SPIRE BSM flight tuning – Initial Report
Doug Griffin

Table of Contents

Change List

0.1 Initial issue circulated for comment. 12 October 2009
Issue 1.0: 02 February 2010

1. Introduction

This TN has the following purposes:

- Document the software used to control the BSM
- Summarise from a phenomenological point of view the issues encountered in the tuning of the BSM control parameters during the flight commissioning of the mechanism
- Propose ways to improve the performance of the mechanism to achieve the required performance

The BSM can be operated in several nominal modes. The only ones considered in this note are the open loop modes used during the functional tests on OD6 and the closed loop operations using the magneto-resistive sensors.

2. Overview of the hardware

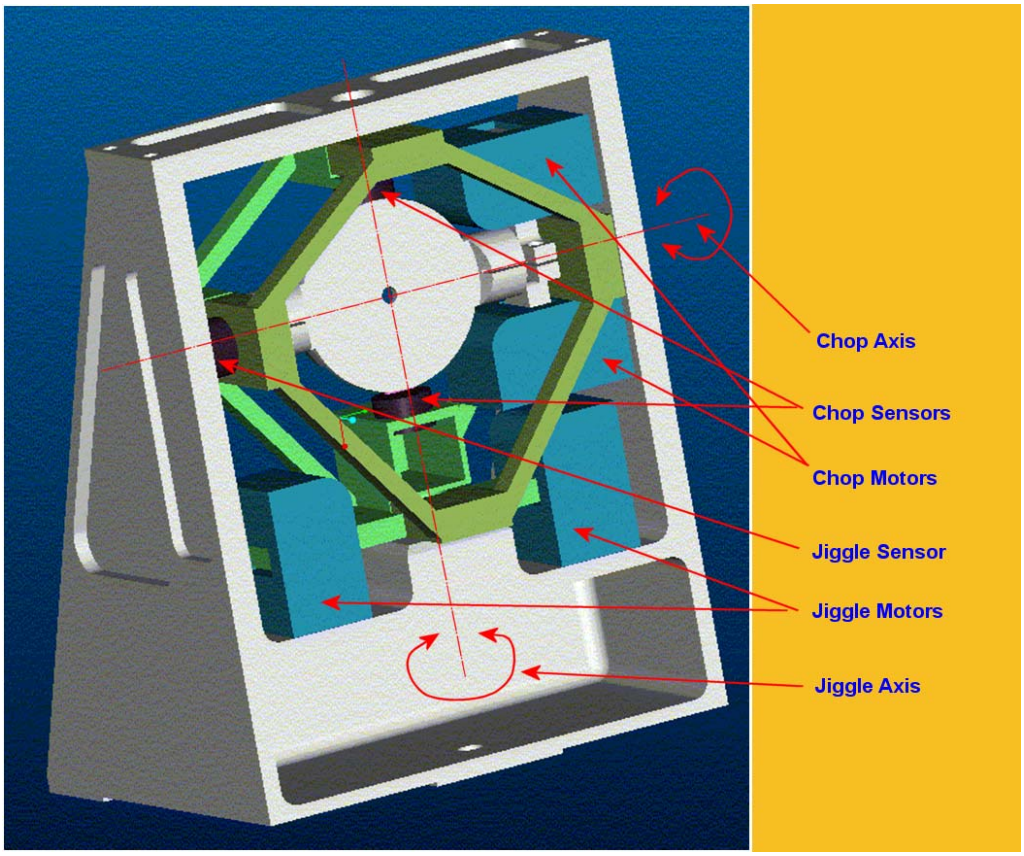

Figure 1: Simplified view of the BSM

An illustrative drawing of the BSM is shown in Figure 1. This image was taken from an early design description of the mechanism and is simplified with respect to the actual design, but it useful to highlight the pertinent features of the mechanism.

The mirror is approximately ∅33mm. To induce a chop rotation, current is supplied to the motor windings. The current supplied by drive circuit for the motors in the MCU is regulated by the output of a 16-bit digital to analogue converter. As the mirror undergoes a chop rotation, permanent magnets bonded into the side walls of the mirror change the magnetic flux around the magneto-resistive Chop Sensors. The resulting change in output of these sensors is used by the digital control algorithm programmed into a Digital Signal Processing device (DSP26020) in the MCU.

The chop mirror is mounted on the jiggle stage which is a large metal supporting frame which surrounds the mirror. The jiggle stage has motors and magneto-resistive sensors similar to the chop stage. As the mechanical inertia of the jiggle stage is much higher than the chop, the settling time is much longer.

The physical size of the maximum mechanism chop is approx. ± 2.5° and ±0.5°.

3. Operational use of the BSM

The BSM is used in four main operational scenarios; spectrometer mapping, peak-up, 64-point jiggle and 7-popint jiggle.

3.1 Spectrometer mapping

In this mode, the BSM is used to hold the BSM in a sequence of 16 offset positions (or 4 if only the SLW is being used) in order to full sample a spectro-photometric map. The dynamics of this are very slow as the BSM has to be held in a constant position throughout the FTS scan.

3.2 Peak up

TBW

3.3 64-point jiggle mapping

It was initially thought in late summer 2009 that this mode would be effectively removed from operational use on SPIRE. At the consortium meeting in early October 2009, there were some voices that wanted to retain the mode in operational use.

TBW

3.4 7-point jiggle map

During a 7-point jiggle map, the BSM is used to chop on and off a compact source when either the precise location of the source is not know with sufficient accuracy, or the blind pointing of the spacecraft does not allow a single pointing of the spacecraft. The Chop and Jiggle coordinates, in the units of the BSM of a typical 7-point jiggle map observation carried out on OD117. The observation starts on the centre pointing, cycles through the other six around the centre and then finishes with a repeat of the central pointing. At each of the locations, the mechanism chops between the left and right beams sixteen times at a nominal frequency of 2 Hz which results in a dwell time at each chop position of nominally 500 ms. The sequence of the chops and the jiggles is shown in Figure 2.

Table 1: Chop and Jiggle parameters used in 0x5000183F

Beam	*Position working name*	*Chop Target*	*Jiggle Target*
		Sensor ADC units	
Left	Centre	27280	39426

Beam	Position working name	Chop Target	Jiggle Target
		Sensor ADC units	
	North	27280	41269
	North-East	28111	40347
	North-West	26448	40374
	South	27280	37582
	South-East	28111	38504
	South-West	26448	38504
Right	Centre	46113	39426
	North	46113	41269
	North-East	46779	40374
	North-West	45434	40347
	South	46113	37582
	South-East	46779	38504
	South-West	45434	38504

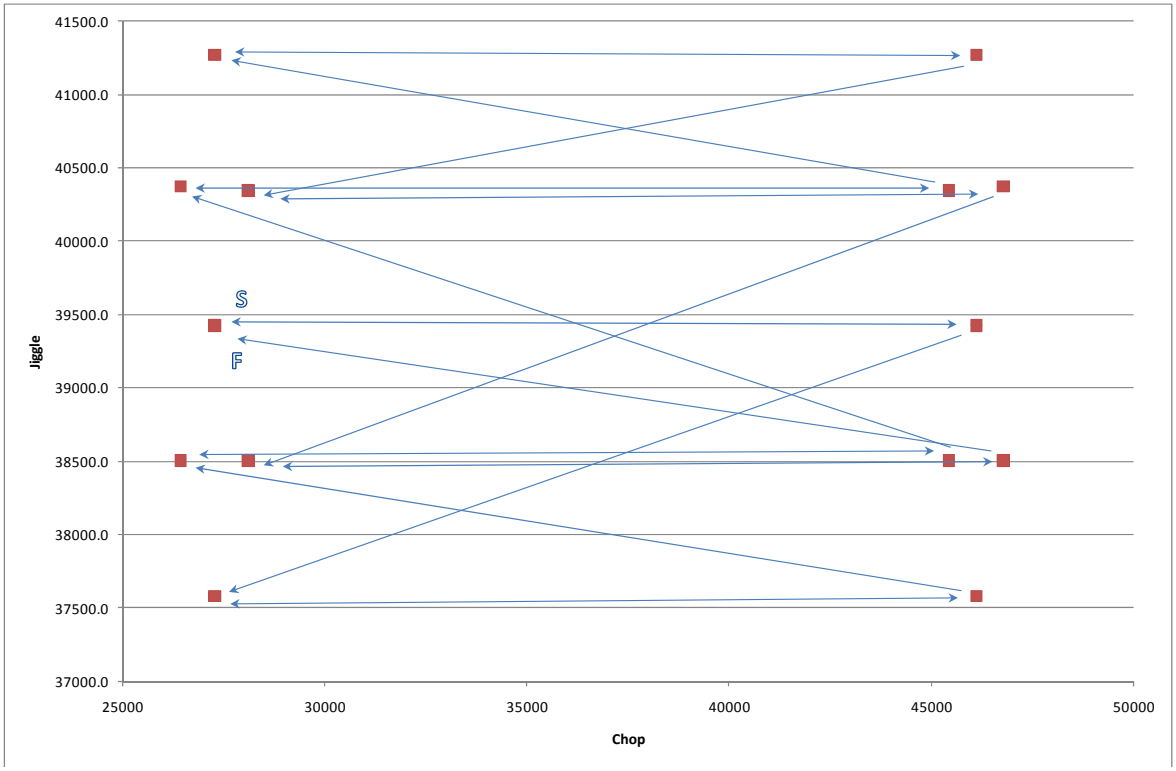

Figure 2: Sequence of 7-point Jiggle Map used in 0x5000183F

4. Observed Behaviour

4.1 Open Loop (0x0000200C OD 6)

During OD 6, functional tests were carried out on the BSM to check out the health of the mechanism. The DAC controlling the jiggle motor was commanded in open loop to three settings ranging from a relatively large "negative" current, to nominally zero current and a large "positive" current (see Figure 4). At each jiggle motor current, the chop motor DAC was stepped through approximately 16 levels which envelope the normal working range of the chop axis (see Figure 3). The outputs from CHOPSENSIG and JIGSENSIG versus time are shown in Figure 5 and Figure 6.

Figure 7 plots JIGSENSIG vs. CHOPSENSIG for these tests and indicates the level of cross-axis coupling. The zero current position for the chop and jiggle axes is indicated by the "bore sight chop" and "bore sight jiggle" traces. The nominal zero of the chop and jiggle sensors is at the ADC value of 2^{15} / 32768 (plotted as "Chop and Jiggle Sensor Zero"). It can be seen that there is a very significant offset between the nominal sensor zero and the nominal bore sights.

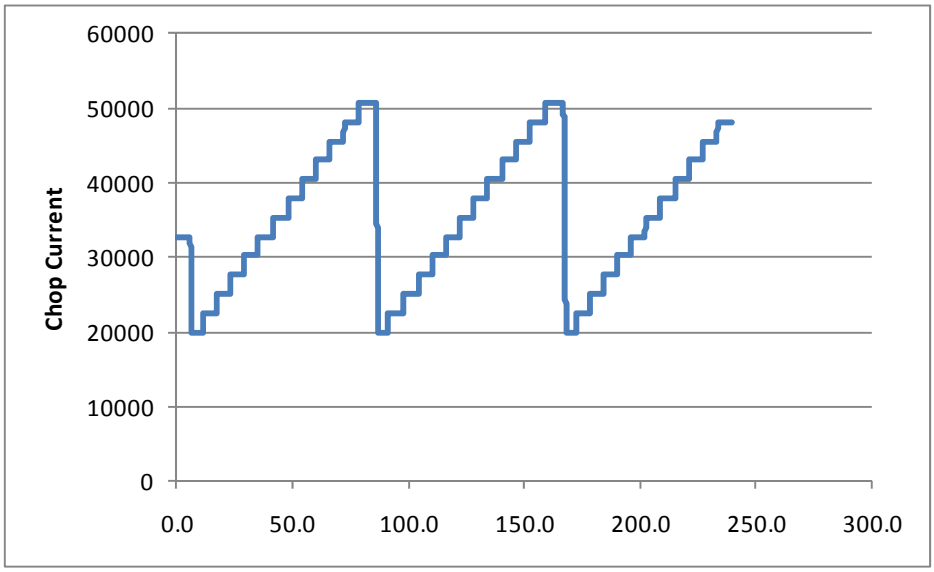

Figure 3: Profile of chop motor current in open loop test

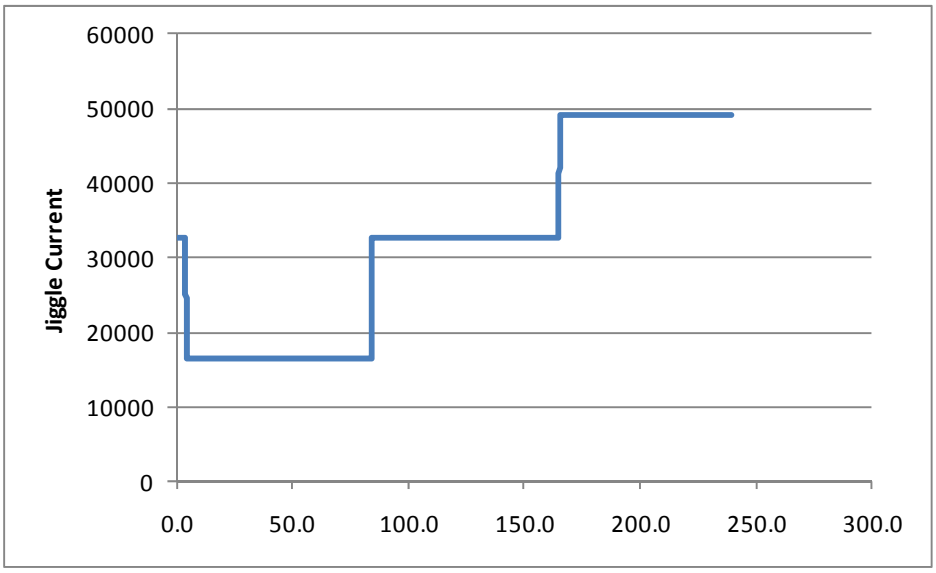

Figure 4: Profile of jiggle motor current in open loop test

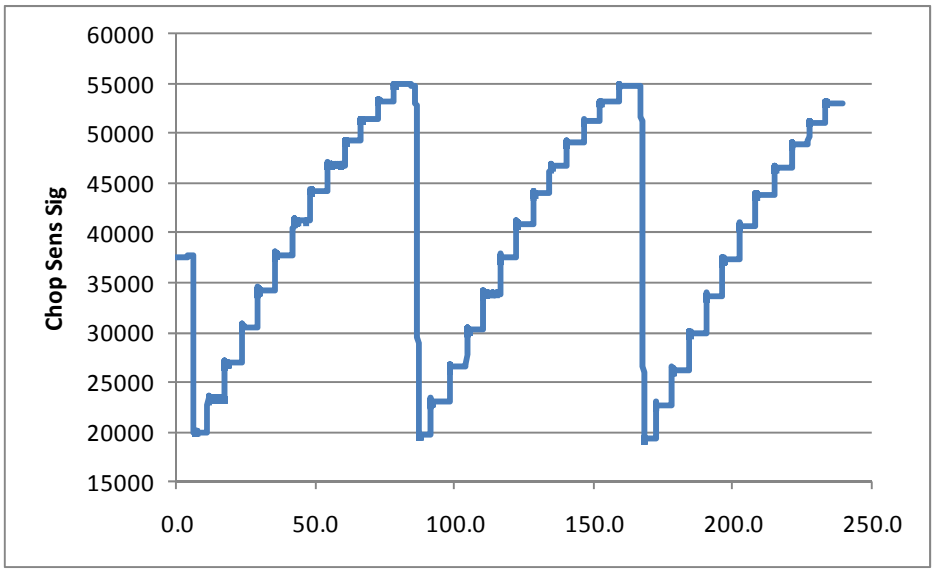

Figure 5: Chop response in open loop test

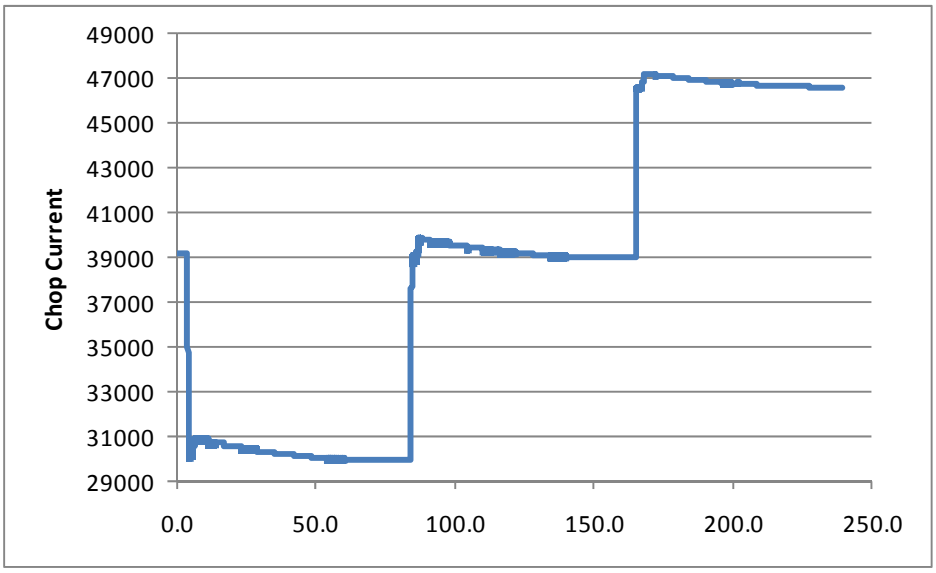

Figure 6: Jiggle response in open loop test (label is wrong)

