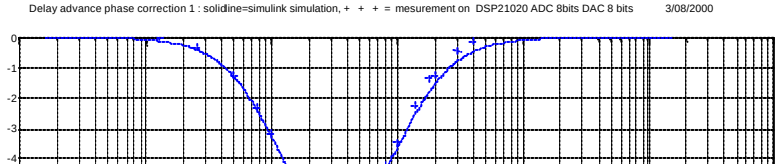
1.2.6.3 SECOND TEST : DELAY ADVANCE PHASE CORRECTION.

To avoid the drift preceding described, we have choosen to limited the integral effect and replace it by a filtering integral. Naturally (the derivative was so filtering) we obtain the correction named «delay advance phase correction »

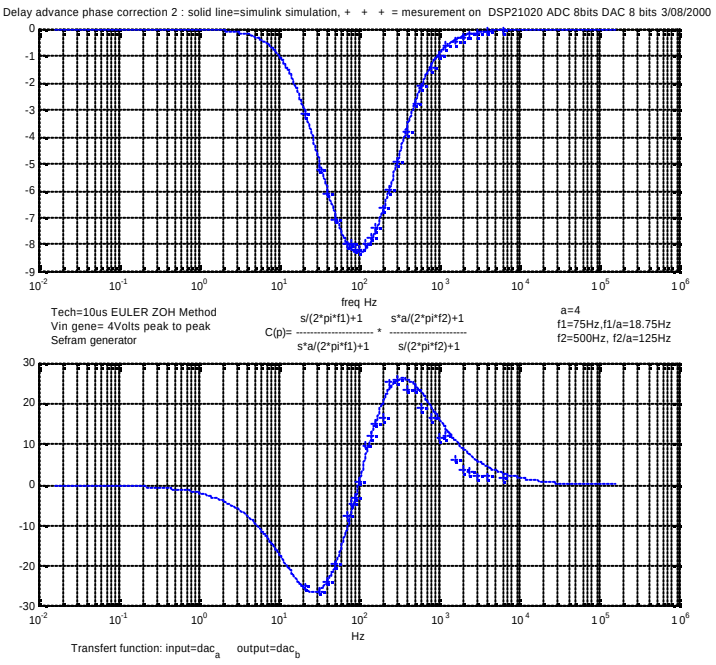




We have tried to adjust the frequences and the coefficient a to make easily measurement.
For f1=600Hz,f2=1500Hz ,a=8 we have found :
-0.90577878575296,1.90535593758009,0.95288590630004,-1.95246305812718
for the (a2,a1,b2,b1) coefficients to perform the difference equation. We use ZOH method given by MATLAB , but we have beware of the precision of coefficient : we must select the format long under Matlab.
With this parameter, we have obtained the following curve :



The magnitude follows the theoretic curve computed by Matlab, but we can see a fall of phase , that begins at 400 hertz. The cause is the pure delay between the start of conversion of ADCs and the DACs write : we start a conversion one period of timer before write the filtered result : we can see this delay when we measure the delay between adc_b and dac_a (that is a copy of adc_b), we measure a pure delay of about 13.6 us. In addition to the pure delay of 10us, there is also a delay due to the setting time of the DAC (about 3us for the AD7769). We can reduce this when we consider that the input is the value delivered by dac_a (the copy of the input adc_b).



For this new measurement, we can see a very good following of the magnitude and a correct phase down to 1Khz. Up to 1 Khz, , we still observe a little error on phase, due to different setting time DACs and to the delay between write dac_a instruction and write dac_b instruction

1.3 General IIR algorithm

1.3.1 Discretize a continuous filter

All the continuous filters that we use can be (for example by the conversion « c2d » under Matlab) converted in time discret by the following Z transform :

$$\frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ...b_M z^{-M}}{1 - a_1 z^{-1} + a_2 z^{-2} + ...a_N z^{-N}}$$

The PID described in previous section is only a particular case of this expression. However, we'll keep the PID1_Control routine because it's a more easy and compact code that the one we'll write for a general IIR and it's better to use PID1_Control when we have to implement an 2 order IIR.
Generally, we have M=N. If it wasn't, we can put null coefficients.
In a first time, we don't implement this filter and we don't make measurement with it because the filters used in control (ex : notch filters) are all 2nd order filters. So the algorithm of 1.3 are given only for examples and for future developments.

