SPIRE-LAM-PRJ-002606

Herschel – SPIRE

MCU QM2 SOFTWARE DESIGN DESCRIPTION

Fichier: Software Design(QM2).doc

Prepared by:	Signature
D. Ferrand Date : 22/02/2006	
Approved by :	
Date :	
PA/QA	**Signature**
Gérard Rousset Date :	
Released by	**Signature**
Dominique Pouliquen Date :	

1 General design architecture

The control software run on a 21020 DSP chip with the following interface:

- 2 multiplex ADCs
- 3 DACs
- digital outputs
- parallel register interfacing command line FPGA
- parallel register interfacing telemetry high speed link FPGA

The DSP architecture is defined in the file: qm2updatedmay05\DEF21020.H

1.1 ADCs and related signals

1.1.1 Value of ADC as read by DSP

The 7805 ADC component provides the digital output as a binary two's complement value.
After digital value of DSP, a bit toggle on b15 is performed to obtain a 0-65535 range.

Signal voltage	ADC digital output	ADC value in DSP after 2' complement
+10V	7FFF	FFFF
0	0000	8000
-10V	8000	0000

The multiplexed ADCs allow acquisition of 13 signals coming from the SMEC and BSM motors and sensors as well as 8 HK signals related to board voltages and temeperatures.
To view allocation address of these channels, click on the following constant define file: `portdef.h`

1.2 DACs

The DSP can send 3 digital values to 3 DACS:

DAC1: -10/+10V SMEC Motor coil power amplifier input
DAC2: -10/+10V BSM Chopper Motor coil power amplifier input
DAC2: -10/+10V BSM Jiggle Motor coil power amplifier input

1.3 Digital outputs

The digital outputs are:

- Latch relay position
- Latch solenoid pulse triggering (front edge level).These 2 output control is described in the following source code:latch.asm
- LVDT oscillator on/off bit
- three bits to encode 8 levels for optical encoder led illumination
- the chopper magneto resistance current on/off bit
- the jiggle magneto resistance current on/off bit.

These other outputs are controlled by the following program: digital.asm

2 Software general architecture

From a cold reset, and when power is on, the software first run on a PROM with a limited software. At this stage, the software waits for 2 commands (copy program and start program) to run the actual application program which runs on RAM. So, the operations from switch on the DSP are:

1. Reset on PROM Chip
2. Start the boot program in PROM
3. Wait for a command to copy the application program from PROM to RAM
4. Copy the application program in RAM
5. Wait for a second reset command
6. Start the application program by sending a second reset command.

2.1 Boot program in PROM

The boot program in PROM :

- Set up the DSP registers
- Set DACs to 0 volt value, ie 0x8000 on digital port
- Verify RAM program and data memories, by writing and reading 0x55555555 and 0xAAAAAAAA
- Authorise a limited set of command to copy the application from EPROM to RAM and to verify proper operation.

The Boot program includes the following routines:

- bootb.asm : called on reset. This routine calls **dspconf** then **initram** and start timer for **tmzhi**
- cmdintb.asm: on IRQ, analyse the limited set of command received on serial link in this mode
- dspconfb.asm: configures the DSP internal registers and wait states for PROM
- initramb.asm: tests the RAM writing and reading 0xAAAAAAAA and then 0x55555555
- portdecb.asm: declaration of adresses of ports.
- romramb.asm: program which tranfers application program from PROM to RAM
- routirq1b.asm : interrupt program linked to the command line
- tmzhib.asm: main program waiting for external commands.

2.2 Application program

The application program call a boot subprogram which:

1. Call dspconf subroutine: general configuration set up of the DSP registers and wait states for RAM
2. Call initram subroutine: initialisation of variables, of the parameter table to default values and of specific tables such as sine interpolation table,
3. Set up authorised interrupts: command line, DSP timer
4. When finished, the program enters an idle loop waiting for the inner timer interrupt every 21 µs.

