*SOFT*DMC

FPGA based

Digital Motion Controller

2.4 - SW Rev. 2.3+ / HW Rev. 23+

This page intentionally not blank

Table of Contents

Table of Contents

Table of Contents

*SOFT*DMC

SAFETY

WARNING

Servo motors are capable of inflicting serious injury both to people and mechanisms associated with the servo system. In addition, some motors use potentially lethal supply voltages.

When a servo system is first configured, unpredictable behavior should be EXPECTED. First time checks of basic servo operation (such as motor position versus drive) should be checked with the motor power leads disconnected.

NEVER depend on software commands to disable a motor when you or others would be exposed to a hazard should the motor start unexpectedly. Motor power should be always be removed when working on mechanical parts of the servo system.

Be especially careful with encoder wiring, as a simple bad connection of one encoder wire can lead to loss of control and a runaway servo system.

*SOFT*DMC

GENERAL

The *SOFT*DMC digital motion controller is a FPGA based multi-axis DC servo motor controller intended for embedding in Xilinx SpartanII,SpartanIIE and Virtex FPGAs. The 4 axis motion controller will fit in a 100K gate SpartanII or SpartanIIE part and the 8 axis controller will fit in a 200k gate SpartanII or SpartanIIE.

All logic, CPU, RAM and program ROM reside in a single FPGA chip making for an extremely flexible, powerful, and very low cost motion control solution. Custom variants of the *SOFT*DMC design can be easily created for specific applications.

The *SOFT*DMC design has an embedded ~50-100 MIPS 16 bit DSP coupled with special hardware for motion control. Each axis has dual quadrature and index inputs, PWM, direction and enable outputs. Up to 72 general purpose I/O bits are also available for limit switches, status outputs, absolute encoder inputs, and other uses.

Position, velocity and acceleration parameters are all 32 bit. Dual encoders per axis permit dual feedback (position/velocity). 32 bit gearing between axis is provided for precise ratioed multi-axis moves. A 1024 word trace buffer is available in some configurations.

The fully synchronous design of the host interface allows almost any parameter to be changed during motion. 16 bit PC/104, 8 bit microcontroller, PCI, serial, and other host interface types are available.

The PID loop has the normal proportional, integral, integral limit, and derivative terms, plus velocity, acceleration, bias, and friction feed forward terms to extract the maximum performance from the mechanics. High sample rates (>30KHz for 4 axis simultaneous motion, 66Mhz clock) support small and fast drive systems.

Programmable event/breakpoint logic allows real time response to internal (position, time,velocity, flags, etc) and external (limit switches, sensors, etc) events. Breakpoint and event logic allow fully buffered profiling operations and filter changes based on breakpoints or external events.

HOST INTERFACE

GENERAL INTERFACE SPECIFICATIONS

GENERAL

The *SOFT*DMC has several host interface types available, but all interface types share as single register map with seven 16 bit registers starting at the BASE address +2. (Note that that all addresses are *byte* addresss) All host interaction with the SOFTDMC is done via these registers. The serially interfaced *SOFT*DMC configurations are similar but are accessed via an ASCII command set with hexadecimal parameters.

BUS INTERFACE DESCRIPTION

COMMAND REGISTER

The register at base address +2 is the command register. The command register is used in conjunction with the data register for reading and writing internal motion controller parameters. The bit definition for command register writes are as follows:

COMMAND REGISTER WRITES @BASE + 2

W	A3	A2	A1	A0	L	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

W (MSB) is the write bit. If this bit is ‘1', a write operation will be performed. If it is a ‘0', a read operation will be performed. The A0 through A3 bits specify which axis will be selected for the parameter read or write. The L bit specifies if the parameter is a 32 or 16 bit parameter, a ‘1' bit specifies a 32 bit parameter. The P0 through P9 bits specify the parameter that will be written or read.

On reads, Bit 15 of the command register is the busy bit, indicating that a command is in progress, and bit 14 is the sync bit, used to synchronize the host with the internal sample rate. The other bits (S) are general purpose status bits that can be used by the firmware or event logic to display internal system status.

COMMAND REGISTER READS @BASE + 2

B	SY	S	S	S	S	S	S	S	S	S	S	S	S	S	S

HOST INTERFACE

GENERAL

DATA REGISTERS

Motion control parameters are transferred to and from the motion controller via the data registers. The registers are located at base address +4 and base address +6. There are two data registers to allow 32 bit parameter reads and writes with a single command cycle.

DATA REGISTER LOW WORD @ BASE+4

L	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**	**L**

DATA REGISTER HIGH WORD @ BASE+6

H	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**	**H**

WRITING A PARAMETER

The process of writing a parameter is as follows: First write the parameter data to the data registers. When 16 bit parameters are written, only the low word of the parameter register need be written, for 32 bit parameters, both high and low data registers need to be written. Next the command register is written with the desired parameter number, word/long bit, axis and write flag (1 bit in MSB). When the command is written, the busy bit will be set and will stay set until the command is completed. Commands and data must not be written when the busy bit is high.

READING A PARAMETER

The process of reading a parameter is as follows: First the command register is written with the desired parameter number, axis, and word/long bit. Once the command is written the busy bit will be set and will stay set until the requested data is available in the data registers. Once the busy bit returns to 0 the data can be read from the data register low word (16 bit parameters) or data register low and high words (32 bit parameters). Commands must not be written when the busy bit is high. Data should not be read from the data registers when the busy bit is high.

HOST INTERFACE

GENERAL

BUSY TIME

The internal DSP handles host interface requests on a synchronous polled basis, that is it does all the motion control operations for *all* axis, then runs a host interface loop until the next sample time. This mode of operation has the advantage that almost any motion related parameter can be accessed and changed during motion.

One side effect of this mode of operation is that the host interface can stay busy for the time it takes to process all axis and that the host interface bandwidth will decrease with increasing sample rates. For example, with a 50 MHz CPU, a sample rate of 10 KHz, and 4 axis active, approximately 64 uSec would be available for the host interface, and 36 uSec would be used doing motion calculations.

If a host interface cycle starts just after a sample period has begun, the interface busy bit will stay high for the maximum period. There is always a minimum of one host interface cycle done per sample period, so that if you set the sample rate faster than the DSP can process all axis, the host interface will still function.

GETTING IN SYNC

For some special applications, it may be desirable to synchronize the host with the internal *SOFT*DMC sample rate. The SYNC bit (Bit 14) in the status register can be used for this purpose. The SYNC bit is set whenever a host interface cycle is done, and cleared when all axis calculation are complete (the point at which the *SOFT*DMC firmware falls into the host interface loop). For example, to synchronize a host interface such that the host reads one parameter per sample period, you would create a loop that 1. reads the desired parameter (which will set the sync bit), and then waits for the sync bit to be cleared, then repeats.

HOST INTERFACE

GENERAL

FIRMWARE DOWNLOAD

The 16 bit DSP in the *SOFT*DMC can have its firmware downloaded from the host if desired. This will overwrite the standard *SOFT*DMC firmware that is part of the FPGA configuration. Three registers are involved in program downloading: the program address register, the program data register, and the processor reset register.

PROGRAM ADDRESS REGISTER @ BASE+8

X	**X**	**X**	**X**	**X**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

PROGRAM DATA REGISTER @ BASE+10

D	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**

DSP RESET REGISTER @ BASE+14

X	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**R**

DOWNLOAD PROCESS

The process for downloading new DSP firmware is as follows: First the DSP is reset by writing a 1 to the DSP reset register. Then all the words of DSP program are written by first writing the target address for the word to the program address register, then writing the program data for that address to the program data register. This is repeated for all the words of the program firmware. When all the program words have been written, DSP reset is removed by writing a 0 to the DSP reset register, starting execution of the new code.