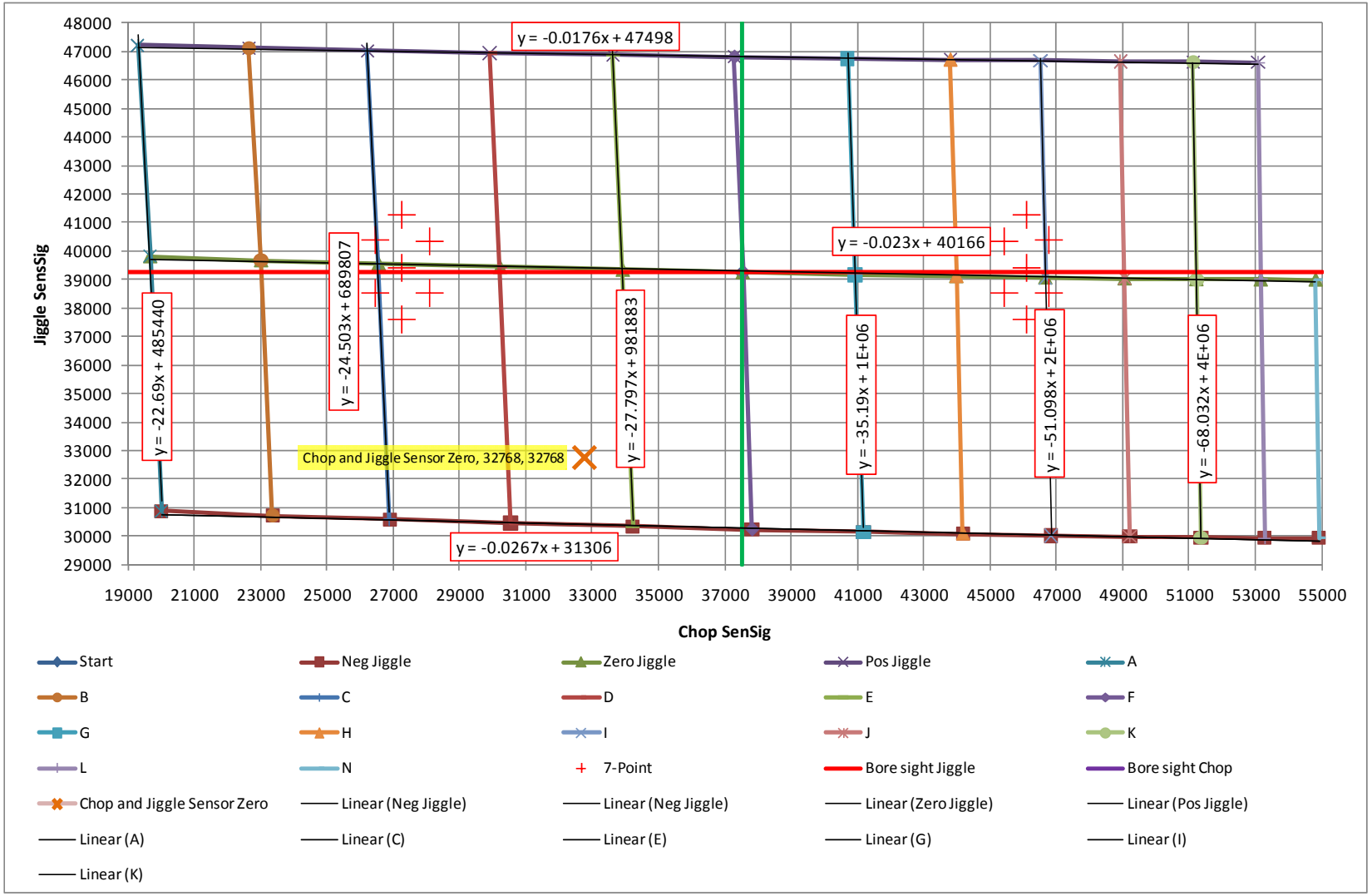

Figure 7: Chop and Jiggle sensor outputs in open loop mode

4.2 7-point Jiggle Map (0x5000183F – OD117)

4.2.1 Jiggle Anomalies

There are several issues observed in the 7-point jiggle maps. Figure 8 shows the typical response of the Jiggle stage to the 16 chop cycles at one position in the 7-point jiggle map. When the chop stage moves to the right hand beam, the JIGSENSIG output falls. Conversely, when the chop moves to the left hand beam JIGSENSIG rises. This is assumed to be due to the cross coupling between the chop and jiggle stages. The jiggle feedback control cannot correct for this within the period of half a chop cycle and therefore never reacquires the target position before the next chop movement. This general behaviour is typical for all positions in the 7-point map as can be seen in Figure 12 where the Chop-Jiggle trajectory plots show the mirror approaching the target position from the top and in Figure 9 where the trajectory approaches the target from below. The typical peak-to-peak amplitude of the disturbances is 570 CHOPSENSIG ADU (see Table 2). The variation across the seven positions is relatively small (~ ±7%).

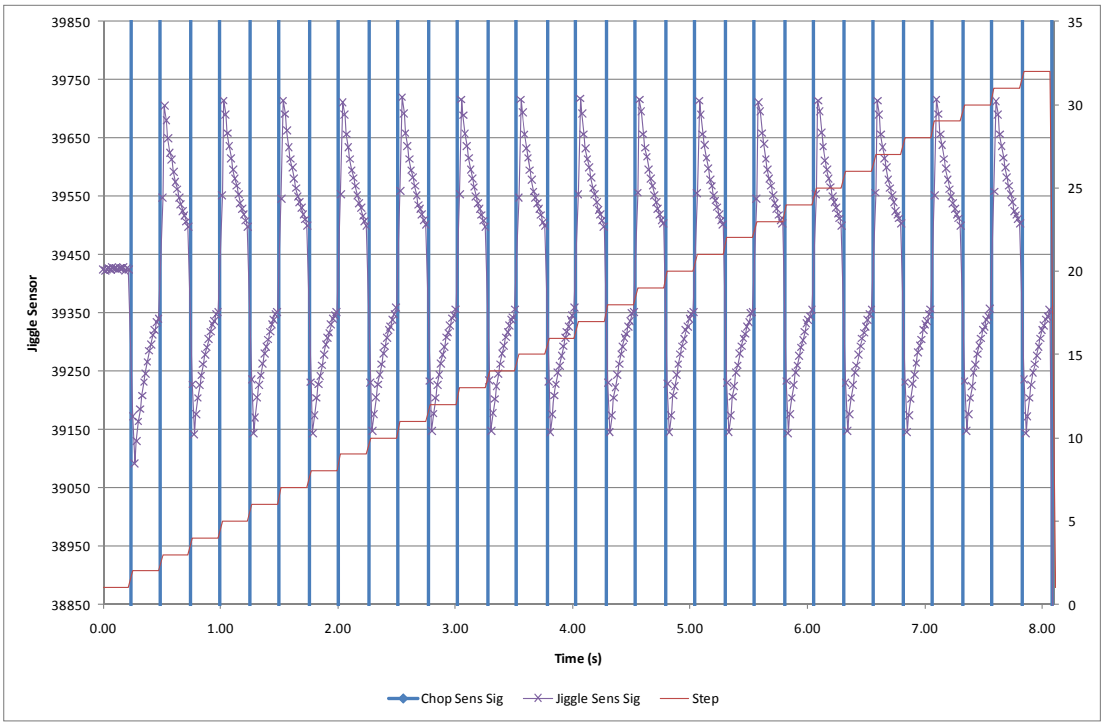

Figure 8: Observed Chop-to-Jiggle cross coupling

Table 2: Variation of the Chop-to-Jiggle cross coupling versus position

Position	Pk-Pk Jiggle Cross Coupling Error
North-East	530
North	548
South-East	558
Centre	571
North-West	583
South	590
South-West	612
Average	570.3
Deviation of Max from mean	7.3%
Deviation of Max from mean	-7.1%

4.2.2 Chop Anomalies

The anomalies observed in the behaviour of the chop stage are more complex than the jiggle stage and depend on the details of the chop movement.

In general, for both the chop directions, there is a long time constant of the settling which is almost certainly associated with the Ki term. To reduce the time constant, CHOPKI probably needs to be increased.

4.2.2.1 Left Beam to Right Beam

When switching from the left hand beam to the right hand beam, the chop stage rapidly overshoots the target by several hundred ADU. For the North-East and South-East positions, there is then a rapid rebound back past the target position followed by a relatively slow recovery back to the target position which does not converge within the dwell time at the chop location. The Central, North and South chop show a similar, but less pronounced behaviour. The North-West and the South-West locations show the rapid over shoot similar to the other locations, but the rebound does not take the stage back past the target location. These phenomena can be seen in Figure 9 and Figure 10.

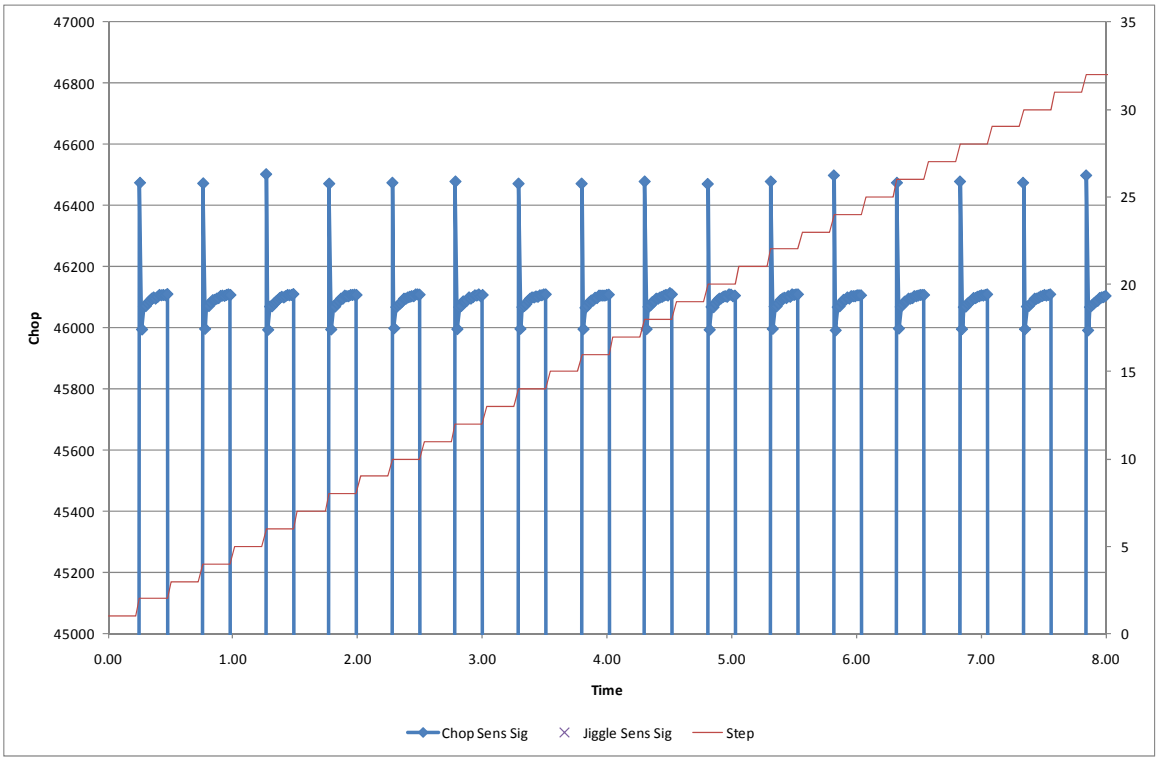

Figure 9: Observed Chop to right hand beam (centre location)

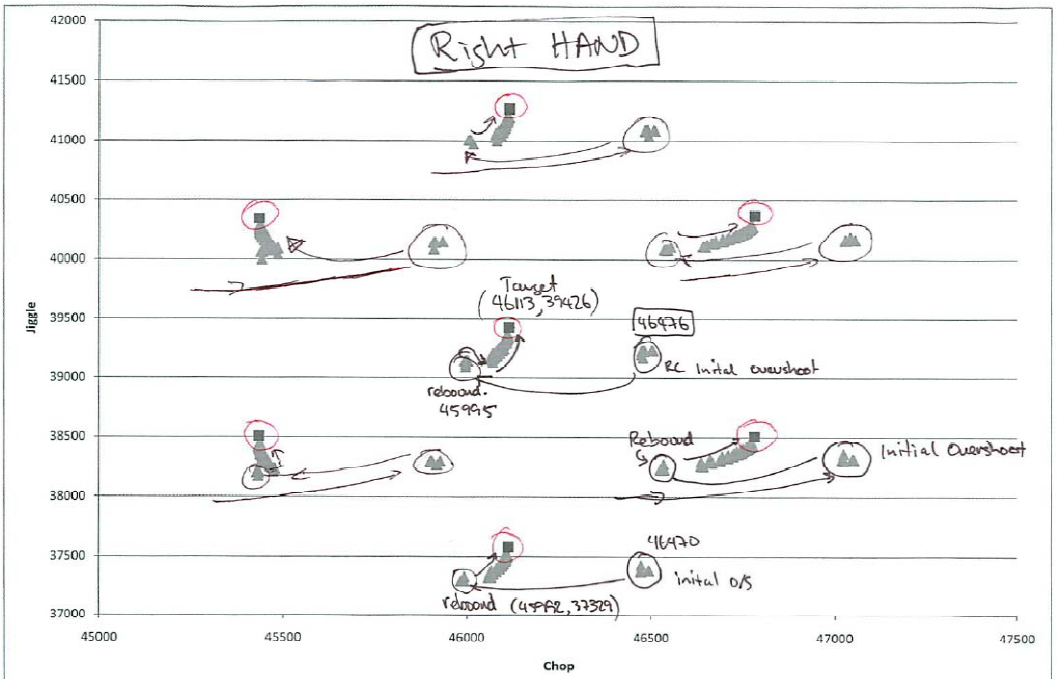

Figure 10: Chop and Jiggle trajectories on right hand beam

4.2.2.2 Right Beam to Left Beam

There are two trajectories for the mechanism to follow when moving from the right beam to the left beam, (1) the **first** movement of the target location in the series of sixteen chops where the mechanism moves from one chop pair position on the right hand beam to another chop-pair position on the left hand beam, (2) the subsequent fifteen chop cycles between the chop pair positions. This means that in general there are two sets of chop-jiggle trajectories per position in Figure 14 while there is one per position in Figure 10. If we neglect for the moment, the first trajectory in the sixteen chop cycles, the following behaviour is observed:

- South-West and North-West Positions (see e.g. Figure 11):
  - Initial overshoot by ~250 ADU followed by
  - a slow recovery towards the target location
- South, Centre and North Positions (See e.g. Figure 12):
  - Initial overshoot by ~75 ADU, followed by
  - a quick rebound towards the target, followed by
  - and a slow recovery towards the target
- North-East and South-East Positions (See e.g. Figure 13):
  - Initial undershoot by ~100 AUD
  - A rapid rebound away (Sic.) from the target!!
  - A slow recovery towards the target

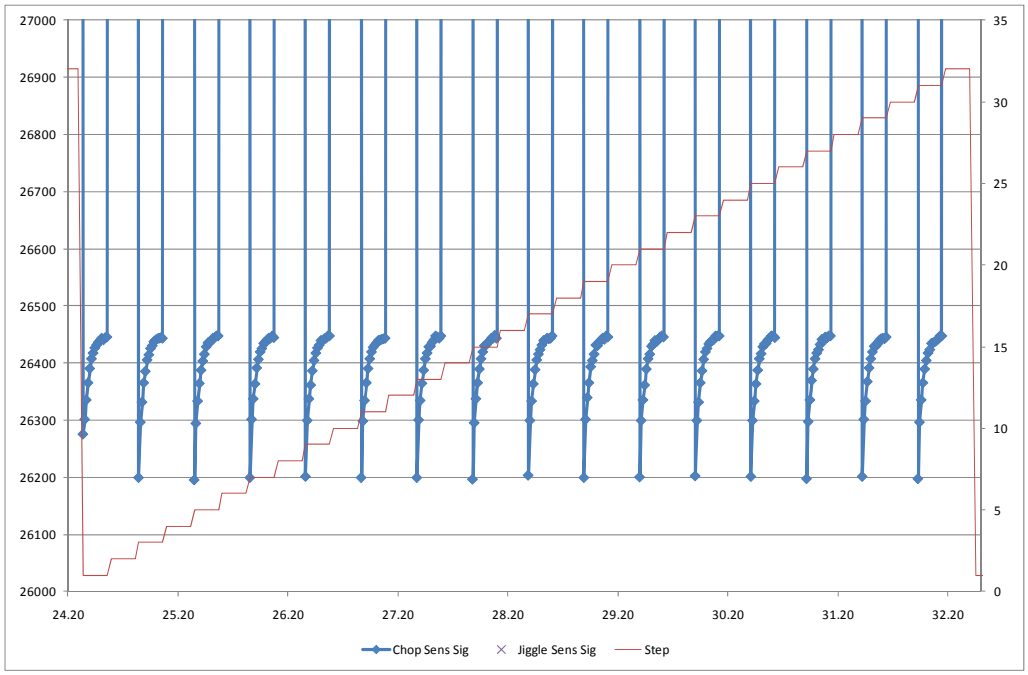

Figure 11: Observed Chop to left hand beam (North-West position)

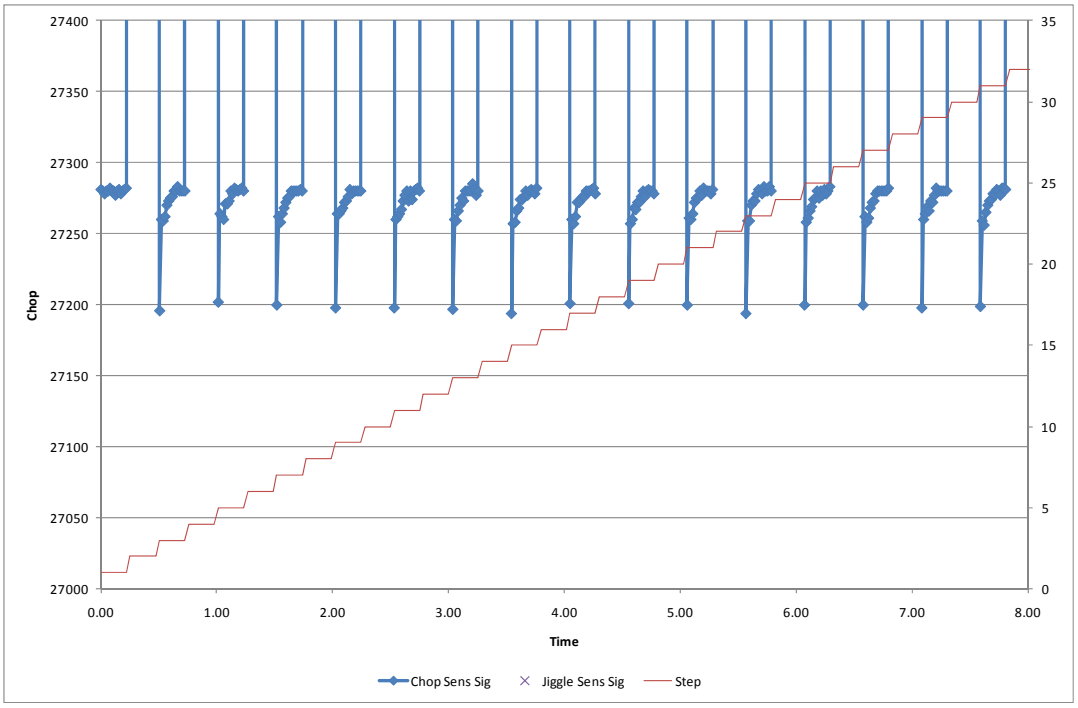

Figure 12: Observed Chop to left hand beam (Centre position)

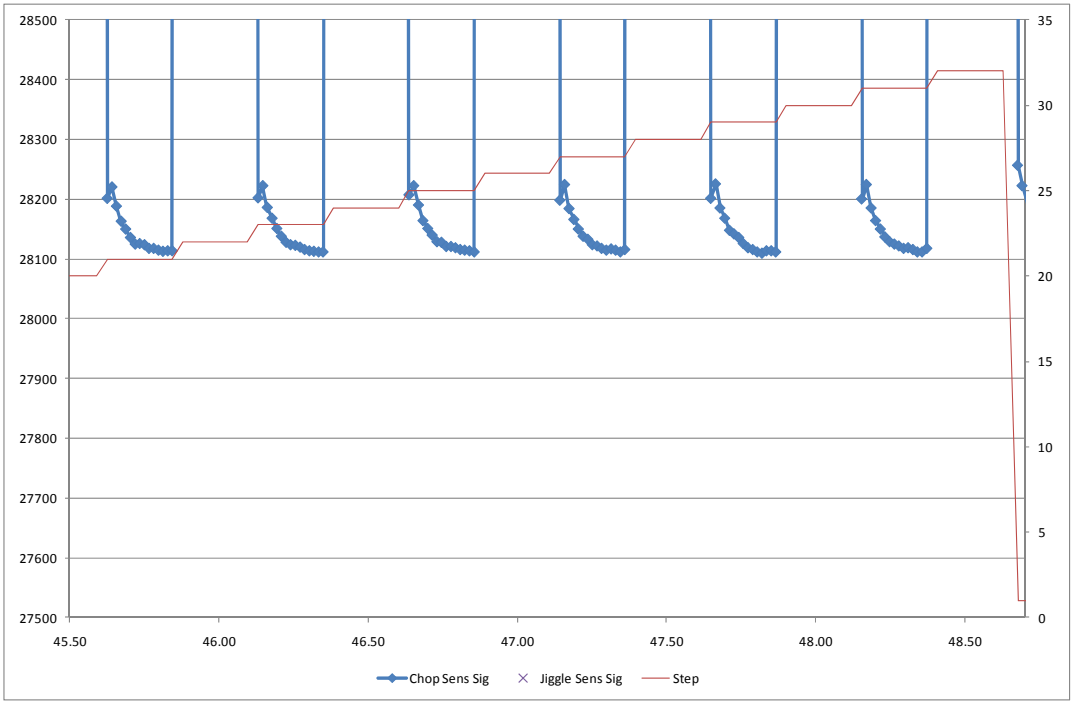

Figure 13: Observed Chop to left hand beam (North-East position)

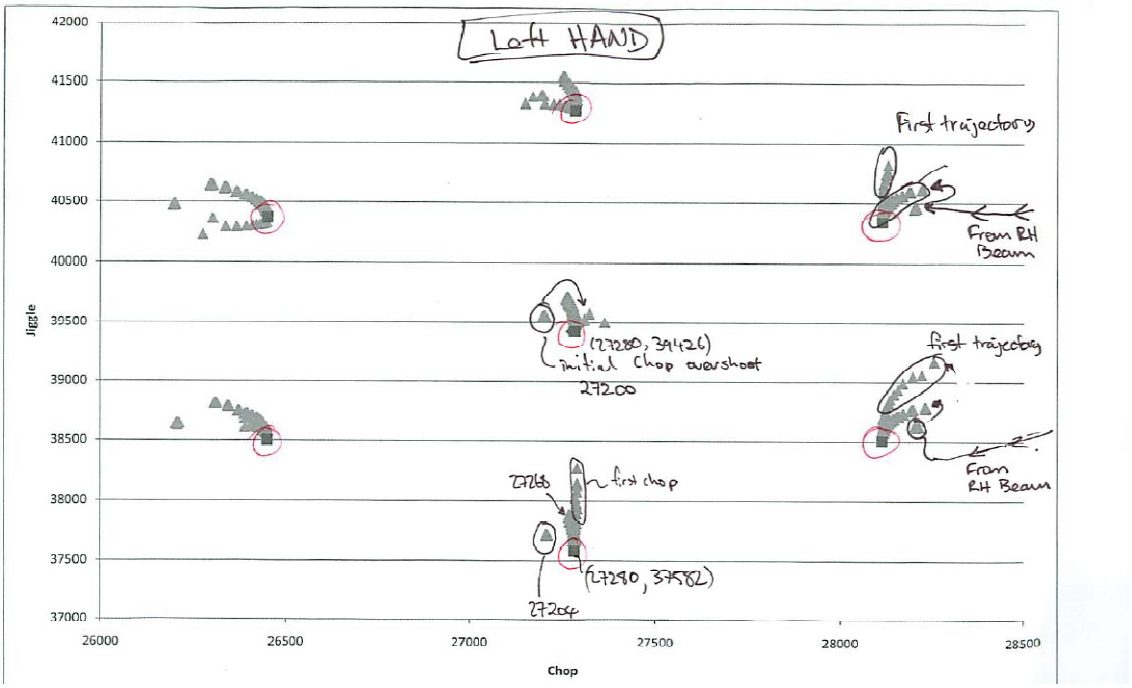

Figure 14: Chop and Jiggle trajectories on left hand beam

5. Overview of the control algorithm

The software which controls the SMEC and BSM runs on a DSP 21020 in the MCU. The main routine is *tmzhi.asm* which cycles through 20 steps sequentially every 420μs. The sampling rate of the controller is thus 420μs. One of these 20 steps performs the calculations for the chop axis controller (*choppid.asm*) and another is used to perform the jiggle calculations (*jigpid.asm*). Both of the PID control assembler source codes are included in the appendices for information. There is also a routine to compute the trajectory of the BSM which can be configured to limit the maximum slew rate in either of the axes.

The control algorithms can be configured to run in (1) open loop, (2) closed loop using the magneto-resistive sensors or (3) a back-up mode using the back emf of the BSM motors as the control reference. The purpose of this note is to describe the normal closed loop operation of the control algorithm using the magneto-resistive sensor and propose operational schemes to improve the performance of it. As such, it is the only control mode which will be considered in the following description of the control software.

5.1 Control terms

Figure 15 identifies the overall control logic of the software which for the chop stage. In the text below, the control logic for the jiggle stage is analogous with the chop except for a few differences which will be highlighted on a case by case basis.

There are three components (or terms) in the feedback control effort applied to the BSM motors; (1) cross-coupling compensation, (2) Feed Forward and (3) PID. These three terms are summed and re-scaled to the range of the 16-bit Digital-to-Analogue converter which controls the drive current supplied to the motors.

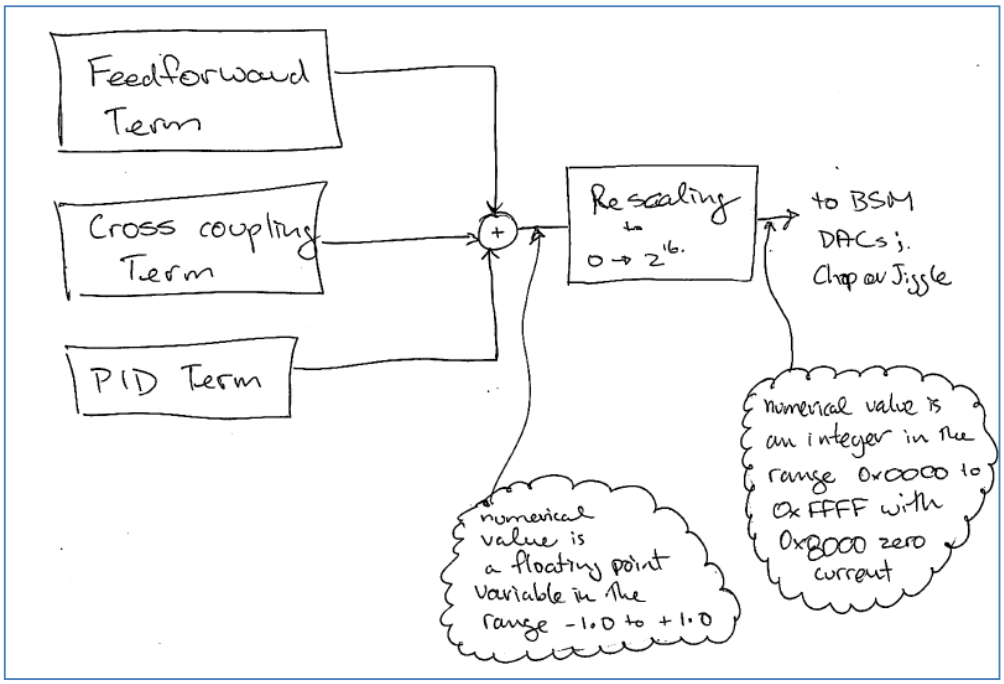

Figure 15: Overall control algorithm showing the three components of the control effort

5.2 Cross Coupling Compensation

5.2.1 Jiggle-to-Chop

Figure 16 shows the logic used to compute the control effort to compensate for the Jiggle-to-Chop cross coupling. The amount of compensation applied to the chop motor is proportional to the instantaneous target position of the jiggle stage. The variable JIGGLECHOPCOUPLE is a user configurable control parameter. A value of 0x8000 corresponds to no cross-coupling compensation. When it is set to a value greater than 0x8000 then increasing the commanded jiggle target position (i.e. towards an eventual larger JIGGSENSIG) will increase the current supplied to the chop motor and tend to drive the chop stage towards a larger CHOPSENSIG position. The opposite effect is achieved for a value less than 0x8000.