3 Main scheduling: TMZHI subprogram called every timer interrupt

To view the code click onthe following hypertext link: tmzhi.asm

This routine is executed each time the internal 21 us timer interrupt. This routine is the main scheduler of the program. A complete cycle is done in 20 steps*21 µs=420 µs. There are 13ADC signals to be read + voltages and temperatures (8 signals). The MAC board provides 2 ADCs so each step allow to read 2 analogue values every step. The sine and sine 120 of smec are over-sampled (they are read evry 2 elementary 21us timer interrupt cycle) in order to optimise the software zero crossing counting. The goal is to get acquisitions every 42 µs so the scheme is the following:

#	Timer (µs)	Operations and subprogram calls					Task duration
1	0 (400)	Read sine	Read sine 120	encpos	Read date on FPGA		9.70us
2	21	Read LVDTDC	Read LVDTAC	Smectraj	Compute LVDT position	sendtel	8.45 us
3	42	Read sine	Read sine120	Encpos			9.75 us
4	63	Read SMotcurrent	ReadSVoltage	Smecpid			14.3 us
5	84	Read sine	Read sine120	Encpos			10 us
6	105	Read sine240		latch	Digital I/0	sendtel	10.1 us
7	126	Read sine	Read sine120	Encpos			11.2 us
8	147	Read Choppos		BsmMove			9.65 us
9	168	Read sine	Read sine 120	Encpos			11.2 us
10	189	Read chop Mcurrent	Read Chop Voltage	ChopPid		sendtel	8.6 us min 16.95 us max
11	210	Read sine	Read sine 120	encpos			11.7 us
12	231	Read Jig Pos					5.9 us
13	252	Read sine	Read sine120	encpos			12.2 us
14	273	Read Jig Mcurrent	Read Jig Voltage	JigPID		sendtel	8.9 us
15	294	Read sine	Read sine 120	encpos			12.5 us
16	315	Read MAC/S/Btemp					7.1 us
17	336	Read sine	Read sine 120	encpos			12.6 us
18	357	Read +15V/ -15V/ -13V/ -13V /5V				sendtel	6.9 us
19	378	Read sine	Read sine 120	encpos			13 us
20	399	/	/	telmetry	Reset scheduler counter		19us max

Table 1: Global scheduling of MCU DSP Program under timer interrupts.

4 Commanding

The command is send to DSP via FPGA and comprises a mnemonic field + a 16 bit parameter filed
When reading the command the parameter is put in a Parameter Table locate in DSP.
All Parameter table address are not used. Consequently the unknown commands are tagged with the bit 28 set to 1 in the variable located in the Parameter Table. The initialisation is done in the initram.asm subprogram.
The first operation is a set of all Parameter Table variables of 0X10000000, setting the bit 28 to 1.

- The program which reads the received command from communication FPGA 32 bits register is: cmdread.asm
- The program which interprets the received command, put the parameter value in the parameter table and reply to communication FPGA is: cmdint.asm

4.1 Command mnemonic constant define list and init of default values

- The list on mnemonics which can be recognised by command interpreter is listed in:cmd.h
- The Parameter values are initialised with default values during an initialisation process done just after the second boot. The initialisation program is listed in: initram.asm

5 Telemetry

At the end of the timer scheduling of 420us, the program telmetry.asm compute the programmed telemetry packets, putting the telemetry data in an internal MCU Ram buffer.

Then, every 4 21 us timer interrupt, the DSP provides the bufferised data to the communication FPGA by mean of the program sendtel.asm.

At the end of the packet, the bit 31 is set to 1. In sendtel call, the flag 2 is set to 1 to indicate to communication FPGA that this is the end of the packet from DSP and start the additionnal words (time stamp and CRC) added in embedded code in communication FPGA .