SERIAL INTERFACE DESCRIPTION

OPERATION

PARAMETERS

GENERAL

The *SOFT*DMC motion controller has a large number of parameters that control its operation. Some of these parameters are global but most are duplicated for each axis. The file INCLUDE.INC and BITS.INC supplied with the *SOFT*DMC configuration have the specific parameter addresses and types. Parameter addresses are not referred to in this document as they may change from firmware revision to revision. This list is not complete as there are many more parameters that are used internally or for special purposes The INCLUDE.INC file lists all parameters.

MAKEINC

The supplied utility program MAKEINC will translate the INCLUDE.INC and BITS.INC to files to include files of various sorts. Assembly, batch, C, and Pascal include files can be created. Invoking MAKEINC with no parameters will print usage information. Examples of MAKEINC usage:

MAKEINC INCLUDE.INC SOFTDMC1.H C

(Create C include file of parameter addresses)

MAKEINC INCLUDE.INC MOTPARMS.PAS P S Loc

(Create Pascal include file of parameter addresses, all with appended Loc string)

MAKEINC INCLUDE.INC MOTPARMS.ASM A M

(Create assembly language include file of parameter addresses, merging parameter type with name)

PARAMETER TYPES

There are six different types externally useable parameters:

FLAG: Flags are a 16 bit parameter, a value of 0xFFFF is “true” and a value of 0 is false.

PTR: 11 bit address pointer range 0 to 2047

INT: 16 bit signed number range -32768 to 32767

UINT: 16 bit unsigned number range 0 to 65535

LONG: 32 bit signed number range -2147483648 to 2147483647

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
OPERATION FLAGS			
GO	FLAG	AXIS	Start profile when set true, cleared when done
MOTION	FLAG	AXIS	True when in motion (velocity <> 0)
PID	FLAG	AXIS	Enable PID portion of control loop when true
PROFILE	FLAG	AXIS	Enable profile generator when true
EXTPROFILE	FLAG	AXIS	Disables velocity follower when true, allowing host control of DDA.
HOME	FLAG	AXIS	True when home operation is complete
DIRINV	FLAG	AXIS	Invert direction output polarity if true
ERROR	FLAG	AXIS	True when excessive position error has occurred
SLEW	FLAG	AXIS	True when slew portion of profile has been reached
SWAPMOT	FLAG	AXIS	Set true to do motion register set swap, cleared when swap is done
SWAPFIL	FLAG	AXIS	Set true to do PID filter register set swap, cleared when swap is done
RESET	FLAG	AXIS	Do hardware reset of Axis if cleared

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MISC MONITORING AND CONTROL FUNCTIONS			
LED	PTR	GLOBAL	Specify which parameter the debug LEDs follow
LEDAXIS	PTR	GLOBAL	Specify which axis the LEDs parameter is read from
BEEPER	PTR	GLOBAL	Specify which parameter the beeper is driven by
ACTVEL	INT	AXIS	Actual velocity (counts/sample)
MAXPWM	UINT	AXIS	Maximum PWM drive applied
MAXNEGERR	INT	AXIS	Maximum negative deviation from profile. (counts)
MAXPOSERR	INT	AXIS	Maximum positive deviation from profile (counts)
EXERR	INT	AXIS	Excessive position error limit (counts)
EVENTS	UINT	AXIS	Number of events (max number depends on free memory)
FIXUP	LONG	AXIS	Difference between profile end position and NEXTPOS (counts)
ERROR	LONG	AXIS	Position error (counts)

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
PID LOOP TUNING PARAMETERS (Working block)			
KK	UINT	AXIS	PWM offset or bias
KP	UINT	AXIS	Proportional constant
KD	UINT	AXIS	Derivative constant
KA	UINT	AXIS	Not used
KI	LONG	AXIS	Integral constant
KIL	UINT	AXIS	Integral limit
KF1	UINT	AXIS	Velocity feed forward term
KF2	UINT	AXIS	Acceleration feed forward term
KF3	UINT	AXIS	Not used
KFF	UINT	AXIS	Friction feed forward term
KDFIL	UINT	AXIS	Derivative term filter coefficient
POSENC	PTR	AXIS	PID loops pointer to position encoder (default = ENCP)
VELENC	PTR	AXIS	PID loops pointer to position encoder used for velocity calculation (default = ENCP)
NEXTFILBLOCK	PTR	AXIS	Pointer to next filter block

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MOTION CONTROL PARAMETERS (Working Block)			
NEXTPOS	LONG	AXIS	Next position for move
SLEWLIMIT	LONG	AXIS	Slew speed during move (counts/2^24/sample)
DESVEL	LONG	AXIS	Commanded velocity for velocity or profile mode (counts/2^24/sample)
ACCEL	LONG	AXIS	Acceleration value (counts^2/2^24/sample)
DELA	LONG	AXIS	Delta acceleration for cubic profiles
BREAKPOINT	LONG	AXIS	Local breakpoint value
BREAKA	PTR	AXIS	Normally points to value compared to BREAKPOINT
BREAKB	PTR	AXIS	Points to breakpoint value (normally BREAKPOINT)
NEXTMOTBLOCK	PTR	AXIS	Pointer to next motion control block
USER	UINT	AXIS	Free user variable or pointer
USER PARAMETERS			
ENCP	LONG	AXIS	Primary encoder position (counts)
ENCS	LONG	AXIS	Secondary encoder position (counts)
FOLLOW	PTR	AXIS	Pointer to position for PID loop to follow (default = DESPOS)
GEARP	LONG	AXIS	Primary gear ratio (8.24). If 0, gearing is disabled (1X used)
GEARS	LONG	AXIS	Secondary gear ratio(8.24) If 0, gearing is disabled (1X used)

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
USER PARAMETERS			
GEARPOFFSET	LONG	AXIS	Primary gear offset
GEARSOFFSET	LONG	AXIS	Secondary gear offset
GEARPPARAM	PTR	AXIS	Pointer to primary geared parameter
GEARSPARAM	PTR	AXIS	Pointer to secondary geared parameter
DESVEL	LONG	AXIS	Current profile generated velocity (Counts/2^24/sample)
HOMEPOSP	LONG	AXIS	Primary encoder starting position count (loaded when index detected)
HOMEPOSS	LONG	AXIS	Secondary encoder starting position count (loaded when index detected)
PHASEK	LONG	AXIS	Phase constant for user timer (PHASEA=(PHASEK*sample)
PHASEA	LONG	AXIS	User phase accumulator timer
COPYSRCL	PTR	AXIS	Long parameter copy source pointer
COPYDSTL	PTR	AXIS	Long parameter copy destination pointer
COPYSRCU	PTR	AXIS	Short parameter copy source pointer
COPYDSTU	PTR	AXIS	Short parameter copy destination pointer