1.3.2 Difference equations

In the same order than the PID, we introduce the intermediate variable w(z) to minimize the managment of the delay_line buffer. So, the 2 difference equations are :

$$w(n) = x(n) + \sum_{k=1}^{N} a_k w(n-k) = x(n) + a_1 w(n-1) + a_2 w(n-2) + ...a_N w(n-N)$$

$$y(n) = \sum_{k=0}^{k=N} b_k w(n-k) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) + ...b_N w(n-N)$$

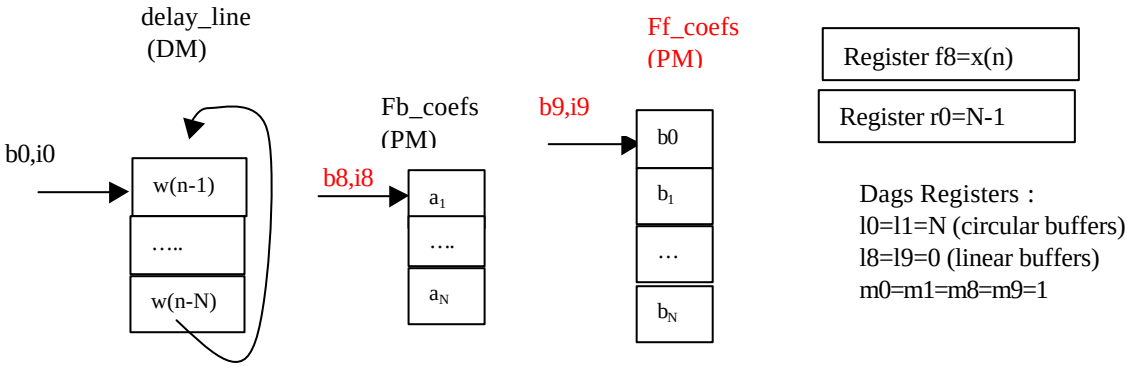
where x(n) is the input sample at the time t=n*(time sampling), y(n) the corresponding ouput sample and N is the IIR filter order

1.3.3 Implementation in DSP : IIR1.asm

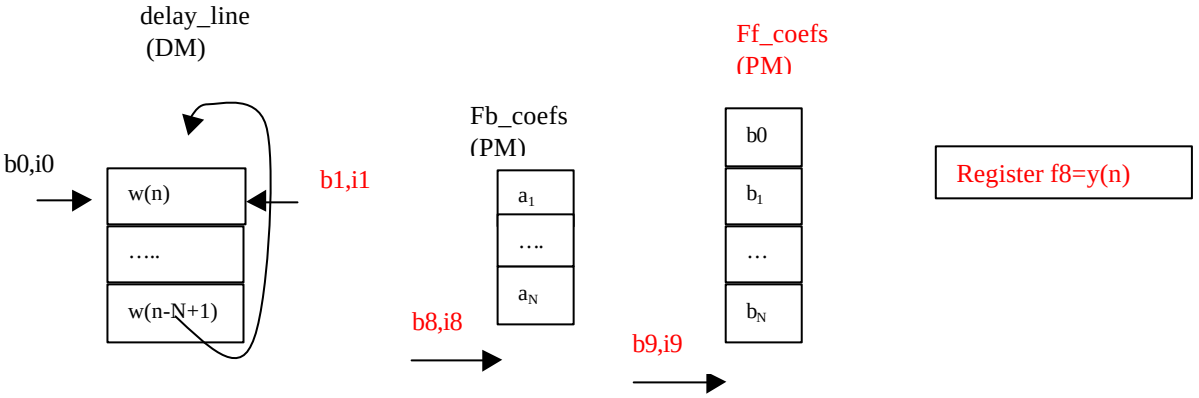
1.3.3.1 ROUTINE CORE

Before see all the code, we can detail the routine core that is the implementation of difference equations. There are 2 parts in this core : part « pole loop » computes w(n) and part « zero loop » computes y(n) and refreshes step by step the buffer delay line to prepare this for the next sample. We must read the buffer delay line in 2 passes, so it's easier to use a circular buffer for delay line. We have also use 2 buffers, not circular, for the coefficients :feedback coefficients (size=N, Fb_coefs={a1,a2,…aN}) and feedforward coefficients (size=N+1, Ff_coefs={b0,b1,b2,…bN})

At the input of the routine, the states of buffers and registers must be :