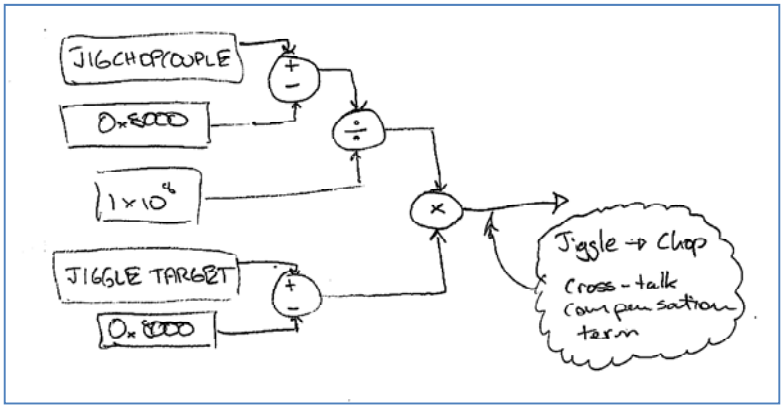

Figure 16: Jiggle-to-Chop cross-coupling term

5.2.2 Chop-to-Jiggle

The logic for the Chop-to-Jiggle compensation is more complex than the Jiggle-to-Chop algorithm (Figure 17) in so far as there is (1) a proportional and (2) a derivate component to the term.

The proportional term applies a corrective force proportional to the chop Feed Forward term calculated (and stored in memory) in the *choppid.asm* sub-program. The proportionality constant between Chop Feed Forward control and the Chop-to-Jiggle cross coupling compensation can be adjusted via the CHOPJIGGCOUPLE control parameter. (N.B. This will mean that changing the Chop Feed Forward term to tune the **Chop** axis will change the Chop-to-Jiggle compensation of the **Jiggle** stage and cannot therefore be adjusted independently!)

The derivative component applies a correction proportional to the low-pass filtered derivative of CHOPSENSIG calculated in the *choppid.asm* sub-program. The MCU documentation[1], states that this term is not implemented although it is clear that the code is present in the *jigpid.asm* sub-program. The proportionality constant can be scaled with the assembler code variable C2JDCROSSCOUPLING (see Appendix 2: *jigpid.asm*). The parameter is ***not*** present in the definition of the NHK file neither used during flight commissioning nor is it mentioned in the DPU-DRCU ICD. As this term does not appear in the top-level documentation, it is assumed that the value of the parameter will be set to the default loaded into DSP from MCU EPROM during the switch on and boot process.

In order to determine the *probable* value loaded into the PFM instrument, the FS was checked by changing the housekeeping table to report the value of this parameter. It was found that the default was set to 0x0000 on the FS instrument. This corresponds to the most negative value of this parameter. In order to remove any differential Chop-to-Jiggle coupling term, this term should be set to 0x8000.

[1] MCU-DPU Command List ICD, LAM.PJT.SPI.NOT.011011 Iss. 5

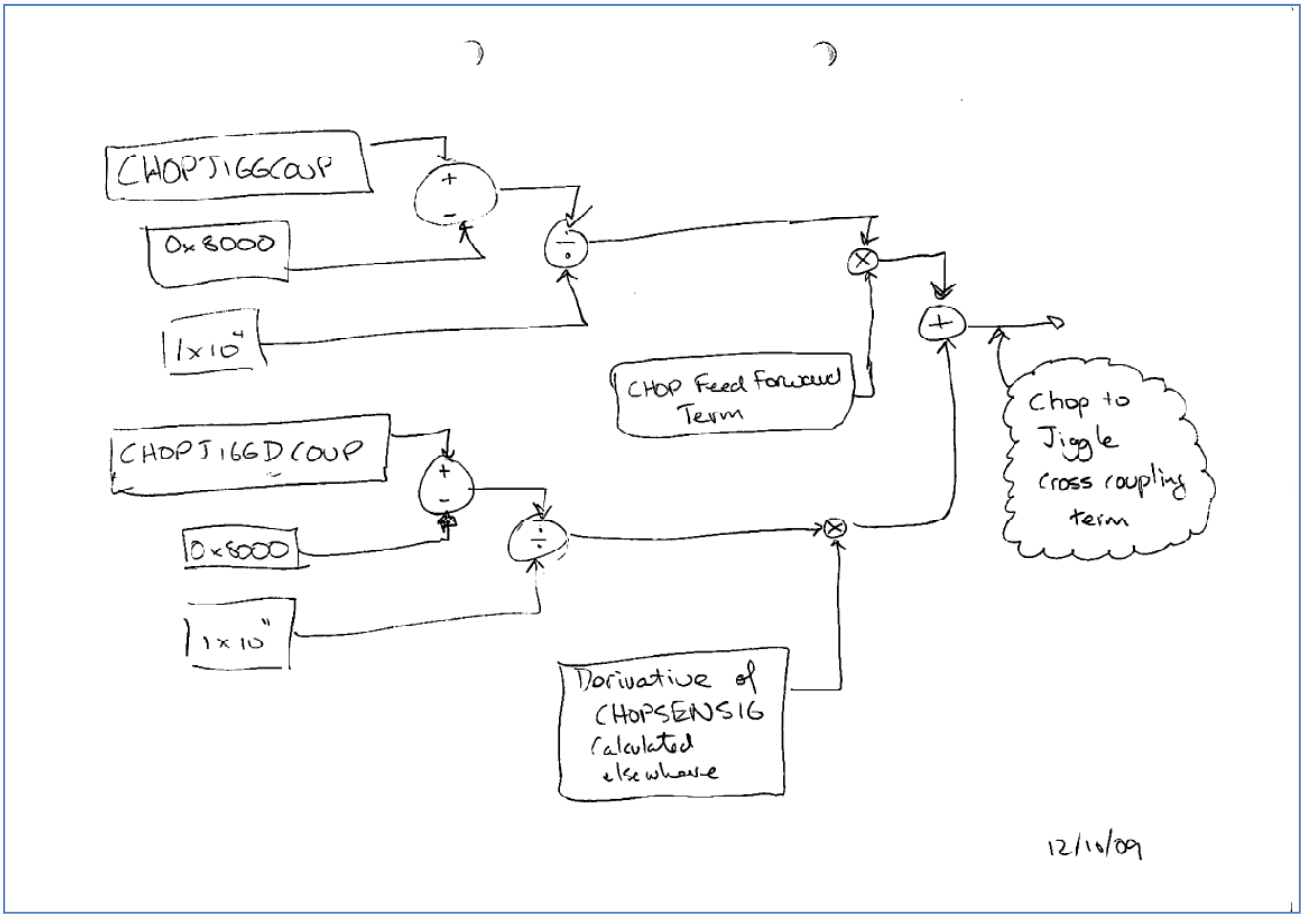

Figure 17: Chop-to-Jiggle cross-coupling term

5.3 Feed Forward Term

There are two components to the Feed Forward control effort. The first applies a control effort which is effectively an open loop term proportional to CHOPFFGAIN parameter. The offset between the zero value of the CHOPSENSIG (i.e. 0x8000 / 32768) and the zero optical/mechanical position is corrected by the term CHOPFFOFFSET. The CHOPFFOFFSET should be set to the value of CHOPSENSIG at the zero optical/mechanical position.

The second component is proportional to the derivate of the low-pass filtered chop trajectory target position. The characteristics of the low-pass filtered derivative algorithm are controlled by the CHOPDIFFTC1 and CHOPDIFFTC2 parameters. The values of these parameters are calculated with Bilinear Transform theory. The CHOPFFGAINDIFF parameter controls the proportionality of the term. The amplitude response of this function is plotted in Figure 19.

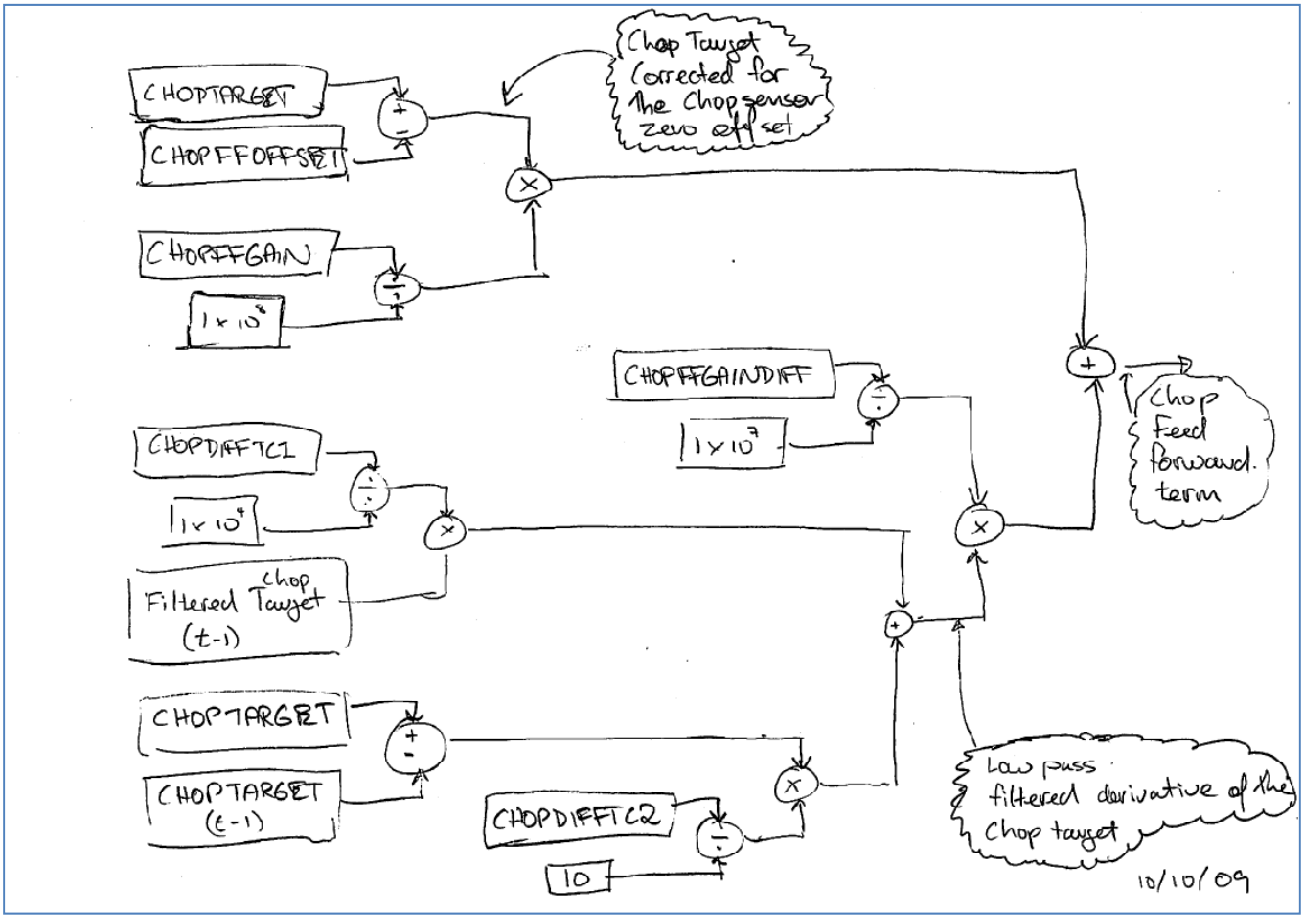

Figure 18: Feed forward term

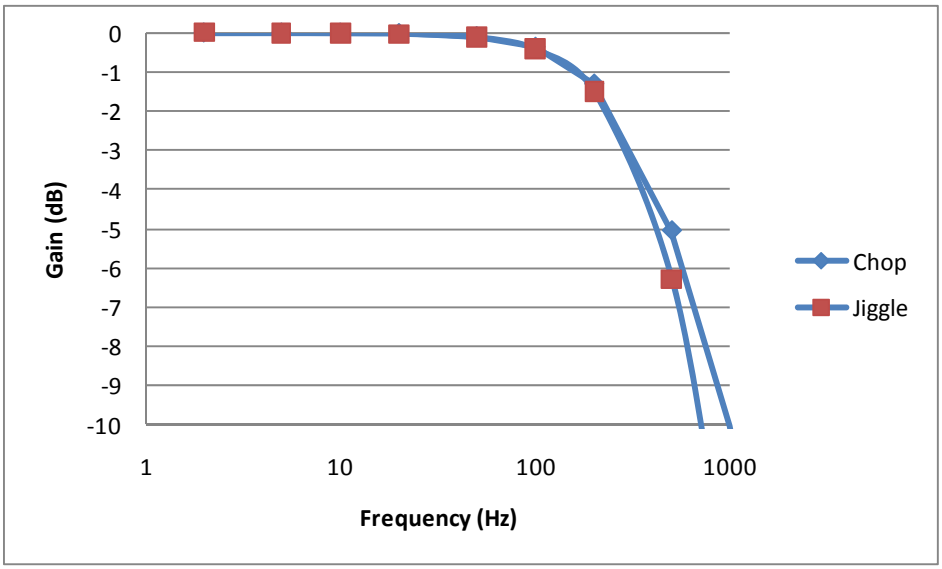

Figure 19: Gain plot of the derivative function

5.4 PID Terms

The PID control algorithm is illustrated below in Figure 20. The scaling of the proportional term is controlled by the CHOPKP parameter. The scaling of the derivative is controlled with the CHOPKD parameter. The calculation of the derivative of the servo error is made using the same method and parameters used in the derivative calculations in the Feed Forward control section of code. The integral term is controlled by the CHOPKI parameter. The integration is limited by an overall windup threshold as well as a limit on the maximum servo error which can be integrated.