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MISCELLANEOUS GLOBAL PARAMETERS			
GEAREDP0	LONG	GLOBAL	Primary geared parameter, axis 0
GEAREDS0	LONG	GLOBAL	Secondary geared parameter, axis 0
GEAREDP1	LONG	GLOBAL	Primary geared parameter, axis 1
GEAREDS1	LONG	GLOBAL	Secondary geared parameter, axis 1
GEAREDP2	LONG	GLOBAL	Primary geared parameter, axis 2
GEAREDS2	LONG	GLOBAL	Secondary geared parameter, axis 2
GEAREDP3	LONG	GLOBAL	Primary geared parameter, axis 3
GEAREDS3	LONG	GLOBAL	Secondary geared parameter, axis 3
GEAREDP4	LONG	GLOBAL	Primary geared parameter, axis 4
GEAREDS4	LONG	GLOBAL	Secondary geared parameter, axis 4
GEAREDP5	LONG	GLOBAL	Primary geared parameter, axis 5
GEAREDS5	LONG	GLOBAL	Secondary geared parameter, axis 5
GEAREDP6	LONG	GLOBAL	Primary geared parameter, axis 6
GEAREDS6	LONG	GLOBAL	Secondary geared parameter, axis 6
GEAREDP7	LONG	GLOBAL	Primary geared parameter, axis 7
GEAREDS7	LONG	GLOBAL	Secondary geared parameter, axis 7
PROCTIMER	UINT	GLOBAL	Process timer, reads SYSCLK*2 cycles for processing all enabled axis

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MISCELLANEOUS GLOBAL PARAMETERS			
TIMEOUT	UINT	GLOBAL	Count of DSP out of time events
SWREVISION	UINT	GLOBAL	Firmware revision number
			Major rev. = MSB, Minor rev. = LSB
GPHASEK	LONG	GLOBAL	Global phase accumulator constant
GPHASEA	LONG	GLOBAL	Global phase accumulator
AXIS EVENT AND BLOCK RAM (16 words per block/8 per event)			
BLOCK9/EVENT1,EVENT2		AXIS	RAM block 9, event 1 and event 2
BLOCK8/EVENT3,EVENT4		AXIS	RAM block 8, event 3 and event 4
BLOCK7/EVENT5,EVENT6		AXIS	RAM block 7, event 5 and event 6
BLOCK6/EVENT7,EVENT8		AXIS	RAM block 6, event 7 and event 8
BLOCK5/EVENT9,EVENT10		AXIS	RAM block 5, event 9 and event 10
BLOCK4/EVENT11,EVENT12		AXIS	RAM block 4, event 11 and event 12
BLOCK3/EVENT13,EVENT14		AXIS	RAM block 3, event 13 and event 14
BLOCK2/EVENT15,EVENT16		AXIS	RAM block 2, event 15 and event 16
BLOCK1/EVENT17,EVENT18		AXIS	RAM block 1, event 17 and event 18
GLOBAL EVENT/MISC RAM			
GEVENT1 through GEVENT12		GLOBAL	Event or general purpose RAM

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
INTERNAL HARDWARE			
CNTCTL	UINT	AXIS	Sets operational mode of quadrature input counter
PRESCALE	UINT	GLOBAL	Sample rate prescale digital oscillator - sets PWM rate as: (SYSCLK/256)*(PRESCALE/65536)
POSTSCALE	UINT	GLOBAL	Byte sample rate postscale divider - sets sample rate as PWM/POSTSCALE
PWMGEN	UINT	AXIS	Byte (in MSB of word) PWM value normally driven by PID loop, but can be host controlled if the PID loop is disabled
DIR	UINT	AXIS	Single bit in MSB controls direction output bit
ENA	UINT	AXIS	Controls Hbridge/Servo Amp enable bit
IRQREG	UINT	GLOBAL	Controls IRQ channel and mask
SETIRQ	FLAG	GLOBAL	Writes here set IRQ
CLRIRQ	FLAG	GLOBAL	Writes here clear IRQ
PUSH	UINT	GLOBAL	Writes push data on Trace buffer/FIFO
POP	UINT	GLOBAL	Reads read trace buffer data. Writes advance read pointer.
FIFOMODE	UINT	GLOBAL	FIFO mode 1= FIFO, 0= Circular buffer
FIFOSTAT-CLR	UINT	GLOBAL	Reads read data count, writes clear FIFO

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
USER I/O PORTS			
PORTA	UINT	GLOBAL	I/O port A data register
PORTADDR	UINT	GLOBAL	I/O port A data direction register
PORTB	UINT	GLOBAL	I/O port B data register
PORTBDDR	UINT	GLOBAL	I/O port B data direction register
PORTC	UINT	GLOBAL	I/O port C data register
PORTCDDR	UINT	GLOBAL	I/O port C data direction register
PORTD	UINT	GLOBAL	I/O port D data register
PORTDDDR	UINT	GLOBAL	I/O port D data direction register
PORTE	UINT	GLOBAL	I/O port E data register
PORTEDDR	UINT	GLOBAL	I/O port E data direction register
PORTF	UINT	GLOBAL	I/O port F data register
PORTFDDR	UINT	GLOBAL	I/O port F data direction register
HARDWARE INFO			
NAXIS	UINT	GLOBAL	Number of Axis for this hardware
HWREVISION	UINT	GLOBAL	Hardware revision number
HWTYPE	UINT	GLOBAL	Hardware type number (4I34/7I60/5I20/CUSTOM etc)

OPERATION

INTERNAL HARDWARE

GENERAL

Most of the internal hardware in the *SOFT*DMC is for use by the DSP and the user need not be concerned with its operation. There are however a few I/O devices that are appropriate for the user to access directly: The counter control register, the interrupt setup register and the I/O ports.

COUNTER CONTROL REGISTER

There are two quadrature counters available per axis (for the primary and secondary encoders) Each counter has an associated counter control register. The counter control register is an 8 bit register in the LS byte of the word:

U/D	FILTER	CLRD	COI	IDXPOL	CL/IDX	B	A

Counter control register bits are defined as follows:

B7	U/D	R/W — Up/Down mode if set (1x mode) quadrature (4X) mode if clear
B6	FILTER	R/W — Enables ~3 MHz digital low pass filter on A,B, Index inputs if set.
B5	CLRD	R/W — If read as 1, indicates counter has been cleared, if written as 0, clears flag
B4	COI	R/W — Clear On Index, if set, counter will be cleared by index
B3	IDXPOL	R/W — Sets polarity of index input - High = active high index Low = active low index
B2	CL/IDX	R/W — Clear counter if set on writes, read back index input status
B1	B	R/O — Reads back B input
B0	A	R/O — Reads back A input

OPERATION

INTERNAL HARDWARE

COUNT MODE

The encoder counters can operate in 2 different modes: Quadrature mode and up/down mode. Quadrature mode(the default) is selected when the U/D bit in the counter control register is a zero, Up/down mode is selected when the U/D bit is a one.

When used in quadrature mode, the counter will count on every edge of the A and B inputs. This is sometimes called the 4X mode, since a X line encoder will generate 4X counts per revolution in this mode. *This is the suggested mode of operation for most motion control applications since it quadruples the encoder resolution, is more resistant to false counts, and will result in higher performance.*

When used in the up/down mode, a count is generated by the rising edge of the A input. This is sometimes called in a 1X mode (a 500 line encoder will generate 500 counts per revolution in up/down mode) When the UP/DOWN mode is selected, the A input becomes the count input and the B input becomes the count direction, When B is high the count direction is up.

The secondary encoder can be used in UP/DOWN mode to emulate a stepper/indexer combination. To do this the PID loops FOLLOW pointer is set to point to the secondary encoder. Now the A, and B inputs of the secondary encoder become step and direction inputs to the emulated stepper. Each step will index the motor one encoder count of the primary encoder.