6 SMEC Trajectory and profile computation

To view source code file, click on smectraj.asm

The trajectory is computed in the smectraj subprogram which is called once every cycle of the main scheduler. In scan mode, the ramp is computed by addition of an increment related to the speed reference parameter (0.1um/s unit). On the basis of a cycle of 420 µs this lead to an increment of:

Inc(um)= scheduler_cycle(s)*speed = 420^{e}-6 (s)*0.1(ums-1) = 4.2^{e}-5.

For a speed of 500um/s the speed parameter to send is 0x1388 (=5000 * 0.1um/s).

6.1 Pseudocode

Load trajectory mode (STRAJMODE) from PT
Case Trajectory Mode is:

0: jump end of subprogram: steady position
1: call step trajectory and then jump end of subprogram
2: read scan number from PT: if not zero call compute trajectory
3: automatic trajectory decrease for init with detection of servo error (to be cancelled)
4: force the encoder count to be equal to trajectory value. Allow to init the encoder count.

6.1.1 Update of smec status word about scanning

The smec status word is updated in smectraj.asm modulus when scanning is up or down as well as the number of remaining scans encoded on 12 bits.

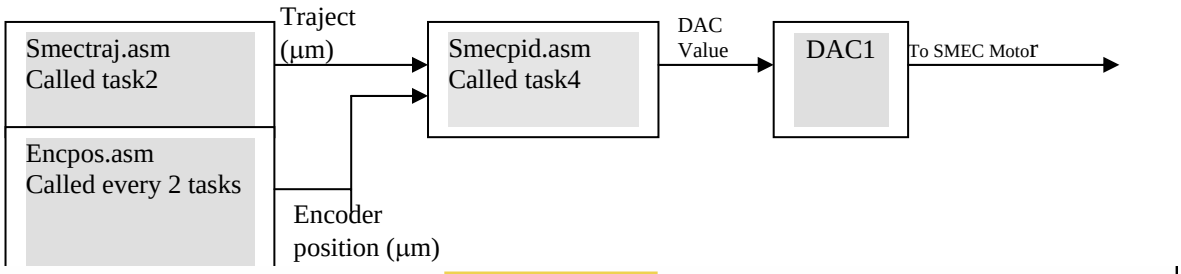
6.1.1.1 Computing scan number in smec status word