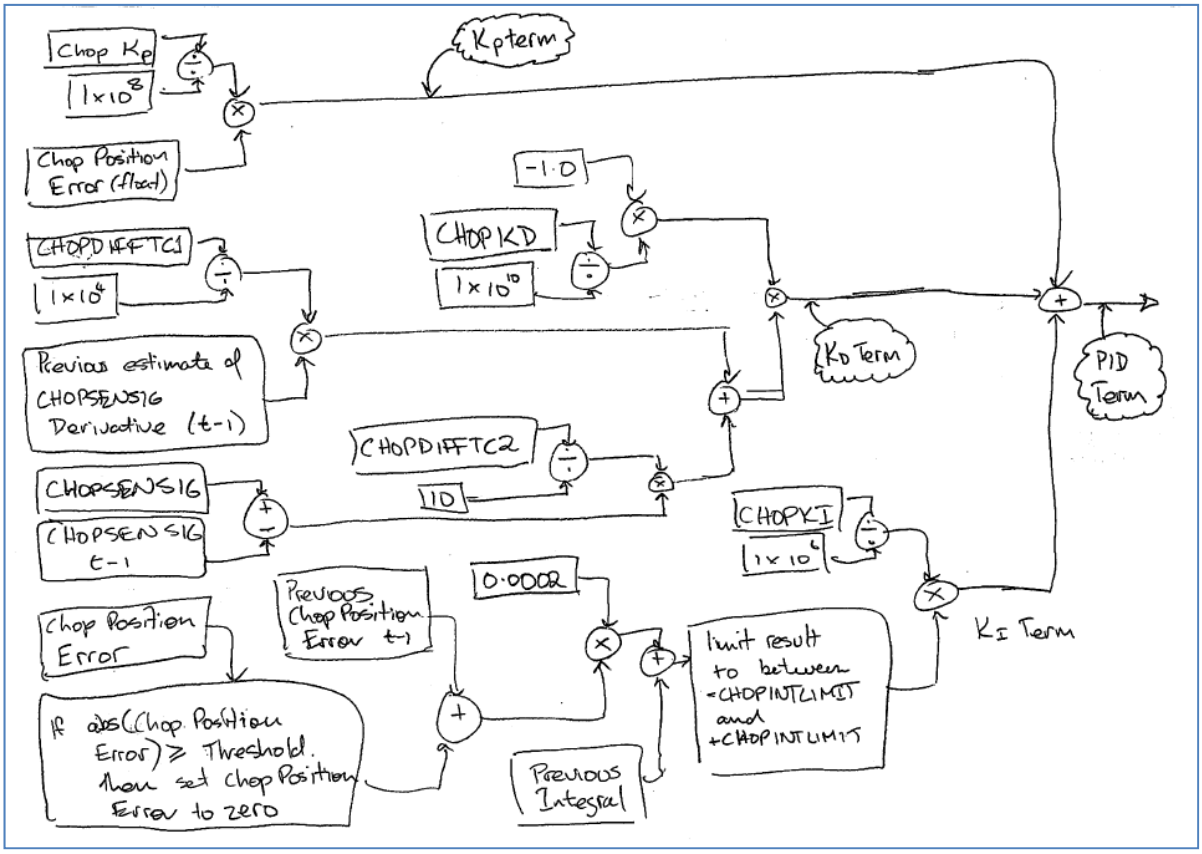

Figure 20: Chop PID term

Table 3: Key control parameters

	Chop Control Parameter	***Description***	***Existing Parameter***	***Proposed Value***
Jiggle to Chop Parameter	JIGCHOPCOUPLE	Adds a correction to the chop motor current based on the instantaneous target position of the jiggle axis (See Note 1)	0x8000 / 32768	It is recommended that this parameter remains as it is. (See Note 2)
Chop to Jiggle Parameters	CHOPJIGGCOUPLE	Adds a correction to the Jiggle motor current based on the Chop Feed Forward term	0x8000 / 32768	0x8161 (See Note 3)
	CHOPJIGGDCOUPLE	Adds a cross coupling compensation term proportional to the derivative of the chop sensor output (CHOPSENSIG)	0x0000 (See Note 4)	0x8000 (See Note 5)
Chop Feed Forward Control Parameters	CHOPFFOFFSET	Corrects for the offset between the optical zero chop position and corresponding CHOPSENSIG value at this position	0x8000 / 32768	37535 / 0x929F
	CHOPFFGAIN	Adds an open loop term	0x0770 / 1904	0x0BEB / 3051
	CHOPFFGAINDIFF	Adds an open loop term based on the derivative of the chop target. The number is an unsigned int	0	0
	CHOPDIFFTC1	Used in the calculation of the low-pass filtered derivative of chop target position	0x1000 / 4096	0x1000 / 4096
	CHOPDIFFTC2		0x208D / 8333	0x208D / 8333
Jiggle Feed Forward Control Parameters	JIGGFFOFFSET	As per Chop FF Description	0x8000 / 32768	0x9946 / 38238
	JIGGFFGAIN	As per Chop FF Description	0x0F6E / 3950	0x0BEB / 3051

	Chop Control Parameter	***Description***	***Existing Parameter***	***Proposed Value***
	JIGGFFGAINDIFF	Adds an open loop term based on the derivative of the chop target. The number is an unsigned int	0	0
	JIGGDIFFTC1	As per Chop FF Description	0x1A0B / 6667	0x1A0B / 6667
	JIGGDIFFTC2	As per Chop FF Description	0x208D / 8333	0x208D / 8333
Chop PID Parameters	CHOPKP	Proportional term	1000	TBD (See Note 6)
	CHOPKD	Differential term	3240	TBD (See Note 6)
	CHOPDIFFTC1 & CHOPDIFFTC2	Same parameter used in the Chop FF terms used here	See above	See above
	CHOPKI	Integral term	620	too low but TBC (See Note 6)
	CHOPINTREF	If the absolute servo error is greater than this threshold, then the error is not integrated. This parameter helps prevent overshoot.	0xFFFF / 65535	2560 / 0x0A00 (See Note 7)
	CHOPINTLIMIT	Limits the absolute magnitude of the total Ki term to prevent windup	0xFFFF / 65535	400 / 0x1024 (See Note 8)
Jiggle PID Parameters	JIGGKP	Proportional term	1500	TBD

	Chop Control Parameter	*Description*	*Existing Parameter*	*Proposed Value*
	JIGGKD	Differential term	7000	TBD
	JIGGDIFFTC1 & JIGGDIFFTC2	Same parameter used in the Jiggle FF terms used here	See above	See above
	JIGGKI	Integral term	500	TBD - too low?
	JIGGINTREF	? Is this the threshold above which the servo error is not integrated to prevent windup?	0xFFFF / 65535	1380
	JIGGINTLIMIT	? Is this the limit of the integrated servo error to prevent windup?	0xFFFF / 65535	

Notes:

1) The parameter is centred at 2^{15} to allow positive and negative coupling between the axes. The parameter is scaled by a factor of $1x10^{-8}$ inside the assembler code.
2) The rationale for this is that the time constant of the jiggle stage is much longer than the time constant of the chop stage and therefore the disturbances can probably be controlled adequately by the Chop PID.
3) When the BSM chops from the left to the right beam or (vice versa) the chop target changes as follows:

	RH Beam Chop Target	**LH Beam Chop Target**	**Delta**
North-East / South-East	46779	28111	18668
South / Centre / North	46113	27280	18833
North-West / South-West	45434	26448	18986
			Average: 18829

Figure 21 plots the shows the chop position versus chop motor (when the Jiggle stage has no power). The gradient of the best fit line is 1.18 [Sensor ADU/Drive DAC ADU]. Thus to effect the average chop throw of 18829, the chop motor has to be changed by 18829/1.18=15957 ADU. Assuming that the CHOPFFGAIN parameter is set so that all of this effort is generated by the FF term, the change in its value will be 15957/2^15 = 0.4870 or from -0.4870/2 to +0.4870/2.

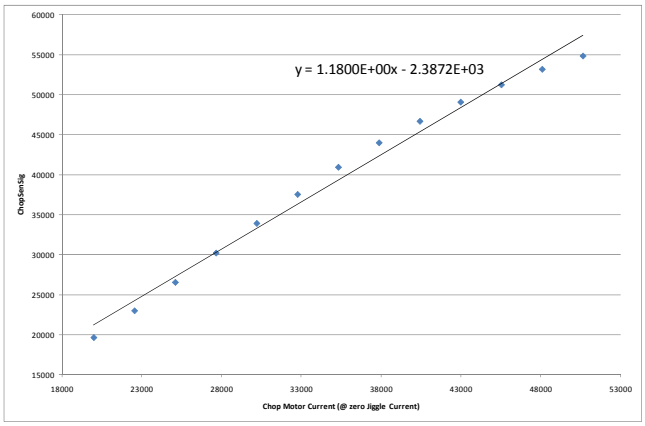

Figure 21: Chop rotation vs. Chop Current

The average Pk-Pk Jiggle cross-coupling is 570.3 ADU of CHOPSENSIG or 285.1 for a single throw. The slope of the Jiggle rotation vs. Jiggle motor current curve (Figure 22) has a slope of 0.5059 JIGSENSIG/JIG MOTOR CURRENT. Thus the jiggle motor current needs to be changed by 285.1/0.5059 = 563.7 ADU. Scaled to the units of the units of the controller summing point, this is 563.7/2^15=0.0172. Given that the assumed FF term changes by 0.4870 (see above), the CHOPJIGGCOUP parameter should be set to:

$0.0172/0.4870*1x10^4+2^{15}$ = 33121 or 0x8161

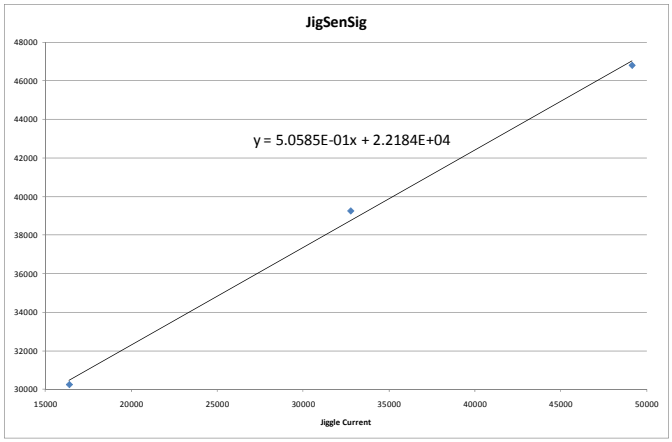

Figure 22: Jiggle rotation vs. Jiggle motor Current

When examining the open loop chop and jiggle coupling curves (Figure 7) the slope of the JIGGSENSIG vs. CHOPSENSIG curve for zero jiggle is -0.023028 and therefore a chop throw

of 18829 should induced a change in the jiggle position by 433.6 ADU. This is (433.6-285.1)/285.1 = 34% higher than the dynamic case and would correspond to a CHOPJIGGCOUP parameter of 0x8219. The cause of this discrepancy is not known but could indicate that the PID algorithm is being partially effective in mitigating the cross-talk. The recommended value is the dynamic one.

4) This value is not set by the uploading of telecommands and is therefore the default value stored in the MCU non-volatile memory. On the flight spare, the definition of the HK table was changed so that it would be reported. The value was found to be 0x0000 which corresponds to the most negative possible. This may not be a problem since the term is scaled by a factor of $1x10^{11}$ in *jigpid.asm* which could effectively mask the error.
5) In order to remove any differential effect, this value should be set to 0x8000
6) PID parameters to be optimise by scanning the matrix of parameter values
7) This parameter needs to be set to a much smaller value so that large servo errors are not integrated and induce large overshoot and/or oscillations. All of the chop values in Figure 10 and Figure 14 fall within a chop range of around ±260, and thus twice this value (520) would be an appropriate value.
8) The Ki term corrects for the fact that the gain of the Kp term is not high enough to reduce the servo error to zero. The CHOPINTLIMT term ensures that when the mechanism is a long way from the correct location, the error integral is not allow to grow in an unbounded fashion and introduce oscillations into the mechanism when it is close to the target position.
9) See Note (7). Max Jiggle error ~ 690, so let's say 1380 for this parameter

6. Tuning recommendations

6.1 Phase 1

Aim: Confirm optimal values for:

CHOPJIGGCOUP
CHOPFFGAIN
CHOPKD
JIGGFFGAIN
JIGGKD

Preparations

- Change the NHK definition so that the CHOPJIGGDCOUPLE is reported
- Set the rate that the BSM data is reported to the maximum frequency
- Change the table for the BSMT table so that CHOPPOSNERR is reported instead of CHOPMOTORVOLT
- Change the table for the BSMT table so that JIGGPOSNERR is reported instead of JIGGMOTORVOLT
- Chop and Jiggle should be in closed loop mode

.start.

```
/* tuning of CHOPFFGAIN, Kd and CHOPJIGGCOUPLE for 7 and 64 point jiggle maps*/
CHOPFFOFFSET  = 0x929F
JIGGLEFFOFFSET = 0x9946
JIGGLETARGET = 0x9946  // This should be the nominal zero current position
CHOPKP = 0x0000
CHOPKI = 0x0000

chopffgain_nominal = 0x0BEC
chopffgain_low = 0.6d
chopffgain_high = 1.4d
chopffgain_diff = 0.05d
/* corresponds to 17 values of FFGAIN */
chopkd_nominal = 0xCA8
chopkd_low = 0.2d
chopkd_high = 1.8d
chopkd_step = 0.2d
/* corresponds to 9 values of Kd */

7_point_left_chop = 27280d // check with Chris if this is the best setting
7_point_right_chop = 46113d // check with Chris if this is the best setting

Increment the ObsID  // tuning CHOPFFGAIN and CHOPKP for 7 point jiggle map

for (chopffloop =  chopffgain_low; chopffloop <= chopffgain_high; chopfflopp += chopffgaindiff)
{
        CHOPFFGAIN = chopffgain_nominal * chopffloop
        for (chopkdloop =  chopkd_low; chopkdloop <= chopkdhigh; chopkdloop += chopkdloop)
        {
                CHOPKD = chopkdnominal * chopkdloop
                set a flag to indentify the data (building block or step etc.)
                do 20 chops @ 2Hz between 7_point_left_chop & 7_point_right_chop // should last 10 sec
        }
}


Increment the ObsID  // tuning CHOPJIGGCOUPLE for 7 point jiggle map

CHOPKI = 0x26C
CHOPKP = 0x3E8
CHOPKD = 0xCA8
CHOPFFGAIN = 0xBEC

CJC_nominal = 0x8161  // estimated Chop to Jiggle Coupling
CJC_low = 0.989342109  // low end of test range
CJC_high = 1.010657891  // high end of test range
CJC_step = 0.001776315
/* corresponds to 13 values of CHOPJIGGCOUPLE between 0x8000 and 0x82C2

for (CJC_loop = CJC_low; CJC_loop <= CJC_high; CJC_loop += CJC_step)
{
        CHOPJIGGCOUPLE = CJC_nominal * CJC_loop
        set a flag to indentify the data (building block or step etc.)
        do 20 chops @ 2Hz between 7_point_left_chop & 7_point_right_chop // should last 10 sec
}


Increment the ObsID   // tuning CHOPFFGAIN and CHOPKP for 64 point jiggle map

64_point_left_chop = XXX // set to the nominal middle position of the left hand chop
64_point_right_chop = XXX // set to the nominal middle position of the right hand chop
CHOPKI = 0x0
CHOPKP = 0x0

for (chopffloop =  chopffgain_low; chopffloop <= chopffgain_high; chopfflopp += chopffgaindiff)
{
```

```
        CHOPFFGAIN = chopffgain_nominal * chopffloop
        for (chopkdloop =  chopkd_low; chopkdloop <= chopkdhigh; chopkdloop += chopkdloop)
        {
                CHOPKD = chopkdnominal * chopkdloop
                set a flag to indentify the data (building block or step etc.)
                do 20 chops @ 2Hz between 64_point_left_chop & 64_point_right_chop // should last 10 sec
        }
}

Increment the ObsID  // tuning CHOPJIGGCOUPLE for 64 point jiggle map

CHOPKI = 0x26C
CHOPKP = 0x3E8
CHOPKD = 0xCA8
CHOPFFGAIN = 0xBEC

CJC_nominal = 0x8161  // estimated Chop to Jiggle Coupling
CJC_low = 0.989342109  // low end of test range
CJC_high = 1.010657891  // high end of test range
CJC_step = 0.001776315
/* corresponds to 13 values of CHOPJIGGCOUPLE between 0x8000 and 0x82C2

for (CJC_loop = CJC_low; CJC_loop <= CJC_high; CJC_loop += CJC_step)
{
        CHOPJIGGCOUPLE = CJC_nominal * CJC_loop
        set a flag to indentify the data (building block or step etc.)
        do 20 chops @ 2Hz between 7_point_left_chop & 7_point_right_chop // should last 10 sec
}



/* tuning of JIGGFFGAIN and JIGKD */


CHOPTARGET = 0x929F    // should be nominal zero chop current position
JIGGKI = 0x0
JIGGKP = 0x0

JIGGFF_NOM = 0x17D8
JIGGFF_LOW = 0.6d
JIGGFF_HIGH = 1.4d
JIGGFF_STEP = 0.05d    // 17 levels

JIGGKD_NOM = 0x1B58
JIGGKD_LOW = 0.2d
JIGGKD_HIGH = 1.8d
JIGGKD_STEP = 0.2d     // 9 levels

7_point_bottom_Jigg = 37582d
7_point_top_Jigg = 41269d

for (Jiggffloop =  JIGGffgain_low; Jiggffloop <= JIGGffgain_high; Jiggffloop += JIGGffgaindiff)
{
        JIGGFFGAIN = JIGGffgain_nominal * Jiggffloop
        for (JiggKdloop =  JIGGkd_low; JiggKdloop <= JIGGkdhigh; JiggKdloop += JIGGkdloop)
        {
                JIGGKD = JIGGkdnominal * JiggKdloop
                set a flag to indentify the data (building block or step etc.)
                do 5 Jiggs @ 0.5Hz between 7_point_bottom_Jigg & 7_point_top_Jigg // should last 10 sec
        }
}

.end.
```