INPUT FILTER

The encoder counters have an optional digital input filter that reduces susceptibility to noise spikes on the encoder lines. The filter is enabled by setting the FILTER bit in the counter control register. When the filter is enabled, the maximum count rate is limited to ~3 MHz. *It is suggested that the filter always be enabled unless count rates faster than 3 MHz need to be tracked.*

TRACE BUFFER/FIFO

Some *SOFT*DMC implementations include a 1024 word trace buffer/FIFO. This is useful for gathering statistics in real time for later playback. The FIFO has 4 I/O ports, PUSH,POP,FIFOMODE and FIFOSTAT/CLR. Writes to the push location “push” data onto the FIFO. Reads from the POP location fetch data from the FIFO. Writes to the POP location advance the POP pointer. Writes to the FIFOSTAT/CLR location clear the FIFO, and reads from the FIFOSTAT/CLR location return the count of items in the FIFO.

The FIFO can operate in 2 different modes depending on the setting of the FIFOMODE bit. When FIFOMODE is high, the FIFO mode is used. In FIFO mode, once the FIFO becomes full, additional writes to PUSH are ignored. When FIFOMODE is low the circular buffer mode is used. In circular buffer mode, writes to PUSH when the FIFO is full push the new data and delete (POP) the oldest data.

OPERATION

INTERNAL HARDWARE

INTERRUPT CONTROL REGISTER

The PC/104 and PCI versions of the *SOFT*DMC have an interrupt control register to specify which interrupt is generated by writing to the SETIREQ register. The Interrupt control register is an 8 bit register in the LS byte of the word:

IRQ	IMASK	XX	IDRVEN	ISEL3	ISEL2	ISEL1	ISEL0

Interrupt control register bits are defined as follows:

B7	IRQ	R/O	Interrupt request status
B6	IMASK	R/W	Interrupt mask - high to enable interrupt
B5	XX	Not used	
B4	IDRVEN	Tri-State IRQ drive enable (high to enable) (PC/104 only)	
B3--B0	ISELX	Interrupt select bits (PC/104 only)	

In addition to the Interrupt control register, there are 2 other registers associated with interrupts, the SETIRQ register and the CLRIRQ register. These are simple 1 bit registers that set or clear the interrupt request when written to. Write data is don't-care.

On the microcontroller version, SETIRQ and CLRIRQ set and clear the ATT line on the microcontroller interface.

I/O PORTS

A number of general purpose I/O ports are available for any use. These can be read and written by the host for simple polled operation, or driven by the DSPs event logic for real time response to and control of external events. All I/O ports are 12 bits wide, in the LS portion of the word (bits 0 through 11). Each port has an associated data direction register (DDR). A 0 in a bit position of the DDR means that the corresponding bit in the I/O port is an input. A 1 in a bit positions in the DDR means that the corresponding bit in the I/O port is an ouput. At reset, the DDR is cleared, so the default port direction is all-bits-in.

When a bit is configured as an output, reads of that bit will return the real time status of the I/O pin, not the latched output data. If a high capacitance load is being driven, an immediate readback after an output may not reflect the latest data written to that bit. This means that care must be taken if multiple events in the same Axis do read-modify-write operations to the same I/O port location.

OPERATION

MOTION UNITS

POSITION UNITS

Position units are simple, they are just a signed 32 bit position numbers. They correspond directly with encoder counts. The encoder counter can be programmed to run in quadrature (4X) mode (default) or up/down mode. In quadrature (or 4X) mode, the counter will change on every input edge, this would, for example, give 2000 counts per revolution with a 500 line encoder. In the up/down mode, a 500 line encoder will give 500 counts per revolution.

VELOCITY AND ACCELERATION UNITS

Dynamic units are a little more complex because they involve the sample period. The DESVEL parameter is signed 32 bit number with units of encoder_counts / (2^24) /sample_period. One way of looking at this is as a 32 bit number with a 24 bit fractional part (8.24). This means that the maximum programmable velocity is ~127 counts per sample period. With a 500 line encoder (in 4X mode = 2000 counts/rev) and a ~10 KHz sample rate this works out to be ~38000 RPM. Minimum velocity would be 1/(2^24) counts per sample period which works out to .000018 RPM (less than 1 revolution per month) at a 10 KHz sample rate, which is also the velocity resolution at a 10 KHz sample rate.

Acceleration units are in encoder_counts/2^24/sample_period^2.

MAXIMUM COUNT RATE

The quadrature counter hardware consists of a 9 bit up/down counter with quadrature input decode logic. At every sample period, each counter are read, the count value added to a 32 bit accumulator and then the 9 bit hardware counter is cleared. This limits the maximum count rate of the encoder inputs to ~255 counts per sample. This is double the fastest programmable velocity, so is very unlikely to occur normally. The count rate works out to be ~2.55 MHz at a 10 KHz sample rate or 6.38 MHz at a 25 Khz sample rate. *Note that the count rate is limited to ~3 MHz if the input filter option is enabled* (see hardware section)

OPERATION

MOTION UNITS

PWM AND SAMPLE RATE

The PID loop and Profile Generator operate at a fixed sample interval determined by the setting of two parameters. PRESCALE and POSTSCALE. PRESCALE sets the rate of a 16 bit phase accumulator. The phase accumulator multiplies the system clock by a factor of PRESCALE/65536. For example, the maximum PRESCALE value of 65535 will result in a multiplication ratio very close to one, while a PRESCALE value of 16384 would result in a ratio of 1/4. The output of the phase accumulator generates the clock for the PWM generator. The phase accumulator is used instead of a programmable divider so that the sample rate may be chosen with high resolution. At normal PWM rates of ~25KHz, the PWM frequency and hence sample rate are selectable to within the accuracy of the system clock crystal oscillator, < .01 %.

The equations for PWM rate and sample rate are as follows:

$$PWMRATE = (SYSCLK/256) * (PRESCALE/65536)$$

$$SAMPLERATE = PWMRATE/POSTSCALE$$

For example, at a system clock frequency of 50 MHz, a PRESCALE value of 8389 would give a PWM value of 25.001 Khz. (50e6/256*8389/65536). With a POSTSCALE value of 2, the sample rate would be 25.001 Khz/2 = 12.5 Khz.

CHOOSING A PWM RATE

PWM rates are normally chosen to be above the audible range (>20 KHz). Lower rates can be used for larger motors and have the advantage of higher efficiency due to lower switching losses in the Hbridge. When the PWM is used with a filter to generate analog signals for standard servo amplifiers (like the 7I33 analog servo interface), the highest PWM rate should be chosen to reduce output ripple.

OPERATION

MOTION UNITS

MAXIMUM SAMPLE RATE

The *SOFT*DMC firmware is capable of running at ~30 KHz sample rate for 4 axis in simultaneous motion and ~15 KHz for 8 axis in simultaneous motion. If the event logic or gearing are used, the maximum sample rate will be decreased by an amount depending on the total number of events / gears used. Time per event is approximately 600 nS. The time used per geared parameter is approximately 500 nS.

If the sample rate is set faster than the DSP can process all the enabled axis, the TIMEOUT count will be incremented. The processing time per loop can be measured via the PROCTIMER parameter. The PROCTIMER parameter is updated every sample and counts the number of system clock/2 counts used by the DSP for processing all the enabled axis. For example, at a system clock of 50 MHz, the PROCTIMER will run at 25 MHz (40 nS/count), so a PROCTIMER value of 700 would be equal to 700*40nS = 28 uSec. *Note that if a timeout event happens, the PROCTIMER parameter will be invalid for that cycle.*

If the sample time is set faster than the DSP can process all the axis, the motion controller will still work, but the sample time will be determined by the (variable) processing time instead the sample rate generator. Since velocity and acceleration values will be variable in this case, you should not normally run the motion controller in this mode.