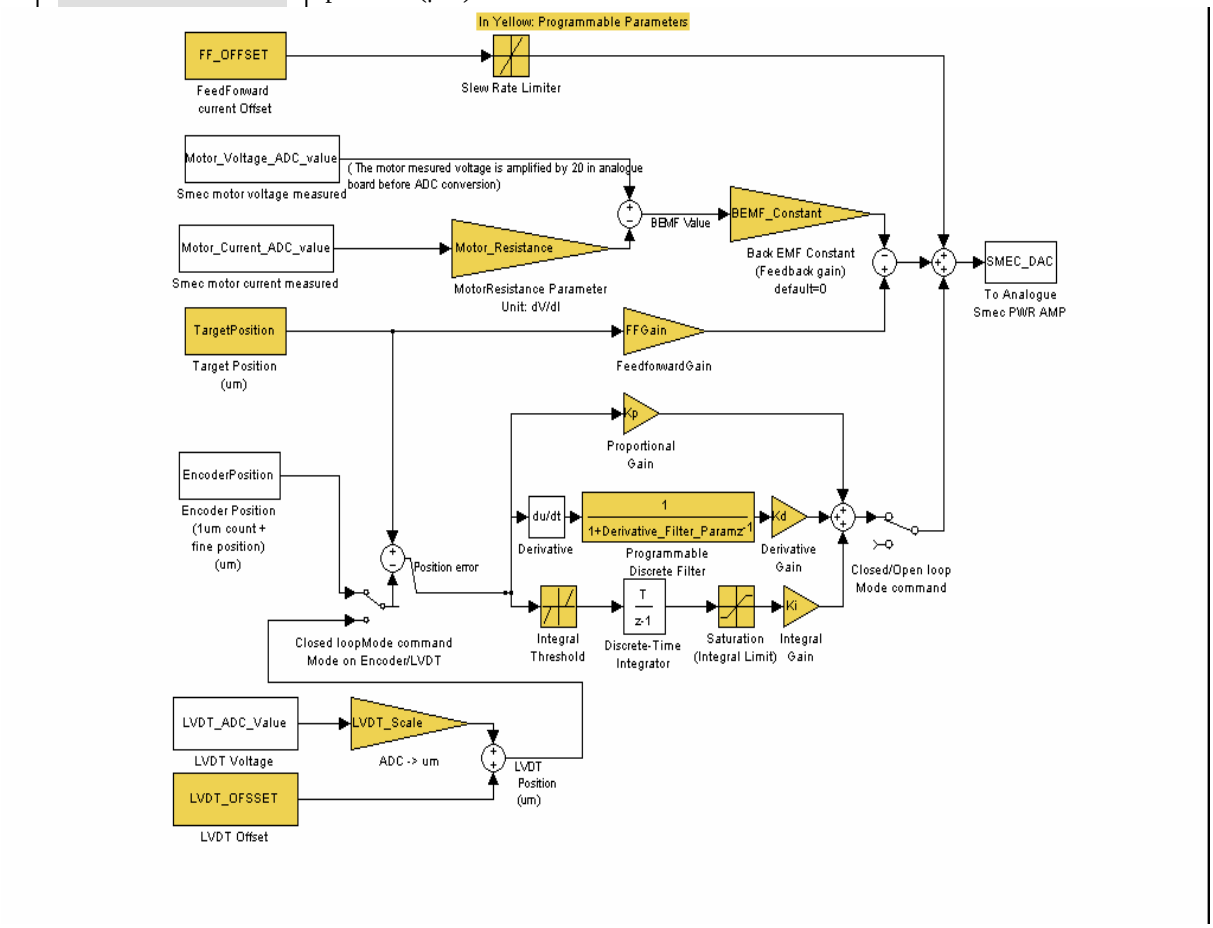
The 12 bit number of scan is computed from the 16 bit word this way.

Read the 16 bit ‘number of scan’ parameter and put in r0
r2= 0xFFF; /* prepare a 12 bit mask */
r3= r0 and r2; /* mask the only 12 LSB of the parameter */
r1= SMEC_STATUS word;
r2= 0xF; /* reset bit field in status word related to scan number */
r1= r1 or fdep r3 by 4:12; /* field deposit of scan number 12 bit in status word*/
SMEC_STATUS=r1;

7 SME PID Controller

To view source code file, click on smecpid.asm





7.1 Pseudo code

- Read encoder fine position in nm
- Convert encoder position in um (float)
- Compute speed by S(t)=A(E(t)-E(t-1))+BS(t-1)
- Save speed in Parameter Table for HK purposeT
- Read trajectory expressed in um (float)
- Make difference trajectory-encoder position and save servo error in variable
- Multiply by 100 and save servo error (10nm unit) in PT
- Read the lvdt signal
- Substract 0x8000 to get a signed value in lvdt_dc_value
- Update SMEC_STATUS in PT depending on lvdt sign.
- Detect when lvdt is crossing level 0 and save encoder_lvdt_diff value in PT
- Compute lvdt_position (um)=0.0002*LVDT_SCALE *lvdt_dc_value+ LVDT OFFSET and save LVDT_POSITION in PT
- Case loop_mode is :

0 : set trajectory mode to 0 (trajectory steady)
clear error in smec_status.
Jump to end_pid (end of program)
1 : on transcient of this mode : force encoder counter to be equal to trajectory value.
If servo error > (1mm) call safety
Perform closed_loop
2 and 3 Cancelled : jump end_pid
4 : (closed loop on LVDT)
read lvdt_position in f1
limit the trajectory value to
LVDT_OFFSET + MAX_TOLERATED_LVDT (about 8000+4000=12000um)
compute servo error = trajectory-lvdt_position
call safety proc if servo error > 1000um.
saturate to 10 um servo error.
Save servo error in PT in 10nm unit
Perform closed_loop
5 : LVDT corrected: cancelled: jump end_pid.
6 : call feedforward
f8=ff effect
jump send to dac
other: jump end_pid