At the end of the routine, delay line buffer is ordonned for the next sample , the dags registers (b8,i8) and (b9,i9) point out of their buffers, the result y(n) is in f8.




```
IIR_Control :
        b1=b0 ;   /* we also point delay line buffer with (b1,i1) : i1 point on w(n-1) */
        f12=f12-f12,                    f2=dm(i0,m0),  f4=pm(i8,m8) ;   /* clear f12, f2=w(n-1),  f4=a1  */
        lnctr=r0, do poleloop until lce ;
poleloop : f12=f2*f4,     f8=f8+f12,    f2=dm(i0,m0),  f4=pm(i8,m8) ;
/* step 1       f12=a1w(n-1),  f8=x(n)+0,   f2=w(n-2),        f4=a2                                  */
/* step 2       f12=a2w(n-2),  f8=x(n)+a1w(n-1), f2=w(n-3),  f4=a3                                  */
/*  ……..                                                                                          */
/* step (N-1)  f12=aN-1w(n-N+1), f8=x(n)+a1w(n-1)+…+ aN-2w(n-N+2), f2=w(n-N), f4=aN          */
        f12=f2*f4,      f8=f8+f12 ;   /* f12=aNw(n-N), f8=x(n)+a1w(n-1)+…+ aN-1w(n-N+1)          */
                        f8=f8+f12 , f4=pm(i9,m9);   /* f8=x(n)+a1w(n-1)+…+ aN-1w(n-N+1)+ aNw(n-N), it's w(n), f4=b0   */
        f9=f8 ;   /* f9=f8=w(n)       */
        f8=f9*f4 ; /* f8= b0w(n) to be tested  if we cannot parallelize with another instruction */
        f12=f12-f12 ; /* clear f12 */
        r0=r0+1 ;
        lnctr=r0, do zeroloop until lce ;
                                f2=dm(i0,m0),  f4=pm(i9,m9) ;
/* step 1                      f2=w(n-1),       f4=b1          */
/* step 2                      f2=w(n-2),       f4=b2          */
/*……..                                                        */
```

```
/* step (N-1)                          f2=w(n-N+1),   f4= b_N-1        */
/* step (N)                            f2=w(n-N),     f4= b_N          */
        f9=f2,                          dm(i1,m1)=f9 ;  /* to prepare the buffer delay line for next sample      */
/* step 1       f9=w(n-1),             w(n)        → buffer delay_line at 1st address, push out w(n-1)        */
/* step 2       f9=w(n-2),             w(n-1)      → buffer delay_line at 2nd address, push out w(n-2)        */
/* …….                                                                                                        */
/* step (N-1)   f9=w(n-N+1),           w(n-N+2)    → buffer delay_line at (N-1) address, push out w(n-N+1)    */
/* step (N)     f9=w(n-N)  (don't care), w(n-N+1)  → buffer delay_line at (N) address, push out for ever w(n-N) */

 zeroloop: f12=f2*f4,   f8=f8+f12;
/* step 1       f12=b1w(n-1),  f8= b0w(n) +0                                                                  */
/* step 2       f12=b2w(n-2),  f8= b0w(n) + b1w(n-1)                                                          */
/*……..                                                                                                       */
/* step (N-1)   f12=bN-1w(n-N+1),  f8= b0w(n) + b1w(n-1)+….+ bN-2w(n-N+2)                                     */
/* step (N)     f12=bNw(n-N),      f8= b0w(n) + b1w(n-1)+….+ bN-2w(n-N+2)+ bN-1w(n-N+1)                       */
      rts(db) ;
      r0=r0-1 ;
      f8=f8+f12 ; /* f8= b0w(n) + b1w(n-1)+….+ bN-2w(n-N+2)+ bN-1w(n-N+1) +bNw(n-N)=y(n)  */
```

1.4 Notch filters

1.4.1 Discretize linear notch filter

We use notch filter for FTS control. The basis form is :

$$H(s) = \frac{s^2 + 2z_1 w_1 s + w_1^2}{s^2 + 2z_2 w_2 s + w_2^2} \quad \text{with} \quad w_1 \approx w_2$$

The discretization of this notch filter gives a 2nd order IIR, so we can use the PID algorithm that we have described in section 1.2.