OPERATION

PROFILE GENERATOR

GENERAL

There are two main parts of the motion controller firmware, the PID loop and the PROFILE generator. The basic job of the profile generator is to provide position information (the *setpoint* position) for the PID loop to track, The PID loop then generates drive and direction signals to control the motor. A profile is a set of positions in the time domain. The profile generator has parameters for acceleration, slew speed and motion endpoints, and one main control flag: GO. The profile generator has two main modes of operation, position mode and velocity mode.

POSITION MODE

In position mode, motion always starts from a stopped condition, in other words the VELOCITY term is 0. To do a move, the desired ACCEL, SLEWLIMIT, and NEXTPOS parameters are written and the GO flag is set. The controller will do a ramp-up/slew/ramp-down motion profile determined by the DELA, ACCEL, SLEWLIMIT and NEXTPOS parameters.

During ramp-up, at every sample, the ACCEL parameter is added to or subtracted from the VELOCITY parameter, and in turn the VELOCITY value is added to the DESPOS parameter. When the absolute VELOCITY parameter is >= SLEW, the acceleration stops and the motion continues at the SLEWLIMIT rate until a the ramp-down portion of the motion profile is reached. At this point the signed ACCEL parameter is added to or subtracted from the VELOCITY parameter until the VELOCITY parameter reaches 0. At this point the GO flag is cleared by the DSP, and the position move is complete. This ramp-up/slew/rampdown motion profile is often called a trapezoidal profile because the velocity profile is a trapezoid. The position profile is a parabolic (square) function of time**.**

OPERATION

PROFILE GENERATOR

OFF BY ONE ERROR

The profile generator has a possible 1 count endpoint error in position mode. This error occurs when the slew velocity is not reached and the total distance of the move is not divisible by 2. This single count error is fixed at the end of the move.

ABORTING A MOVE

An executing profile can be aborted by clearing the GO flag and setting the DESVEL parameter to 0. This can be done by the host, or the event logic.

When doing an abort, the host can determine when the motion has stopped by polling the MOTION flag. When the MOTION flag is cleared, motion has stopped. At this point the DESPOS parameter can be read to determine the current position.

If a fast stop is needed, the acceleration parameter should be loaded with an appropriate value after DESVEL has been set to 0.

OPERATION

PROFILE GENERATOR

VELOCITY MODE

In velocity mode, the motion is controlled by the commanded velocity (DESVEL) parameter, and unlike position mode, motion parameters (DELA, ACCEL, AND DESVEL) can be changed on the fly. This is useful for for profiling operations, allowing complex profiles to be built up from piecewise line segments with new data sent to the motion controller for each line segment instead of every point. In velocity mode, the GO bit is not used, and the commanded velocity (DESVEL) is manipulated directly.

When the DESVEL parameter is changed, the profile generator will increment or decrement VELOCITY by the current ACCEL value until it equals DESVEL, at that point VELOCITY will stay constant until DESVEL is changed. DESPOS is always incremented by the current VELOCITY parameter in velocity mode.

To stop when in velocity mode, DESVEL is set to 0. The MOTION flag can then be polled to determine when motion has stopped.

The velocity mode can be used for profiling and also for continuous motion: for conveyors, stirrers etc, as nothing “funny” happens when the DESPOS count wraps at 2^32.

Complicated profiles created in velocity mode require quick response from the host to supply new parameters. This requirement can be eased by using the event logic and the motion control blocks to automatically load new motion parameters at position or time breakpoints. (See event and motion control block section)

EXTERNAL PROFILE MODE

In this mode the host is can control the profile by manipulating the DELA and ACCEL parameters. This allows creation of cubic position profiles. To use the External profile mode, the EXTPROFILE flag must be set. When this is set, only the DDA portion of the profile generator is active and does the following calculations every sample::

IAccel <= IAccel + DELA

DELA is a signed 32 bit number that is added to the least significant 32 bits of the IAccel parameter. IAccel is a signed 48 bit number consisting of ACCEL as the most significant 32 bits and ACCELFRACT as the least significant 16 bits.

VELOCITY <= VELOCITY +ACCEL (32 bit signed add with ACCELFRACT ignored)

IDespos <= IDespos + VELOCITY / 2^24 (64 bit signed add)

IDespos is a 64 bit number consisting of DESPOS as the most significant 32 bits and DESPOSF as the least significant 32 bits.

OPERATION

PROFILE GENERATOR

EXTERNAL PROFILE MODE

In the external profile mode, the velocity follower is disabled so the DESVEL parameter will have no effect on operation. GO should not be set when using the external profile mode.

HOMING

One special function of the profile generator is homing, or establishing the initial reference point for encoder position readout. Homing requires that there be some kind of mechanical or optical switch to detect home position, and that this switch is wired to the index input. To detect this the counter control register of the primary encoder needs to be setup to recognize the index input. First,The IDXPOL bit should be written to match the active state of the index input, in other words set to a 1 for active high index signals and 0 for active low index signals. Homing is then accomplished by setting the CLEAR_ON_INDEX (COI) bit in the counter control register. The counter control register is an 8 bit register in the LS byte of a 16 bit parameter. Counter control register bits are defined as follows:

U/D	FILTER	CLRD	COI	IDXPOL	CL/IDX	B	A

Once the clear on index bit is set, host software should start a slow move in the desired direction. It may be necessary to poll a limit switch and the index bit (CL/IDX) before motion is started so that motion is not started when the system is already past the index detection position. The slow move towards home will proceed until the desired edge of the index signal is detected. The counter will be cleared only when the edge of the index signal has been detected and the quadrature inputs (A,B) are both low. When this happens, the encoder count will be loaded from the HOMEPOSP parameter, the profile generator will set the desired position to HOMEPOSP, the COI bit in the counter control register will be cleared, the home FLAG set true and the GO flag set false.

The secondary encoders cannot be used for homing, but can still be preset with any desired count at index by setting the desired preset count in the HOMEPOSS and initializing the secondary encoders counter control register as done above for the primary encoder.

OPERATION

PID LOOP

GENERAL

The second part of the motion controller firmware is the PID loop. The PID loop acts as a feedback loop that keeps the ***actual*** position equal to the ***setpoint*** position.

*The PID loops **actual** position and the **setpoint** position parameters are selected with pointers to allow dual encoder feedback, encoder gearing, and ratioed multi-axis moves.*

Using pointers for the *actual* position reading also allows the use of absolute encoders connected to one of the I/O ports.

The PID loop is enabled by setting the PID flag, and disabled by clearing the PID flag. Clearing the PID flag does 2 things, it disables the PID loop, and sets the PWM value to zero.

The pointers that the PID loop uses are POSENC, for the position encoder, VELENC for the velocity encoder and FOLLOW for the setpoint position. The default value of POSENC and VELENC are ENCP, the primary encoder for the axis. The default value of FOLLOW is DESPOS, the desired position number from the profile generator. For simple motion operations, the POSENC,VELENC, and FOLLOW pointers can be left at their default settings. The PID loop is controlled by 6 main parameters:

KP Proportional term or Gain

KI Integral term

KD Derivative term or Damping

KF1 Velocity feed forward term

KF2 Acceleration feed forward term

KIL Integration limit

The output of the PID loop is the a drive signal that sets PWM and direction signals that control the amount and direction of the current that is applied to the motor. The simplified equation for this drive is:

$$Drive = KP(-E) + KI(\sum -E\Delta T) + KD(-ACTVEL) + KF1(VELOCITY) + KF2(ACCEL)$$

Where E is the position error *(SETPOINT* POSITION - *ACTUAL* POSITION) .