Perform closed_loop :

- Load servo_error in f9
- Compute kpeffect= Kp*0.0002*servo_error
- Compute filtered derivative effect : S(t)=0.003*E(t)-0.003*E(t-1)+0.905..*S(t-1)

- Compute integral of servo error with threshold and saturation :
  Read servo error
  Compute abs value of servo
  Read Intgration threshold parameter
  If abs servo error is greater pid__i_in =0
  Else pid_i_in=servo error
  Load pid_i_in sample t-1
  Add pid_i_in and pid_i_in t-1
  Multiply by 0.0002 (0.5*400us)
  Saturate integral effect :

Compute integral command by 0.000001*SKI*integral of servo error.

- Sum up the 3 effects (Kp, Kd and Ki)
- Call feedforward: perform FFGAIN*traject +FFOFFSET + Backemf (slew rate applied on dac)
- Add ffeffect (feedforward effect) with previous PID command value
- Limit dac command to +/-1.0
- Perform DAC command= 32767*daccommand +32768.

7.2 Speed calculation

The speed calculation is used for telemetry and for check of the maximum tolerated speed which depends on encoder signals minimum acquisition delta time (otherwise pulse counting may loss some 0 crossing detection).
The speed is calculated on the basis of :

Speed = (encoder position(t)-encoder position(t-1))/(cycle time) filtered by a first order filter at 25Hz cut-off

7.3 Slew rate limiter on DAC

The DAC increment is limited to a slew rate limit to avoid high current differences in open loop.
This is a fixed value for mechanism safety.

7.4 LVDT position for control

The lvdt position for the control is computed from the LVDT_DC_ SIGNAL read by the ADC.
The algorithm for LVDT is:

- Read ADC value of LVDT DC every 400us cycle
- Substract 0x8000 to get a 0 centered value
- If LVDT value >0 set 1 in SMEC STATUS word
- Else set 0 in SMEC STATUS word
- Read LVDT scale factor in parameter table
- Multiply with 0.00002 factor
- Multiply with LVDT value to get LVDT position
- Add LVDT Offset value.

8 Encpos: encoder signal normalisation, position counting, and fine position interpolation

(To view the source file, click on encpos.asm)

This subroutine computes the optical encoder sine and sine 120 signals to perform the SMEC position by both crude 1micron counting in addition with a fine interpolated position computation.

The compuations are:

1. read the centered sine and sine120 (or also called cosine) to detect min and max values based on every 2 periods of the sines
2. once min and max detected on 2 periods, compute an updated ENCODER_OFFSET values, so a new mean value is calculated by (MAX+MIN)/2.
3. By the mean time compute the amplitude of the sine and sine 120 which is (MAX-MIN)/2

4. At this point the encoder signals are normalised to 1 in float format
5. Detection of the cases when the zero crossing on sine is detected, (the values change of sign)
6. Detection of the sign of cosine (sine120) to decide if the count to be done is positive or negative.
7. Perform increment or decrement of the counter
8. Read the value of the sine signal, divide by the normalised amplitude to obtain a ratio which is the entry of a 500 Look Up Table made of arcsine precomputed values.
9. Add the arcsin interpolated value (expressed in nm) to the 1 micron step counter to perform the accurate position.
10. Save in Parameter table for HK both counting value (position micron) and fine interpolation value (in nm)

8.1 Readsin: read out of the encoder sine signal

(to view source file, click on: readsin.asm)

This subroutine called at the first scheduler step and t-hen every 2 steps reads the optical encoder sine photodiod signal (the center one, most illuminated). The computation is :

1. convert the 16 bit ADC word into a 0-65535 inter value (2' complement)
2. save this value in the parameter table for HK purpose
3. remove programmable last bits to avoid noise effect on counting
4. center the signal by substration with SINE_OFFSET1_VALUE to obtain a centered signed number to allow zero crossing detection for 1 micron counting.