6.2 Phase 2

Aim: tune the chop and jiggle axes for

CHOPKI
CHOPINTLIMIT
CHOPINTREF
JIGGKI
JIGGINTLIMIT
JIGGINTREF

6.3 Phase 2

Aim: final retuning

7. Appendix 1: choppid.asm

Note: The code listed below is not configured and can be used for illustrative use only.

```
/*****************************************************************************************************/
/*                 choppid.asm                                                                       */
/* Written by Didier FERRAND :20/02/2003                                                             */
/* Issue:9.0                                                                                         */
/* Last modified: 05/03/2005                                                                         */
/* Associated Coff file: qm1march05.exe build by make.bat                                            */
/* Modification History:                                                                             */
/* Issue 8.0:                                                                                        */
/* - added anti wind up in integral control                                                          */
/* - replaced feef forward gain parameter instead of position scale factor                           */
/* 05/03/2005: added CFF_OFFSET on ff current instead of 0x8000  and ff_gain instead of pos_scale    */
/* Verified by DF: 05/03/2005                                                                        */
/*                                                                                                   */
/* Main Routine Description:                                                                         */
/*    PID and feedforward control of the BSM chopper axis                                            */
/* Internal subroutines:                                                                             */
/*    pid                                                                                            */
/*    feedforward                                                                                    */
/*    openloop                                                                                       */
/* External subroutines:                                                                             */
/*    none                                                                                           */
/*****************************************************************************************************/

/*****************************************************************************************************/
/*                                    INCLUDE FILES AND LIBRARIES                                    */
/*****************************************************************************************************/
#include "def21020.h" /* dsp chip constants                                                          */
#include "cmd.h"      /* command memonics                                                            */
/*****************************************************************************************************/
/*                                   CONSTANT DEFINED IN THE MODULE                                  */
/*****************************************************************************************************/

/*****************************************************************************************************/
```

```
/*                            GLOBAL ROUTINE DEFINED IN THIS MODULE                                */
/*************************************************************************************************/
.GLOBAL choppid;     /* this main pid routine                                                      */
/*************************************************************************************************/
/*                                  GLOBAL VARIABLES                                             */
/*************************************************************************************************/
.GLOBAL cservo_error,cxnmu,cynmu,ckieffectnmu,cffeffect;
/*************************************************************************************************/
/*                               PORTS DECLARATION   (only in initram)                           */
/*               in accordance with architecture described in macram.ach                         */
/*************************************************************************************************/

/*************************************************************************************************/
/*                                  GLOBAL PORTS (only in INITRAM)                               */
/*************************************************************************************************/

/*************************************************************************************************/
/*                                       EXTERNAL ROUTINES                                       */
/*************************************************************************************************/

/*************************************************************************************************/
/*                                       EXTERNAL VARIABLES                                      */
/*************************************************************************************************/
.EXTERN parameter_table;
.EXTERN choptraj;
.EXTERN jigtraj;
/*************************************************************************************************/
/*                                         EXTERNAL PORTS                                        */
/*************************************************************************************************/
.EXTERN dac2;

/**********  CHOPPER PID VARIABLES ***************************************************************/
.SEGMENT /DM    dm_sram;

.VAR cxnmu = 0.0;
.VAR cynmu = 0.0;
.VAR cservo_error;
.VAR ckieffect,ckpeffect,ckdeffect,cpideffect,cffeffect,ckieffectnmu;
.VAR c_pid_iin; /* value of the servo error to integrate in integral effect of the PID           */
.VAR c_pid_iinnmo; /* idem for t-1 sample                                                        */
.VAR c_pid_ioutl_p;/* output of the digital integration at t-1                                   */
.VAR mrs_adc;      /* float ADC value of the magnetoresistive sensor                             */
.VAR cffdiffnmo;   /* sample S(t-1) of the differential feedforward command                      */
.VAR ctrajnmo;     /* sample S(t-1) of the trajectory reference                                  */
.VAR chopmotcur;   /* chopper motor current float value                                          */
.VAR chopmotcurnmo;/* chopper motor current float value                                          */
.VAR backemfnmo;   /* sample e(t-1) of non ilterd back emf value                                 */
.VAR fbackemfnmo;  /* sample S(t-1) of the filtered backemf                                      */
.VAR diffcurnmo;   /* sample E(t-1) of motor current                                             */
.ENDSEG;
/*************************************************************************************************/

/***************************** BEGIN OF SUB PROGRAM ******************************************/
.SEGMENT /PM    pm_sram;             /* selected by PMS0~   program in ram*/

choppid:
/* Load position reference */
r11=dm(choptraj);      /* the trajectory reference is given in ADC units*/
f11=float r11 ;
/* Read MRS signal*/
b0=parameter_table;
m0=CHOP_MR_POS1;
r10=dm(m0,i0);
f10=float r10;
dm(mrs_adc)=f10;      /* save internally the magnetoresistive float adc value for pid*/

/* compute servo error value */
```

```
f9=f11-f10;
dm(cservo_error)=f9;  /* save internally the servo error float value for pid computation */
r1=fix f9;
r2=0xFFFF;
r1=r1 and r2;          /* keep the only 16 bits */
b0=parameter_table;    /* put position error in PTA for telemetry purpose */
m0=CHOP_POS_ERROR;
dm(m0,i0)=r1;

/***** READ THE CLOSED/OPEN LOOP MODE*******************************/
b0=parameter_table;
m0= CHOP_LOOP_MODE;
r15=dm(m0,i0);  /* load the command parameter related to loop mode */

/* TEST IF OPENLOOP */
LOOP_MODE0 :
r14=0;
r14=r15-r14;
if ne jump cloopmode1;
f8=0.0;
dm(ckpeffect)=f8;
dm(ckieffect)=f8;
dm(ckdeffect)=f8;
dm(cxnmu)=f8;/* reset the sample of the servo error for derivative / integral servo_error(t-1) */
dm(cynmu)=f8;
dm(c_pid_iinnmo)=f8;  /*reset integral input for next sample               */
dm(c_pid_ioutl_p)=f8; /* reset last integral limited output for t-1 sample */
dm(ckieffectnmu)=f8 ;
dm(cffeffect)=f8 ;
jump end_choppid ;            /* escape from routine: no new action,the dac value is steady */
/* TEST IF CLOSED LOOP ON MAGNETORESITIVE */
cloopmode1 :
r14=1;
r14=r15-r14;
if ne jump cloopmode3; /* perform closed loop using sensor position */
call cpid ;
jump send_to_dac2 ;


/* TEST IF OPEN LOOP NO SENSOR CONTROL USING FEEDFORWARD */
cloopmode3 :
r14=3;
r14=r15-r14;
if ne jump end_choppid; /* ELSE ESCAPE FROM ROUTINE: SMEC REMAINS AT STEADY POSITION */
call copenloop ;        /* call the subroutine for openloop control algorithm        */
jump send_to_dac2 ;     /* send the value on the dac                                 */

/*SEND COMMAND VALUE TO DAC#2 *******************************/
send_to_dac2: /* send the value contained in f8 to DAC2 */

/* before add the cross coupling effect from jiggle */
b0=parameter_table;
m0=J2CCROSSCOUPLING;
r9=dm(m0,i0);
r5=0x8000;
r9=r9-r5;         /* center the parameter to 32768 to allow negative values          */
f9=float r9;
f5=0.00000001;    /* 10e-8 scale factor on parameter                                 */
f9=f9*f5;         /* compute final crosscoupling coeff                               */
r5=dm(jigtraj);   /* load position reference of jiggle                               */
r6=0x8000;
r5=r5-r6;         /* substract 2^15 to get centered signed value                     */
f5=float r5 ;
f9=f9*f5;         /* compute final cross coupling command                            */
f8=f8+f9;         /* ad to chopper motor command to dac                              */
```

```
normalize_dac:
         /* limit the command to +/- 1.0 */
         f9=1.0;
         f1= abs f8;   /* take the absolute value of dac command                    */
         f1=f1-f9;     /* make the difference with max value of dac                 */
         if ge jump satur_dac; /* compare and saturate  if max reached              */
         jump dac_span; /* else send the value to dac                               */
satur_dac:
         f5=-1.0;
         f4= 1.0;
         f1=0.0;
         f1=f8-f1;            /* compute the sign of dac command                   */
         if ge f5=f4;         /* the sign is positive                              */
         f8=f5;               /* f8 is 1.0 or -1.0 depending on sign               */

dac_span:
         f9=1.0;
         f8=f8+f9;
         f9=65535.0/2.0;
         f8=f8*f9;
         r8=fix f8;
         dm(dac2)=r8; /* send value to dac2 */
         b0=parameter_table; /* memorise in parameter table for HK */
         m0=CHOP_DAC_VALUE;
         dm(m0,i0)=r8;



end_choppid:
rts;
nop;
nop;


/*************************************************************************/
/*********************** INTERNAL ROUTINES ******************************/
/*************************************************************************/
/* COMPUTE PID CONTROLLER ALGORITHM */
cpid:

f9=dm(cservo_error);
/*******PROPORTIONAL EFFECT **********************************/
b0=parameter_table;
m0=CKP;
r11=dm(m0,i0);      /* load the command related to loop mode                */
f11= float r11;     /* Kp 0.0002 typical                                    */
f1= 0.00000001;     /* 10-8 Kp parameter unit scale factor                  */
f11=f11*f1;
f8=f9*f11;          /* proportional command                                 */
dm(ckpeffect)=f8;
/******DERIVATIVE EFFECT*****************************************************/
/* compute ckdeffect= Kd*( tc2*(E(t)-E(t-1)) + tc1*S(t-1) )                  */
b0=parameter_table;
m0=CDIFF_FILTER_TC1;
r1=dm(m0,i0);    /* load the derivative gain tc1 param from parameter table */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter          */
f1=f1*f3;         /* compute differential gain tc1 value                    */
f2=dm(cynmu);     /* recall S(t-1) sample                                   */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                     */
f2=dm(mrs_adc);   /* load the sensor signal E(t)                            */
f4=dm(cxnmu);     /* recall the sensor signal E(t-1)                        */
f2=f2-f4;         /* compute the difference                                 */
b0=parameter_table;
```

```
m0=CDIFF_FILTER_TC2;
r1=dm(m0,i0);    /* load the derivative gain tc2 param from parameter table */
f1= float r1;
f4=0.1;           /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;          /* compute tc2*(E(t)-E(t-1))                             */
f3=f2+f3;          /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                */

/* SAMPLE MEMORISATION ******************************************************/
f9=dm(mrs_adc); /* load in f9 the mrs signal at t  for next sample (t-1)    */
dm(cxnmu)=f9;/* memorize the sample mrs signal for derivative (t-1)         */
dm(cynmu)=f3;    /* memorize the sample of the Kd effect (Kd effect (t-1))  */

b0=parameter_table;
m0=CKD;
r11=dm(m0,i0);    /* load the derivative gain parameter from parameter table*/
f11= float r11;   /* Kd 0.000002                                            */
f1= 0.0000000001; /* 10-10 Kd parameter scale factor                        */
f1=f1*f11;        /* compute Kd value                                       */
f2=-1.0;          /* invert sign                                            */
f1=f1*f2;         /*the derivative  acts on minus encoder position only     */

f3=f1*f3;         /* end of filtered derivative filter algorithm            */
dm(ckdeffect)=f3;


 /* INTEGRAL EFFECT **********************************************************/
f9=dm(cservo_error); /* load in f9 the servo error at t                     */
f6=abs f9;           /* take the absolute value of servo error              */
r6=fix f6;           /* convert absolute error servo into integer           */
b0=parameter_table;
m0=CKI_THRESHOLD;
r5=dm(m0,i0);      /* load the integral effect threshold from parameter table*/
r6=r6-r5;          /* compare the absolute servo error with threshold       */
if ge jump integral_limitation;
dm(c_pid_iin)=f9;  /* take the servo error for integration                 */
jump antiwindup;   /* no limit on integral effect                          */
integral_limitation:
f9=0.0;
dm(c_pid_iin)=f9;  /* limit the servo error integration                    */

/*anti windup : saturate on integral effect                                 */
antiwindup:
f6=dm(c_pid_iinnmo) ;       /* load in f6 the integration at t-1 sample     */
f6=f6+f9;           /* add integral samples at t and t-1                    */
f5=0.0002;          /* 0.5 * 400us coeff                                    */
f6=f5*f6;           /* apply digital integration coeff                      */
f5=dm(c_pid_ioutl_p);/* recall last limited digital integration result t-1  */
f6=f5+f6;            /* add the components -> c_pid_iout  in f6             */
f1=abs f6;           /* take the absolute value to check saturation         */
/* apply limitation on Ki command                                           */
m0=CKI_LIMIT;        /* load from parameter table the saturation value      */
r11=dm(m0,i0);
f11=float r11;       /* take float value of Ki limit                        */
f5=1.0;              /*  scale factor                                       */
f11=f11*f5;          /* multiply the limit parameter with scale factor      */
f1=f1-f11;           /* compare the saturation with actual value of integral*/
if ge jump saturate; /* saturate value in max value reached                 */
jump integral_gain;
saturate:
f5=-1.0;
f4= 1.0;
f1=0.0;
f1=f6-f1;            /* compute the sign of integral in                     */
if ge f5=f4;         /* the sign is positive                                */
f6=f5*f11;           /* c_pid_iout is the sign*saturation limit             */
```

```
/* apply final Ki parameter                                          */
integral_gain:
dm(c_pid_ioutl_p)=f6; /* memorize last integral limited output for t-1 sample */
b0=parameter_table;
m0=CKI;
r11=dm(m0,i0);      /* load the 16bit integer Ki parameter from parameter table*/
f11= float r11;     /* convert to float the Ki parameter of parameter table  */
f1= 0.000001;       /* 10-6 Ki scale factor on parameter unit                */
f11=f11*f1;         /* perform multiplication for conversion                 */
f6=f11*f6;           /* perform Ki command                                  */
dm(ckieffect)=f6;

/* integral effect sample memorisation ************************************/
f9=dm(c_pid_iin);
dm(c_pid_iinnmo)=f9;  /*memorize integral input for next sample              */
dm(ckieffectnmu)=f6 ;
/* PERFORM TOTAL COMMAND ON DAC */
send_cpid: /* sum of the parallel PID action */
f8=dm(ckpeffect);
f3=dm(ckdeffect);
f8=f8+f3;
f3=dm(ckieffect);
f8=f8+f3;
f3=1.0;   /* final scale factor of the PID command*/
f8=f8*f3;
dm(cpideffect)=f8 ;
/* ADD THE FEEDFORWARD COMPONENT*/
call cfeedforward ;
f3=dm(cffeffect);
f8=dm(cpideffect);
f8=f8+f3 ;
end_cpid :
rts ;
nop ;
nop ;

/**********************END OF PID*****************************************************/
/*******FEEDFORWARD CONTROL*************************************************************/
cfeedforward:
r11=dm(choptraj);      /* the trajectory reference is given in ADC units***********/
m0=CFF_OFFSET;
r12=dm(m0,i0);
r11=r11-r12;           /* substract programable 2^15 to get centered signed value */
f11=float r11 ;
b0=parameter_table;   /* load the feedforward coefficient in PTable***************/
m0=CFF_GAIN;
r1=dm(m0,i0);
f1= float r1;         /* convert to float*/
f2= 0.00000001;       /* apply 10-8 unit scale factor*/
f1=f1*f2;             /* compute scale factor        */
f8=f11*f1;
/* f8 contains the proportional feedforward command */

/* feedforward differential command= ffdiffgain*( tc2*(E(t)-E(t-1)) + tc1*S(t-1) ) */
b0=parameter_table;
m0=CDIFF_FILTER_TC1;
r1=dm(m0,i0);    /* load the derivative gain tc1 param from parameter table */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter          */
f1=f1*f3;         /* compute differential gain tc1 value                    */
f2=dm(cffdiffnmo);/* recall S(t-1) sample                                   */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                     */
f2=f11;           /* load the trajectory reference E(t)                     */
f4=dm(ctrajnmo);  /* recall the trajectory reference E(t-1)                 */
f2=f2-f4;         /* compute the difference                                 */
b0=parameter_table;
m0=CDIFF_FILTER_TC2;
```

```
r1=dm(m0,i0);     /* load the derivative gain tc2 param from parameter table */
f1= float r1;
f4=0.1;           /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;         /* compute tc2*(E(t)-E(t-1))                              */
f3=f2+f3;         /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                 */
b0=parameter_table;
m0=CFF_DIFFGAIN;
r5=dm(m0,i0);     /* load the derivative gain parameter from parameter table */
f5= float r5;     /* convert to float                                       */
f1= 0.0000001;    /* differential ff gain parameter scale factor            */
f1=f1*f5;         /* compute feedforward differential parameter             */
f3=f1*f3;         /* compute feedforward differential command               */
dm(cffdiffnmo)=f3;/* save S(t-1) sample                                     */
dm(ctrajnmo)=f11; /* save E(t-1) trajectory reference sample                */

f8=f8+f3;         /* sum of the proportional and differential feedforward command */
dm(cffeffect)=f8 ;/* save the command value                                 */
rts ;
nop ;
nop ;

/*****************************************************************************/
/*******OPEN LOOP + BACK EMF CONTROL (no sensor)*****************************/
copenloop:

/* feedforward command computation */
r11=dm(choptraj);      /* the trajectory reference is given in ADC units*****************/
m0=CFF_OFFSET;
r12=dm(m0,i0);
r11=r11-r12;           /* substract programmable 2^15 to get centered signed value ******/
f11=float r11 ;
b0=parameter_table;   /* load the feedforward coefficient in PTable**********************/
m0=CFF_GAIN;
r1=dm(m0,i0);
f1= float r1;         /* convert to float************************************************/
f2= 0.00000001;      /* apply 10-8 unit scale factor************************************/
f1=f1*f2;             /* compute scale factor        ***********************************/
f8=f11*f1;           /* f8 contains the open loop feed forward command *****************/

/* add chopper backemf effect */
b0=parameter_table;
m0=CHOP_MOTOR_VOLTAGE;
r2=dm(m0,i0);
r1=0x8000;
r2=r2-r1;      /* 0=0V substract to obtain signed number*/
f14=float r2;   /* f14 contains the motor volatef float value */
m0=CHOP_MOTOR_CURRENT;
r2=dm(m0,i0);
r1=0x8000;
r2=r2-r1;      /* 0=0V substract to obtain signed number*/
f10=float r2;
dm(chopmotcur)=f10; /* save chopper motor current */

m0=CHOP_MOT_RESIST;
r12=dm(m0,i0);
f12=float r12;  /* apply the difference between current and voltage (RI) */
f7= 0.0001;
f12=f12*f7;
f10=f10*f12;   /* Ri computation */
f14=f14-f10;   /* substract the RI from voltage to obtain back emf f14 contains the backemf command */

/* take into account inductance of motor ***********************************/
/* compute filtered current differential ( tc2*(E(t)-E(t-1)) + tc1*S(t-1) ) */

b0=parameter_table;
m0=CDIFF_FILTER_TC1;
```

```
r1=dm(m0,i0);    /* load the derivative gain tc1 param from parameter table */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter          */
f1=f1*f3;         /* compute differential gain tc1 value                    */
f2=dm(diffcurnmo);/* recall S(t-1) sample                                   */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                     */
f2=dm(chopmotcur);/* load the motor current E(t)                            */
f4=dm(chopmotcurnmo);/* recall the motor current E(t-1)                     */
f2=f2-f4;         /* compute the difference                                 */
b0=parameter_table;
m0=CDIFF_FILTER_TC2;
r1=dm(m0,i0);    /* load the derivative gain tc2 param from parameter table */
f1= float r1;
f4=0.1;           /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;          /* compute tc2*(E(t)-E(t-1))                             */
f3=f2+f3;          /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                */

/* SAMPLE MEMORISATION ***************************************************/
f2=dm(chopmotcur); /* load in f2 the mot current at t  for next sample (t-1)*/
dm(chopmotcurnmo)=f2;/* memorize the sample mrs signal for derivative (t-1) */
dm(diffcurnmo)=f3;/* memorize the sample of the filtered output S(t-1)      */

m0=CHOP_MOT_INDUCT;
r12=dm(m0,i0);
f12=float r12;  /* apply the difference on diff current                    */
f7= 0.0001;
f12=f12*f7;
f3 = f3*f12;   /* Ldi computation                                          */
f14=f14-f3;    /* substract the Ldi from voltage to obtain back emf f14 contains the backemf command */


m0=CHOP_BEMF_GAIN;
r12=dm(m0,i0);
f12= float r12; /* apply the feedback gain of back emf on control          */
f7=0.0000001;
f12=f12*f7;
f14=f14*f12;    /* f14 contains the unfiltered back emf feedback command    */

/* filter of the backemf*************************************************/
/* compute filtered backemf              ( t1*(E(t)-E(t-1)) + t2*S(t-1) )   */
b0=parameter_table;
m0=CBEMF_RATE_FILT2;
r1=dm(m0,i0);    /* load the t2 param from parameter table                  */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter          */
f1=f1*f3;         /* compute differential gain tc1 value                    */
f2=dm(fbackemfnmo);/* recall S(t-1) sample                                  */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                     */

f2=f14;           /* load unfiltered backemf command E(t)                   */
f4=dm(backemfnmo);/* recall the unfiltered backemf E(t-1)                   */
f2=f2+f4;         /* compute the sum                                        */
b0=parameter_table;
m0=CBEMF_RATE_FILT1;
r1=dm(m0,i0);    /* load the filter coeff t1 param from parameter table     */
f1= float r1;
f4=0.000001;      /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;         /* compute tc2*(E(t)-E(t-1))                              */
f3=f2+f3;         /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                 */

/* SAMPLE MEMORISATION ***************************************************/
dm(fbackemfnmo)=f3;/* memorize the sample  for  S(t-1)                      */
dm(backemfnmo)=f14;/* memorize the sample of the filtered E(t-1)            */
/*f3 contains the filtered backemf command*********************************/
endopenloop:
```

```
f8=f8-f3;          /* compute the total open loop + back emf feedback command */
dm(cffeffect)=f8 ;
rts ;
nop ;
nop ;

/*************************************************************************/
.ENDSEG;
```