OPERATION

PID LOOP

MAIN PARAMETERS

The six main PID parameters are called tuning parameters and have to be set to match the dynamics of the controlled system.

KP is the most important tuning parameter as it sets the over-all gain or “stiffness” of the servo loop. The KP parameter determines how much restoring force is applied to the motor relative to a given position error. If KP is too low, the overall servo accuracy will be low. If KP is too high it will be hard to make the servo system stable. Depending on encoder counts and load dynamics, values from 50 to 5000 are a reasonable range for KP.

KI is the integral parameter. A feedback loop with only a proportional term (KP) will always have some remaining error caused by the fact that a finite error is necessary to supply the drive needed to correct that error. In a real system with friction and static loads and reasonable values of KP, this error can be significant. The Integral part of the PID loop is used to accumulate small errors over many sample periods, creating a larger and larger correcting drive so that even a small position error will eventually be corrected. This can be useful where friction, spring, or gravity loads cause static error hard to correct with a reasonable KP term.

The Integral term should be used carefully with dynamic loads and can cause instability if not used with caution. One other problem with the integral term is what is sometimes called ‘integral windup’. This happens for example when a position move is made at a faster rate than the servo system can respond, since in this case the real position will lag the desired position for the duration of the move, a large integral term will have accumulated at the end of the move, causing a large, slow to recover overshoot as the accumulated integral term counts are “deaccumulated” once the move is over.

The **KIL** term is a bound on the maximum size of the integral error term. It can help eliminate integral windup, but does limit the amount of drive contributed by the integral term. A KIL value of 32767 allows maximum drive from the integral term. A value of 16384 would limit integral related drive to ½ full scale.

KD is the Damping parameter. It is needed to make the servo system stable, especially at high gains (high KP). The servo control loop is basically a second order linear differential equation whose solution without the damping term is a sine wave. The damping term contributes a exponential decay to the equation.

Higher values of KD are needed with higher values of KP. Higher values are also needed with higher sample rates. This is because damping is dependent on KD and ACTVEL, and the ACTVEL parameter is inversely proportional to the sample rate. Reasonable values of damping are from ~200 to 65535 (max).

OPERATION

PID LOOP

FEEDFORWARD PARAMETERS

The next two terms are called feed-forward terms because they are not part of the motion control *feedback* loop, that is they do not depend on the actual measured motion but rather their values are calculated based on the *desired* motion profile.

KF1is the velocity feed forward term. It supplies an amount of drive proportional to the VELOCITY parameter. Its has two purposes:

1. To counteract the shift in operating point of the servo system due to the damping term when moving at high speed. For example, when moving at a high constant speed, the damping term in the PID loop will offset the position servo loop such that the actual motor profile will lag the profile generators profile. Setting KF1 to KD/256/KDF will compensate for this phenomena. (See KDFIL for value of KDF)

2. To center the operating point of the PID loop about the current velocity. When moving at a high speed, a constant amount of motor drive must be applied just to maintain the motor speed, but in order to apply this drive, a position error must exist. This has the effect of causing the motor profile to lag the profile generators position profile. This is corrected by making KF1 a little more than KD/256/KDF. A reasonable value for KF1 is KD/256/KDF + ~5 to ~200

KF2 is the acceleration feed forward term. It supplies an amount of drive proportional to the profile generators acceleration value. Its purpose is to center the operating point of the feedback loop when accelerating or deccelerating. It can be used to compensate for the undershoot when starting a quickly accelerated motion and overshoot when stopping.

SECONDARY PARAMETERS

KK is a signed bias on the PWM output. It can be used for zeroing servo amplifier outputs, or as a feedforward term when operating into a fixed load (gravity for example)

KFF is a friction feed forward term that is used to overcome friction (stiction) in the drive system. It supplies a selectable amount of drive in the direction of motion. It can also be used to compensate for the deadzone that Hbridges generate with their blanking time.

KDFIL is the derivative filter parameter. It sets the controlling coefficient in a smoothing filter for the calculated velocity. The velocity term is always problematic in digital servo loops because it is calculated from the change in position from one sample interval to the next. The change in encoder readings at small velocities may be less than one count per sample interval so the derivative term in the PID loop will alternate between 0 and 1 * KD, giving very coarsely quantized damping.

OPERATION

PID LOOP

SECONDARY PARAMETERS

To reduce this quantization noise, the *SOFT*DMC incorporates a digital smoothing filter that is applied to the measured velocity before it is used in the PID loop. The KDFIL parameter determines the coefficients of a time weighted running average of the measured velocity. Valid numbers for KDFIL are between 0 and 65535. If KDFIL is set to 0, no filtering takes place. Larger values will result in longer time constants which may have a negative effect on loop stability. A reasonable starting value of KDFIL is 49152. This will make the filtered velocity consist of one part current measured value and 3 parts that are the time weighted sum of all previous velocity values. This will also increase the damping by a factor of 4. The increase in damping factor contributed by KDFIL is:

$$KDF = 1/(1-(KDFIL/65536))$$

*Large values of KDF can result in overflow of the damping term. Encoder counts per sample * KDF must be limited to < 127 or this overflow will occur causing an erratic damping term in the PID loop. For example, using the suggested starting value of KDFIL = 49152, which gives a KDF of 4, and assuming a 500 line encoder in 4X mode, and a sampling rate of 20 KHz, we get 127/4 = 31 = max counts/sample = 18600 RPM.*

Using a large value of KDFIL can reduce the acoustic noise from the motor during slow moves that results from the coarse quantization of the damping.

OPERATION

PID LOOP

FAULT CONDITIONS

There are several possible conditions that can cause loss of control or runaway conditions in the PID control loop, with the possible result of harm to equipment or personnel. One special task of the PID loop is to monitor the servo loops position error and PWM drive signal to check for these system faults.

Faults caused by mechanical problems during normal motion can usually be detected by using a small enough excessive position error limit to disable the PID loop and shut down the PWM drive in this case.

Other faults include system connection and component failure. One obvious connection related fault condition is reversed encoder or motor leads, resulting in positive feedback and immediate runaway. This can be avoided by using keyed connectors to prevent mis-assembly in the field. A small enough excessive position error limit will also help in these cases.

A failed encoder or bad encoder connection, broken encoder wire, etc, can cause runaway when the PID loop is simply holding a static position. The excessive position limit does not help in this case since the PID loop is “blinded” and unable to see the motors motion. The excessive drive detection can be used in this case to shut down the drive.

If more positive detection of electrical faults is needed, one option is to use an extra encoder to detect motion when none is expected. This encoder can connect to one of the alternate *SOFT*DMC encoder inputs, and be monitored by the host or event logic to detect a runaway condition. This motion detection encoder can be as simple as a slotted wheel with a single detector since we only need to detect an accumulation of counts where none are expected and are not concerned with the direction of the counts. When a slotted wheel is used, the alternate encoder would be used in UP/DOWN mode.

OPERATION

PID LOOP

EXCESSIVE POSITION FAULT

One fault is excessive position error. Excessive position error means that the absolute value of the PID loops error in counts is greater than the EXERR parameter for that axis. This can occur because of a mechanical fault (stall), attempting to attain faster velocity or acceleration than the mechanical system can deliver, PID filter values that result in unstable operation, or electrical faults in the drive system. *Having a reasonable value of excessive position limit is a safety issue*.