The numerical application that we have measured was : $H(s) = \frac{s^2 + 3s + 30000}{s^2 + 3.1s + 31000}$

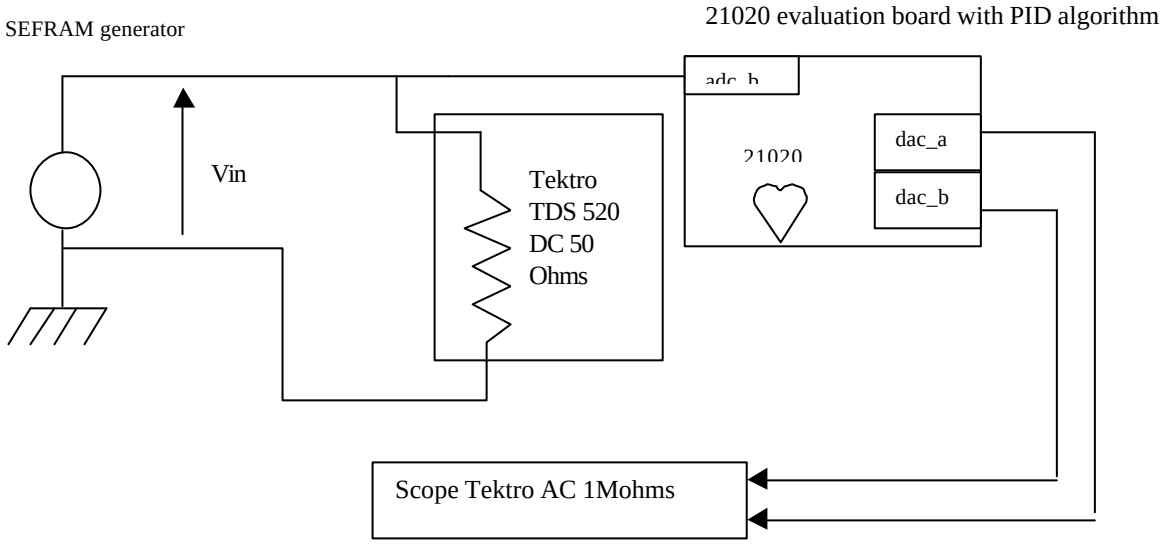
that has given with the ZOH method a 2nd order IIR $H(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$ with the following coefficients :

a1 = 1.99996590052935, a2 = -0.99996900048050, b1 = -1.99996695051280 ,b2 = 0.99996995046552

1.4.2 Measurement

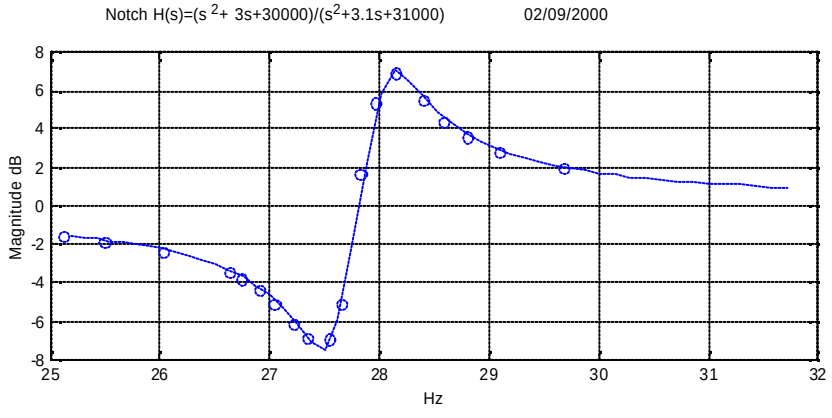
1.4.2.1 HARD SETUP

To make measurement on the implementation of the notch filter on the DSP evaluation board, we have used the setup of the PID measurement, except that we have used a Tektronix TDS 520 oscilloscope and we have used the measurement capabilities of this scope (frequency mesurement of dac_a, pk to pk measurements of dac_a and dac_b. We have compute the tranfert function between dac_a and dac_b (dac_a is the copy of adc_b and avoid the problem of delay explained in 1.2.6.3. Also this avoid the problem of different span for ADC and DAC of the evaluation board)



1.4.2.2 CURVE COMPARAISON

We have compared the theoric bode of simulink and the measured bode of DSP between, 25Hz and 32Hz, we haven't at theses frequencies no problem of pure delays explained in 1.2.6.3. So, the measured curve is really very beautiful and follows very well the theoric bode :



Centre National de la Recherche Scientifique
Laboratoire Astrophysique de Marseille
BP 8 - 13376. Marseille Cedex 12.
Tel. : 33 .4.91.05.59.00 - Fax : 33 .4.91.66.18.62