8. Appendix 2: jigpid.asm

Note: The code listed below is not configured and can be used for illustrative use only.

```
/*************************************************************************************************/
/*                 jigpid.asm                                                                    */
/* Written by Didier FERRAND :20/02/2003                                                         */
/* Issue:8.0                                                                                     */
/* Last modified: 18/09/2003                                                                     */
/* Associated Coff file: qm1mcu01.exe build by make.bat                                          */
/* Modification History:                                                                         */
/* Issue 8.0:                                                                                    */
/* - added anti wind up in integral control                                                      */
/* - replaced feed forward gain parameter instead of position scale factor                       */
/* Verified by xx: not verified                                                                  */
/*                                                                                               */
/* Main Routine Description:                                                                     */
/*    PID and feedforward control of the BSM jiggle axis                                         */
/* Internal subroutines:                                                                         */
/*    pid                                                                                        */
/*    feedforward                                                                                */
/*    openloop                                                                                   */
/* External subroutines:                                                                         */
/*    none                                                                                       */
/*************************************************************************************************/

/*************************************************************************************************/
/*                                    INCLUDE FILES AND LIBRARIES                                */
/*************************************************************************************************/
#include "def21020.h" /* dsp chip constants                                                      */
#include "cmd.h"      /* command memonics list                                                   */
/*************************************************************************************************/
/*                                   CONSTANT DEFINED IN THE MODULE                              */
/*************************************************************************************************/
/* none                                                                                          */
/*************************************************************************************************/
/*                              GLOBAL ROUTINE DEFINED IN THIS MODULE                            */
/*************************************************************************************************/
.GLOBAL jigpid;     /* this main pid routine                                                     */
/*************************************************************************************************/
/*                                  GLOBAL VARIABLES                                             */
/*************************************************************************************************/
.GLOBAL jservo_error,jxnmu,jynmu,jkieffectnmu;
/*************************************************************************************************/
/*                                PORTS DECLARATION   (only in initram)                          */
/*              in accordance with architecture described in macram.ach                          */
/*************************************************************************************************/
```

```
/*****************************************************************************************************/
/*                                  GLOBAL PORTS (only in INITRAM)                                   */
/*****************************************************************************************************/

/*****************************************************************************************************/
/*                                        EXTERNAL ROUTINES                                          */
/*****************************************************************************************************/
/* none                                                                                              */
/*****************************************************************************************************/
/*                                        EXTERNAL VARIABLES                                         */
/*****************************************************************************************************/
.EXTERN parameter_table; /* memory location of the command parameter table                           */
.EXTERN jigtraj;         /* value of the desired jiggle position generated from bsmmove             */
.EXTERN cynmu;           /* derivative of chopper sensor for cross coupling compensation            */
.EXTERN cffeffect;       /* feedforward command from chopper for cross coupling compensation        */
/*****************************************************************************************************/
/*                                         EXTERNAL PORTS                                            */
/*****************************************************************************************************/
.EXTERN dac3;            /* dac3 relates to jiggle axis power amplifier and motor input              */
/************************************  ROUTINE VARIABLES ************************************************/
.SEGMENT /DM    dm_sram;
/*****************************************************************************************************/
.VAR jxnmu = 0.0;   /* sample E(t-1) of the filtered derivative effect                               */
.VAR jynmu = 0.0;   /* sample S(t-1) of the filtered derivative effect                               */
.VAR jservo_error;  /* servo error (difference between desired position and sensor value)            */
.VAR jkieffect,jkpeffect,jkdeffect,jpideffect,jffeffect,jkieffectnmu;
.VAR j_pid_iin;     /* value of the servo error to integrate in integral effect of the PID          */
.VAR j_pid_iinnmo;  /* idem for t-1 sample                                                          */
.VAR j_pid_ioutl_p; /* output of the digital integration at t-1                                     */
.VAR j_mrs_adc;     /* float ADC value of the jiggle magnetoresistive sensor                        */
.VAR jffdiffnmo;    /* sample S(t-1) of the differential feedforward command                        */
.VAR jtrajnmo;      /* sample S(t-1) of the trajectory reference (desired position (t-1))           */
.VAR backemfnmo;    /* sample e(t-1) of non filtered back emf value                                 */
.VAR fbackemfnmo;   /* sample S(t-1) of the filtered backemf                                        */
.ENDSEG;
/*****************************************************************************************************/

/*****************************          BEGIN OF SUB PROGRAM      **********************************/
.SEGMENT /PM    pm_sram;            /* selected by PMS0~   program in ram*/
jigpid:
/* Load position reference ***************************************************************************/
r11=dm(jigtraj);      /* the trajectory reference is given in ADC units**************************/
f11=float r11 ;       /* convert the desired position reference into float number format **********/
/* Read MRS signal ***********************************************************************************/
b0=parameter_table;
m0=JIG_MR_POS1;       /* readin parameter table the value of the magnetoresistive sensor ADC value*/
r10=dm(m0,i0);        /* put the value in internal register ************************************/
f10=float r10;        /* convert the magnetoresistive signal value into float format**************/

dm(j_mrs_adc)=f10;    /* save internally the magnetoresistive float adc value for pid*************/

/* compute servo error value *************************************************************************/
f9=f11-f10;           /* make the difference between position reference and sensor value **********/
dm(jservo_error)=f9;  /* save internally the servo error float value for pid computation **********/
r1=fix f9;            /* convert to integer the servo value ************************************/
r2=0xFFFF;            /* load 16 bits constant *************************************************/
r1=r1 and r2;         /* keep the only 16 bits in the servo error value ************************/
b0=parameter_table;   /* put position error in PTA for telemetry purpose ***********************/
m0=JIG_POS_ERROR;
dm(m0,i0)=r1;

/********************************** READ THE IDLE/CLOSED/OPEN LOOP MODE*************************/
b0=parameter_table;
m0= JIG_LOOP_MODE;
r15=dm(m0,i0);        /* load the command parameter related to loop mode ***********************/
```

```
/* TEST IF OPENLOOP ********************************************************************************/
LOOP_MODE0 :
r14=0;
r14=r15-r14;
if ne jump jloopmode1; /* here the loop mode parameter is not idle (0) go to next case ************/
f8=0.0;                /* reset some memorised control parameters *******************************/
dm(jkpeffect)=f8;      /* reset the proportional effect ****************************************/
dm(jkieffect)=f8;      /* reset the integral effect ********************************************/
dm(jkdeffect)=f8;      /* reset the derivative effect ******************************************/
dm(jxnmu)=f8;          /* reset the derivative E(t-1)sample ************************************/
dm(jynmu)=f8;          /* reset the  derivative S(t-1)sample ***********************************/
dm(j_pid_iinnmo)=f8;   /* reset integral input for next sample  ********************************/
dm(j_pid_ioutl_p)=f8;  /* reset last integral limited output for t-1 sample ********************/
dm(jkieffectnmu)=f8 ;  /* reset of the integral command t-1 sample *****************************/
dm(jffeffect)=f8 ;     /* reset of the feedforward effect **************************************/
jump end_jigpid ;      /* escape from routine: no new action,the dac value is steady ***********/

/* TEST IF CLOSED LOOP ON MAGNETORESITIVE ***********************************************************/
jloopmode1 :           /* test if the jiggle loop mode is closed on magnetoresistive sensor *******/
r14=1;
r14=r15-r14;
if ne jump jloopmode3; /* if mode is 1 then perform closed loop using sensor position *************/
call jpid ;            /* perform the digital pid algorithm ************************************/
jump send_to_dac3 ;    /* send the related command to dac 3 dedicated for jiggle motor pwr amp ****/


/* TEST IF OPEN LOOP NO SENSOR CONTROL USING FEEDFORWARD ************************************/
jloopmode3 :
r14=3;
r14=r15-r14;
if ne jump end_jigpid;  /* ELSE ESCAPE FROM ROUTINE: JIGGLE REMAINS AT STEADY POSITION ************/
call jopenloop ;        /* call the subroutine for openloop control algorithm *******************/
jump send_to_dac3 ;     /* send the value on the dac ********************************************/

/*SEND COMMAND VALUE TO DAC#3 ***************************************************************/
send_to_dac3: /* send the value contained in f8 to DAC3 ***************************************/

/* before add the cross coupling effect from chopper */
b0=parameter_table;
m0=C2JCROSSCOUPLING;
r9=dm(m0,i0);
r5=0x8000;
r9=r9-r5;         /* center the parameter to 32768 to allow negative values          */
f9=float r9;
f5=0.0001;        /* 10e-8 scale factor on parameter                                 */
f9=f9*f5;         /* compute final crosscoupling coeff                               */
f5=dm(cffeffect) ;
f9=f9*f5;         /* compute final cross coupling command                            */
m0=C2JDCROSSCOUPLING;
r6=dm(m0,i0);
r5=0x8000;
r6=r6-r5;         /* center the parameter to 32768 to allow negative values          */
f6=float r6;
f5=0.00000000001; /* 10e-10 scale factor on parameter                                 */
f6=f6*f5;
f7=dm(cynmu);     /* load the derivative of chopper sensor                           */
f7=f7*f6;
f9=f9+f7;
f8=f8+f9;         /* ad to jiggle motor command to dac the cross coupling effect     */



        /* saturate the command to +/- 1.0 ***************************************************/
        f9=1.0;
        f1= abs f8;   /* take the absolute value of dac command ********************************/
```

```
         f1=f1-f9;     /* make the difference with max value of dac ****************************/
         if ge jump satur_jdac; /* compare and staurate  if max reached ************************/
         jump jdac_span;/* else send the value to dac ***********************************************/
satur_jdac:            /* saturation of the dac to + or - 1.0 depending on sign ******************/
         f5=-1.0;
         f4= 1.0;
         f1=0.0;
         f1=f8-f1;      /* compute the sign of dac command ***************************************/
         if ge f5=f4;   /* the sign is positive ************************************************/
         f8=f5;         /* f8 is 1.0 or -1.0 depending on sign ***********************************/

jdac_span:
         f9=1.0;
         f8=f8+f9;
         f9=65535.0/2.0;