8.2 Readcos: read out of the sine 120 signal

(To view the source file, click on : readcos.asm)

This subroutine called at the first scheduler step and t-hen every 2 steps reads the optical encoder sine 120 photodiod signal (the side one, less illuminated). The computation is:

1. convert the 16 bit ADC word into a 0-65535 inter value (2' complement)
2. save this value in the parameter ENCODER_SIGNAL2 table for HK purpose
3. remove programmable last bits to avoid noise effect on counting
4. center the signal by substration with SINE_OFFSET2_VALUE to obtain a centered signed number to allow normalisation for counting direction (up or down).

9 Launch Latch control

The launch latch control is performed once every cycle of the scheduler by the latch subprogram.
The algorithm relates to the latch.asm assembly file. The principle is:

1. to perform a polling of a latch command in the parameter table
2. to set the engage/disengage relay depending on a latch or unlatch command value by mean of the dedicated digital output bit
3. to trig the voltage pulse to the latch solenoid after 100ms (wait for the relay transcient) by mean of the dedicated digital output bit.
4. To wait for a total duration of 200ms to be sure that the analogue voltage pulse is performed in SMEC board
5. To set the SMEC status word bit to latch or unlatched
6. To reset the command in parameter table to 0 to permit a new latch command.

The detailed algorithm is:

Read ‘latch state’ variable value
Case ‘latch state’:

0 : send 0 to digital output LATCH COMMAND (this digital output relates to the trigger of the voltage pulse to solenoid)
Set the pulse counter to 500 (500 * 400 us = 200ms)
Read the latch command value in the parameter table
If latch command == 1 then set ‘latch state’ = 1
If latch command ==2 then set ‘latch state’ =2

1: Reset latch command value to 0 in parameter table (the latch command is an impulse)
Decrement pulse counter of 1
Set to 1 the digital output of LATCH ENGAGE (set the relay to engage position)
If pulse counter is <= 250 Set to 1 the digital output of LATCH COMMAND
If pulse counter is <= 0 then set ‘latch state’ to 0

2: Reset latch command value to 0 in parameter table (the latch command is an impulse)
Decrement pulse counter of 1
Set to 1 the digital output of LATCH DISENGAGE (set the relay to disengage position)
If pulse counter is <= 250 Set to 1 the digital output of LATCH COMMAND
If pulse counter is <= 0 then set ‘latch state’ to 0

10 BSM Control

The BSM control uses a voice coil motor actuator and a magneto resistive sensor. The software modulus dedicated to BSM control are:

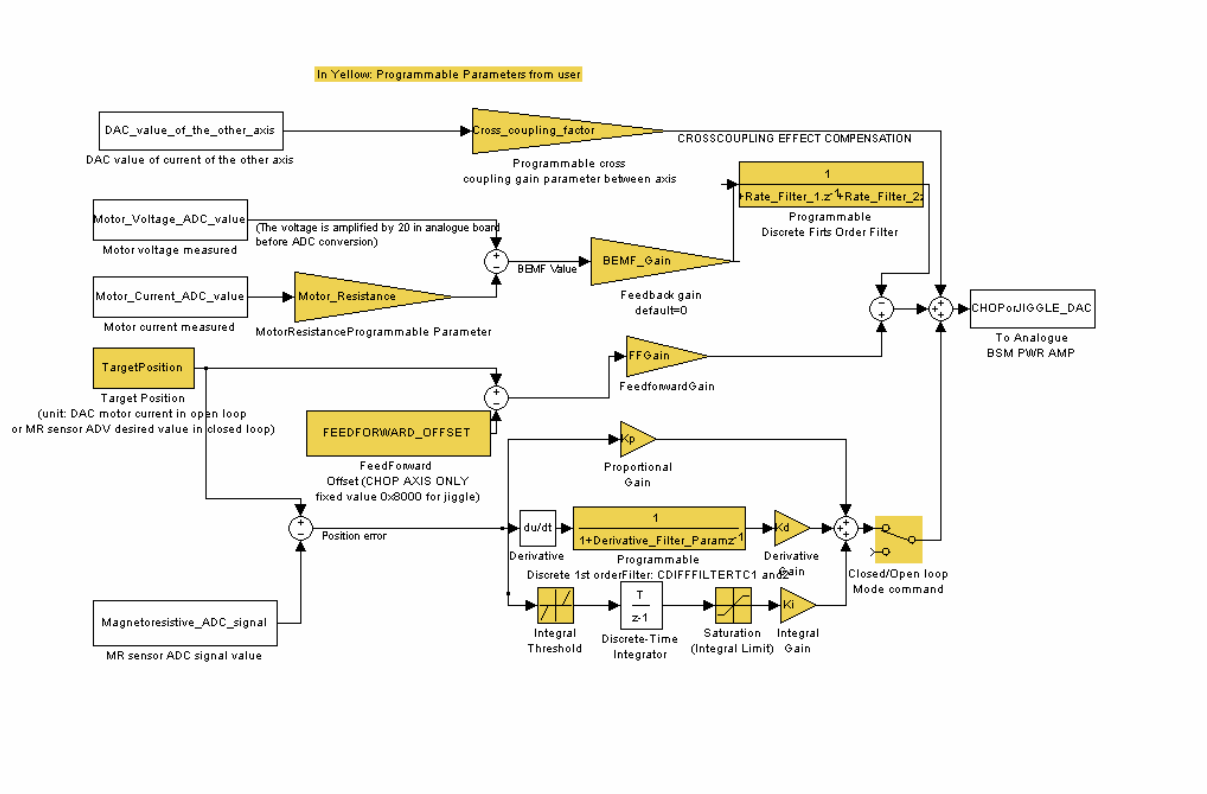
- Bsmmove.asm: step generation with programmable slew rate for both chopper and jiggle axis
- Choppid.asm: chopper open and closed loop (PID) computation
- Jigpid.asm: same function as Choppid for the jiggle axis.

10.1 BSM Trajectory reference computation

To see source code file, click on: bsmmove.asm

10.2 Choppid and Jigpid

The control algorithm is a classical PID with filtered derivative in addition with open loop ffedforward control, back emf damping and crosscoupling compensation, asdescribed in the following simulink diagram:



11 Annex: Scheduler Timing measurements

11.1 TMZHI

The timer counter generating the interrupt starts at 1A4 (420) ie 21us (each instruction lasts 50ns)
Tmzi task 8 the counter as simulated with 21020 simulator is 17D (381)
Task 20 : the timer is 149 (329) ie the tmzhi takes 91 cycles to reach task 20

11.2 Task 4

(command to get value 0x99F10000)
This task contains **smecpid**. The duration depends on the open/closed loop programmed mode

Mode	Measured by get task in real time	Value using the 21020 Simulator
Open loop	0x6E= 110	0x5B= 91
Closed loop	0x13=19	0x1a4= 190

NB: in case of task duration > 1A4 the timer cannot be interrupted by itself so the next timer interrupt is served the next decount zero crossing.

11.3 Task 20

To get the measured task 20 duration the command is :

Get task20timer (9A010000)

It is the task when telemetry is computed. The timer is 12A(298) when entering telemetry.

After end of telemetry and before the interrupt is finished it remains 13 instructions.
So the max telemetry length is 298-13=285 instructions.

Conditions	**Timer value at the end of task**
Open loops telemetry stopped	254
Open loops telemetry started no frame (all sampling to 0)	193
Open loops telemtry smec on at 5	123/186
Idem + BSM at sampling10	109/116/179
Idem + Engineering at sampling 10	30/102/109/172
Idem – smec which is stopped	37/179
Idem – bsm stopped as well	***18/75/147/204***

----------End of Document----------