Note that the maximum excessive position error is 32767. An EXERR value of zero will disable excessive position error checking. This saves some time, so if excessive position error detection is not needed, EXERR should be set to zero.

When an excessive position fault occurs, four things happen: 1. The GO flag will be cleared, 2. The ERROR flag will be set, 3. The PID flag will be cleared, which disables the PID loop and sets the PWM value to zero, and finally 4. The ENA bit is cleared, allowing external hardware to detect the fault condition.

EXCESSIVE DRIVE FAULT

Another fault condition that is monitored by the PID loop is excessive drive. Excessive drive means that the motor drive has exceeded a preset amount (DRIVEMAX). An excessive drive fault is generated when an excessive drive condition lasts longer than selectable number of samples. The excessive drive detection works by comparing the motor drive to the DRIVEMAX parameter every sample. If the motor drive exceeds the DRIVEMAX parameter, the DRIVESUM parameter is incremented. If the motor drive is less than the DRIVEMAX parameter, the DRIVESUM parameter is decremented. When DRIVESUM is decremented, the count is bounded at zero so that it does not underflow. If DRIVESUM becomes equal to or greater than the DRIVELIMIT parameter, an excessive drive fault is generated.

The excessive drive detection is valuable because it can detect fault conditions such as a bad (non counting) encoder that cannot be detected with excessive position error detection alone.

OPERATION

PID LOOP

EXCESSIVE DRIVE FAULT

Two parameters are used to setup the excessive drive detection: DRIVEMAX and DRIVELIMIT. DRIVEMAX would normally be set for close to full scale drive. A reasonable value would be 95% of full scale drive = .95*65535 = 62258. The DRIVELIMIT parameter determines how many samples of excessive drive must occur before an excessive drive fault is generated. At a sample rate of 25 KHz for example, a DRIVELIMIT value of 100 would allow detection of a drive problem that resulted in full scale drive in a 4 mSec. (1/25KHz*100). Larger values of DRIVELIMIT delay fault detection but also prevent false fault triggering. Maximum DRIVELIMIT value is 32767.

If drive monitoring is not needed, DRIVEMAX should be set to zero. This will disable then drive monitoring code.

When an excessive drive fault occurs, four things happen: 1. The GO flag will be cleared, 2. The ERROR flag will be set, 3. The PID flag will be cleared, which disables the PID loop and sets the PWM value to zero, and finally 4. The ENA bit is cleared, allowing external hardware to detect the fault condition. The DRIVESUM parameter can be checked to determine of the fault was an excessive position error or an excessive drive fault. DRIVESUM should be reset to zero if an excessive drive fault has occurred.

RECOVERY FROM FAULT CONDITIONS

Once an excessive position error or excessive drive fault has occurred, recovery can be done in two different ways. One recovery option is to set the ENA bit and then do a complete re-homing operation on the axis that has suffered the fault. This has the disadvantage that it may be too time consuming to be practical. The other option is to read the current position (usually ENCP), and set the desired position equal to the current position before proceeding with re-enabling the PID loop and setting the ENA bit. FAULT CONDITIONS

OPERATION

TUNING

DMCTUNE

The PID loop tuning parameters must be adjusted for each different motor/load/amplifier combination. A tuning program (DMCTUNE.EXE) is provided with the *SOFT*DMC firmware and allows manual adjustment of the main PID tuning parameters while displaying the servo systems response. DMCTUNE displays 4 parameters: The programmed motion profile (Green), The actual motion profile (Yellow), The motor drive signal (Red) and the magnified error, that is the difference between programmed profile and actual profile (Violet).

DMCTUNE will attempt to access a PCI or PC/104 interface motion controller if invoked with no command line parameters. For use with serial interfaced *SOFT*DMC controllers, DMCTUNE should be invoked with the COM port and BAUD rate on the command line, for example:

C>DMCTUNE COM1 115200

DMCTUNE COMMANDS:

UpArrow/DownArrow	Chose parameter to change
RightArrow/LeftArrow	Increment/decrement parameter 10%
End/PageDown	Increment/decrement parameter 1%
M/m	Set parameter to maximum
Z/z	Set parameter to minimum
Insert	Do step
S/s	Save current motor parameters to temp buffer
R/r	Restore current motor parameters from temp buffer
E/e	Export all parameters to file
I/i	Import all parameters from file
L/l	Print all parameters to ASCII list file
ALTX	Exit program

OPERATION

TUNING

TUNING PROCEDURE

It is suggested that the PID loop parameters be adjusted in the following order:

KP and KD: First the gain (KP) and damping (KD) should be adjusted. What you are trying to do here is get the highest gain possible with a commensurate amount of damping to prevent overshoot and ringing during a fast step. A fast step here means one that is faster than the mechanics can follow. This is done by setting the acceleration and velocity numbers very high (with the 'M' command).

The feed-forward terms are adjusted next.

Feedforward term KF1 should be adjusted next. This is done by doing a fast motion but one with slow enough acceleration to be just within the capabilities of the servo system to follow. The servo position will lag the profile during the move. A good starting value of KF1 is KD/256/KDF, This will compensate for most (but not all) of the lag. Then KF1 is slowly increased until the servo system response matches as closely as possible to the profile it is following.

Then KF2 is adjusted to compensate for the small lag at the beginning of a move and small overshoot at the end. Note that the errors corrected by KF2 will be very small unless you are doing quite fast moves, close to the dynamic limits of the servo system. Adjusting KF2 is done by setting the velocity and acceleration for a fast move that reaches slew velocity for about 3/4 of the move, thus the motion profile will have a first section (1/8 of the time total time) with constant positive acceleration, a middle section (3/4 of the total time) with constant velocity (0 acceleration) and an end section (1/8 of the total time)with constant negative acceleration. KF2 will only adjust the portions of the profile when acceleration <> 0, that is during ramp-up and ramp-down.

Finally KI and KIL are adjusted. For best overall accuracy KI should be used for correcting the last remaining error after all other PID tuning parameters have been adjusted. The Integral term can reduce static error to 0 counts, and improve dynamic (profile following) error. Too large an integral amount will result in instabilities. If the motion system is ever operated open-loop, the integral term should be disabled by setting the integral limit (KIL)to 0. This also has the advantage of bypassing the Integral part of the firmwares PID loop, speeding up the loop and allowing higher sampling rates.

OPERATION

GEARING

GENERAL

The *SOFT*DMC has the capability of gearing one or more axis to another so that precise ratioed multi-axis moves can be done with one master axis generating the motion profile and multiple slaves following the master axis at selectable gear ratios. Gearing can also be used to allow one or more axis to follow an external encoder with a selectable ratio and offset. The source parameters for gearing are accessed via pointers (GEARPPARAM and GEARSPARAM) so they can be any local (AXIS) or global parameters. The two gear ratios available per axis (GEARP and GEARS) are 32 bit signed numbers used as 8.24 (8 bit integer part and 24 bit fractional part) numbers. This allows positive and negative ratios from 1/(2^24) to ~127/1 with resolution better than 1 PPM for ratios from 1/16 to 16/1.

The 8.24 scaling of the gear ratios means that a ratio of 1/1 is represented by a GEARP or GEARS value of 16777216 (2^24). A positive ratio means that the geared axis will move in the same direction as the master axis. A negative ratio means that the geared axis will move in the opposite direction as the master axis.

In addition to the gear ratio, there is an offset parameter to set initial conditions for geared moves (GEARPOFFSET and GEARSOFFSET).