         f8=f8*f9;

         r8=fix f8;
         dm(dac3)=r8; /* send value to dac3 ***************************************************/
         b0=parameter_table; /* memorise in parameter table for HK ****************************/
         m0=JIG_DAC_VALUE;
         dm(m0,i0)=r8;

end_jigpid:
rts;                   /* END OF ROUTINE ***************************************************/
nop;
nop;


/*****************************************************************************************************/
/************************ INTERNAL ROUTINES **********************************************************/
/*****************************************************************************************************/
/* COMPUTE PID CONTROLLER ALGORITHM ******************************************************************/
jpid:

f9=dm(jservo_error); /* load the jiggle servo error floating format value ***********************/
/*******PROPORTIONAL EFFECT ***************************************************************************/
b0=parameter_table;
m0=JKP;
r11=dm(m0,i0);      /* load the command related to loop mode ************************************/
f11= float r11;     /* Kp in float format*********************************************************/
f1= 0.00000001;     /* 10-8 Kp parameter unit scale factor ***************************************/
f11=f11*f1;
f8=f9*f11;          /* proportional command ******************************************************/
dm(jkpeffect)=f8;
/******DERIVATIVE EFFECT*****************************************************************************/
/* compute jkdeffect= Kd*( tc2*(E(t)-E(t-1)) + tc1*S(t-1) )                  ***********************/
b0=parameter_table;
m0=JDIFF_FILTER_TC1;
r1=dm(m0,i0);    /* load the derivative gain tc1 param from parameter table ***********************/
f1= float r1;
f3=0.0001;         /* scale factor of the derivative filter parameter        ***********************/
f1=f1*f3;          /* compute differential gain tc1 value                   ***********************/
f2=dm(jynmu);      /* recall S(t-1) sample                                  ***********************/
f3=f1*f2;          /* compute tc1*S(t-1) and store in f3                    ***********************/
f2=dm(j_mrs_adc); /* load the sensor signal E(t)                            ***********************/
f4=dm(jxnmu);      /* recall the sensor signal E(t-1)                       ***********************/
f2=f2-f4;          /* compute the difference                                ***********************/
b0=parameter_table;
m0=JDIFF_FILTER_TC2;
r1=dm(m0,i0);      /* load the derivative gain tc2 param from parameter table ***********************/
f1= float r1;      /* convert to floating format ************************************************/
f4=0.1;            /* scale factor of the derivative tc2 filter parameter   ***********************/
f1=f1*f4;          /* compute differential gain tc2 value                   ***********************/
f2=f1*f2;           /* compute tc2*(E(t)-E(t-1))                            ***********************/
f3=f2+f3;           /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)               ***********************/
/* SAMPLE MEMORISATION *******************************************************************************/
```

```
f9=dm(j_mrs_adc); /* load in f9 the mrs signal at t  for next sample (t-1)  **********************/
dm(jxnmu)=f9;/* memorize the sample mrs signal for derivative (t-1)         **********************/
dm(jynmu)=f3;    /* memorize the sample of the Kd effect (Kd effect (t-1))  **********************/
b0=parameter_table;
m0=JKD;
r11=dm(m0,i0);    /* load the derivative gain parameter from parameter table **********************/
f11= float r11;   /* Kd 0.000002                                            **********************/
f1= 0.0000000001; /* 10-10 Kd parameter scale factor                        **********************/
f1=f1*f11;        /* compute Kd value                                       **********************/
f2=-1.0;          /* invert sign                                            **********************/
f1=f1*f2;         /* the derivative  acts on minus encoder position only    **********************/
f3=f1*f3;         /* end of filtered derivative filter algorithm            **********************/
dm(jkdeffect)=f3;

 /* INTEGRAL EFFECT ***************************************************************************/
f9=dm(jservo_error); /* load in f9 the servo error at t                     **********************/
f6=abs f9;           /* take the absolute value of servo error              **********************/
r6=fix f6;           /* convert absolute error servo into integer           **********************/
b0=parameter_table;
m0=JKI_THRESHOLD;
r5=dm(m0,i0);      /* load the integral effect threshold from parameter table**********************/
r6=r6-r5;          /* compare the absolute servo error with threshold       **********************/
if ge jump integral_limitation;
dm(j_pid_iin)=f9;  /* take the servo error for integration                 **********************/
jump antiwindup;   /* no limit on integral effect                          **********************/
integral_limitation:
f9=0.0;
dm(j_pid_iin)=f9;  /* limit the servo error integration                    **********************/

/*anti windup : saturate on integral effect*************************************************/
antiwindup:
f6=dm(j_pid_iinnmo) ; /* load in f6 the integration at t-1 sample          **********************/
f6=f6+f9;           /* add integral samples at t and t-1                   **********************/
f5=0.0002;          /* 0.5 * 400us coeff                                   **********************/
f6=f5*f6;           /* apply digital integration coeff                     **********************/
f5=dm(j_pid_ioutl_p);/* recall last limited digital integration result t-1  **********************/
f6=f5+f6;           /* add the components -> j_pid_iout  in f6             **********************/
f1=abs f6;          /* take the absolute value to check saturation         **********************/
/* apply limitation on Ki command ***********************************************************/
m0=JKI_LIMIT;       /* load from parameter table the saturation value      **********************/
r11=dm(m0,i0);
f11=float r11;      /* take float value of Ki limit                        */
f5=1.0;             /*  scale factor                                       */
f11=f11*f5;         /* multiply the limit parameter with scale factor      */
f1=f1-f11;          /* compare the saturation with actual value of integral*/
if ge jump saturate; /* saturate value in max value reached                */
jump integral_gain;
saturate:
f5=-1.0;
f4= 1.0;
f1=0.0;
f1=f6-f1;           /* compute the sign of integral in                     */
if ge f5=f4;        /* the sign is positive                                */
f6=f5*f11;          /* j_pid_iout is the sign*saturation limit             */

/* apply final Ki parameter                                                */
integral_gain:
dm(j_pid_ioutl_p)=f6; /* memorize last integral limited output for t-1 sample */
b0=parameter_table;
m0=JKI;
r11=dm(m0,i0);     /* load the 16bit integer Ki parameter from parameter table*/
f11= float r11;    /* convert to float the Ki parameter of parameter table  */
f1= 0.000001;      /* Ki scale factor on parameter unit                      */
f11=f11*f1;        /* perform multiplication for conversion                  */
f6=f11*f6;         /* perform Ki command                                  */
dm(jkieffect)=f6;
```

```
/* integral effect sample memorisation ************************************/
f9=dm(j_pid_iin);
dm(j_pid_iinnmo)=f9;  /*memorize integral input for next sample              */
dm(jkieffectnmu)=f6 ;
/* PERFORM TOTAL COMMAND ON DAC */
send_jpid: /* sum of the parallel PID action */
f8=dm(jkpeffect);
f3=dm(jkdeffect);
f8=f8+f3;
f3=dm(jkieffect);
f8=f8+f3;
f3=1.0;   /* final scale factor of the PID command*/
f8=f8*f3;
dm(jpideffect)=f8 ;
/* ADD THE FEEDFORWARD COMPONENT*/
call jfeedforward ;
f3=dm(jffeffect);
f8=dm(jpideffect);
f8=f8+f3 ;
end_jpid :
rts ;
nop ;
nop ;
/***********************END OF PID*****************************************************/
/*******FEEDFORWARD CONTROL*********************************************************/
jfeedforward:
r11=dm(jigtraj);      /* the trajectory reference is given in ADC units*/
r12=0x8000;
r11=r11-r12;           /* substract 2^15 to get centered signed value */
f11=float r11 ;
b0=parameter_table;   /* load the feedforward coefficient in PTable*/
m0=JFF_GAIN;
r1=dm(m0,i0);
f1= float r1;         /* convert to float*/
f2= 0.00000001;       /* apply 10-8 unit scale factor*/
f1=f1*f2;             /* compute scale factor        */
f8=f11*f1;
/* f8 contains the proportional feedforward command */

/* feedforward differential command= ffdiffgain*( tc2*(E(t)-E(t-1)) + tc1*S(t-1) ) */
b0=parameter_table;
m0=JDIFF_FILTER_TC1;
r1=dm(m0,i0);    /* load the derivative gain tc1 param from parameter table */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter          */
f1=f1*f3;         /* compute differential gain tc1 value                    */
f2=dm(jffdiffnmo);/* recall S(t-1) sample                                   */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                     */
f2=f11;           /* load the trajectory reference E(t)                     */
f4=dm(jtrajnmo);  /* recall the trajectory reference E(t-1)                 */
f2=f2-f4;         /* compute the difference                                 */
b0=parameter_table;
m0=JDIFF_FILTER_TC2;
r1=dm(m0,i0);     /* load the derivative gain tc2 param from parameter table */
f1= float r1;
f4=0.1;           /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;         /* compute tc2*(E(t)-E(t-1))                              */
f3=f2+f3;         /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                 */
b0=parameter_table;
m0=JFF_DIFFGAIN;
r5=dm(m0,i0);     /* load the derivative gain parameter from parameter table */
f5= float r5;     /* convert to float                                       */
f1= 0.0000001;    /* differential ff gain parameter scale factor            */
f1=f1*f5;         /* compute feedforward differential parameter             */
f3=f1*f3;         /* compute feedforward differential command               */
dm(jffdiffnmo)=f3;/* save S(t-1) sample                                     */
```

```
dm(jtrajnmo)=f11; /* save E(t-1) trajectory reference sample               */

f8=f8+f3;         /* sum of the proportional and differential feedforward command */
dm(jffeffect)=f8 ;/* save the command value                                 */
rts ;
nop ;
nop ;

/*****************************************************************************/
/*******OPEN LOOP + BACK EMF CONTROL (no sensor)*************************************/
jopenloop:
r11=dm(jigtraj);      /* the trajectory reference is given in ADC units*/
r12=0x8000;
r11=r11-r12;           /* substract 2^15 to get centered signed value */
f11=float r11 ;
b0=parameter_table;   /* load the feedforward coefficient in PTable*/
m0=JIG_POSITION_SCALE;
r1=dm(m0,i0);
f1= float r1;         /* convert to float*/
f2= 0.00000001;       /* apply 10-8 unit scale factor*/
f1=f1*f2;             /* compute scale factor        */
f8=f11*f1;           /* f8 contains the open loop feed forward command */

/* add chopper backemf effect */
b0=parameter_table;
m0=JIG_MOTOR_VOLTAGE;
r2=dm(m0,i0);
r1=0x8000;
r2=r2-r1;      /* 0=0V substract to obtain signed number*/
f14=float r2;   /* f14 contains the motor volatef float value */
m0=JIG_MOTOR_CURRENT;
r2=dm(m0,i0);
r1=0x8000;
r2=r2-r1;      /* 0=0V substract to obtain signed number*/
f10=float r2;

m0=JIG_MOT_RESIST;
r12=dm(m0,i0);
f12=float r12;  /* apply the difference between current and voltage (RI) */
f7= 0.0001;
f12=f12*f7;
f10=f10*f12;   /* Ri computation */
f14=f14-f10;   /* substract the RI from voltage to obtain back emf */
m0=JIG_BEMF_GAIN;
r12=dm(m0,i0);
f12= float r12; /* apply the feedback gain of back emf on control */
f7=0.0000001;
f12=f12*f7;
f14=f14*f12;    /* f14 contains the back emf feedback command */


/* filter of the backemf*************************************************/
/* compute filtered backemf             ( t1*(E(t)-E(t-1)) + t2*S(t-1) )   */
b0=parameter_table;
m0=JBEMF_RATE_FILT2;
r1=dm(m0,i0);    /* load the t2 param from parameter table                */
f1= float r1;
f3=0.0001;        /* scale factor of the derivative gain parameter         */
f1=f1*f3;         /* compute differential gain tc1 value                   */
f2=dm(fbackemfnmo);/* recall S(t-1) sample                                 */
f3=f1*f2;         /* compute tc1*S(t-1) and store in f3                    */

f2=f14;           /* load unfiltered backemf command E(t)                  */
f4=dm(backemfnmo);/* recall the unfiltered backemf E(t-1)                  */
f2=f2+f4;         /* compute the sum                                       */
b0=parameter_table;
m0=JBEMF_RATE_FILT1;
```

```
r1=dm(m0,i0);    /* load the filter coeff t1 param from parameter table     */
f1= float r1;
f4=0.000001;      /* scale factor of the derivative gain parameter          */
f1=f1*f4;         /* compute differential gain tc2 value                    */
f2=f1*f2;         /* compute tc2*(E(t)-E(t-1))                              */
f3=f2+f3;         /* compute tc2*(E(t)-E(t-1)) + tc1*S(t-1)                 */

/* SAMPLE MEMORISATION ***************************************************/
dm(fbackemfnmo)=f3;/* memorize the sample  for  S(t-1)                      */
dm(backemfnmo)=f14;/* memorize the sample of the filtered E(t-1)            */
/*f3 contains the filtered backemf command*********************************/
endopenloop:
f8=f8-f3;          /* compute the total open loop + back emf feedback command */


dm(jffeffect)=f8 ;
rts ;
nop ;
nop ;

/************************************************************************/
.ENDSEG;
```