The result of the gearing operation is a set of global 32 bit parameters, 2 per axis: GEAREDP(N) and GEAREDS(N). The equation for the GEAREDP(N) and GEAREDS(N) is:

For primary gear:

GEAREDP(N) = GEARP(N)*@GEARPPARAM(N)+GEARPOFFSET(N)

For secondary gear:

GEAREDS(N) = GEARS(N)*@GEARSPARAM(N)+GEARSOFFSET(N)

OPERATION

GEARING

GEAR VERSUS COPY

The GEARING operation is relatively expensive (500 nSec per gear) and is disabled by default. Primary and or secondary gearing are enabled by having a value of the GEARP or GEARS (ratio) parameter not equal to zero. If the ratio parameter equals zero (the default value), a ratio of 1 will be used, which is just a copy and add operation. This means that a simple add and copy operation is always done by default. The default copy and add operation is useful for moving a local axis parameter to global memory so it becomes available to all the other axis. The equation for the copy operation is:

For primary gear:

GEAREDP(N) =@GEARPPARAM(N)+GEARPOFFSET(N)

For secondary gear:

GEAREDS(N) =@GEARSPARAM(N)+GEARSOFFSET(N)

OPERATION

GEARING

OFFSET PART

When initializing geared multi axis operations, even if only the copy part of gearing is used, you must initialize the offset part of the gear equation so that you end up with the slave axis's *geared* position equal to its current position or you will generate an uncontrolled motion in the slave when you change the slaves FOLLOW pointer to point to the GEARED parameter. *This is also why you should always change the FOLLOW pointer last when changing gearing.*

If Y represents the geared axis' initial position and X represents the master axis initial position so that Y=MX+B where M is the gear ratio and B is the gear offset, you must initialize B to equal Y-MX before changing the FOLLOW pointer.

MASTER-SLAVE GEARED EXAMPLE

A common use of gearing is to do a multi-axis move with a specific gear ratio between the axis. The following short discussion will show one way this can done for a 2 axis move (axis 0 = master axis 1 = slave):

First the GEARPPARAM pointer of axis 0 is initialized to point to the DESPOS parameter of axis 0, then the gear ratio for the slave axis is written to the GEARP parameter of axis 0. Next the GEARPOFFSET parameter of the master axis is written. Finally the FOLLOW pointer of axis 1 is initialized to point to the GEAREDP0 global parameter.

ENCODER GEARING EXAMPLE

It is possible to have one or more axis follow the primary or secondary encoder of any other axis with a selectable ratio and offset. For example if we wanted axis 0 to follow the secondary encoder of axis 3, we could proceed as follows:

First set the GEARPPARAM parameter of axis 3 to point to ENCS, then set the GEARP parameter of axis 3 to zero to enable copying instead of gearing. Next set the GEARPOFFET parameter of axis 3 to zero. This results in the secondary encoder (ENCS) of axis 3 being copied into the global parameter GEAREDP3. Next we setup axis 0 to scale this parameter and use it as its setpoint. To do this we set axis 0's GEARPPARAM to point to GEAREDP3, then set axis 0's GEARP parameter to the desired gear ratio, set axis 0's GEARPOFFET parameter to the desired offset, and finally set axis 0's FOLLOW parameter to point to GEAREDP0.

OPERATION

EVENT LOGIC

GENERAL

The *SOFT*DMC has an extremely flexible built in, real time (within one sample period) event logic system for handling internal and external events. These events include limit switch actuation, position/velocity/acceleration or time breakpoints, external hardware events, the host setting or clearing a flag etc etc. The result of a detected event can be Starting a motion profile, aborting a motion profile, loading new acceleration or velocity parameters, Motion register block swapping, PID filter block swapping, interrupt generation, I/O bit manipulation, etc etc.

CONDITIONS AND OPERATIONS

Basic event logic operations are of this form:

If STATICMODE then if CONDITION then OPERATION

If EDGEMODE then if ^CONDITION then OPERATION

Where ^CONDITION means the the current condition is true and the condition at the previous sample interval was false.

In Static mode the OPERATION is executed whenever the CONDITION is true. *This may mean that the OPERATION will be repeated every sample interval.* This is useful when you want some parameter or some I/O to follow the input condition, or you want to maintain a count of samples when a condition is true. This can be useful for example when using limit switch activation as a *hard* limit, to disable the PID loop, disabling motor drive.

Edge mode is used when the OPERATION should only occur on a change in the condition: when the condition is first met. This is useful for operation such as setting an interrupt when the condition is first detected. For example if we wished to generate an interrupt when GO return to 0 at the end of a move, we would use an edgemode event so that when the event is triggered, (by go changing to 0) the IRQ FlipFlop is only set once, allowing the interrupt service routine to clear the IRQ by simply writing to the CLRIRQ parameter.

OPERATION

EVENT LOGIC

GLOBAL AND LOCAL (AXIS) EVENTS

Event blocks are available in both AXIS memory and GLOBAL memory. GLOBAL events only have access to global memory, axis events have access to the current axis and global memory. The number of active global events is determined by the GEVENTS parameter. The number of active axis events is determined by the EVENTS parameter. Events in AXIS memory share space with motion and filter blocks, so you must be careful not to allocate an axis event on top of an existing filter or motion control block. If EVENTS or GEVENTS is <>0 then all axis events or global events from 1 to EVENTS or GEVENTS are enabled. Event blocks are allocated from low memory to high, starting with event 1. You should not activate a event without properly initializing all fields or unpredictable behavior can result.

All event blocks consist of eight 16 bit words. These words have somewhat different definitions for logical events and compare events. Note that the first four words in the event block are *condition* related, and the last four words are *operation* related. Compare and logical *conditions* and be combined with logical, add, and copy *operations* in any way, so that there are six basic event types.

Events are processed in sequence, that is, event 1 is processed before event 2. This sequence is important when the operation part of one event affects subsequent events, or precise timing is needed. For example, if event 1 sets an I/O bit and event 2 clears the I/O bit, the I/O bit will be only be set for one event time = ~300 nS. If on the other hand event 2 sets an I/O bit and event 1 clears it, the I/O bit will remain set for 1 sample period.

OPERATION

EVENT LOGIC

LOGICAL CONDITION EVENT STRUCTURE

CONDPTR	Pointer to condition parameter
	Bit 15 of CONDPTR must be low to select logical operation
CONDXOR	Condition XOR (polarity) mask
CONDAND/PTR	Condition AND (bit selection) mask
CONDFLAG	Result of compare (for edge detection or subsequent logic)
OPERPTR	Pointer to output parameter
	Bit 15 high selects edge mode / low selects static mode
	Bit 14 high selects add mode/ low selects logical operation mode
	Bit 13 high selects copy mode
OPERXOR	Operation XOR (polarity) mask or ADDed value
OPERAND	Operation AND (bit selection)
OPEROR/SRC	Operation OR (output data) or pointer to copy source

OPERATION

EVENT LOGIC

COMPARE CONDITION EVENT STRUCTURE

CONDPTR	Pointer or pointer to pointer to compared parameter (A)
	Bit 15 of CONDPTR must be high to select comparison operation
	Bit 14 high selects 32 bit unsigned compare / low selects 16 bit signed compare
	Bit 13 high selects doubly indirect mode (pointer to a pointer)
CONDXOR	0x0000 for normal compare 0xFFFF to reverse result
COMDAND/PTR	Pointer or pointer to pointer to compared value (B)
CONDFLAG	Result of compare (for edge detection or subsequent logic)
OPERPTR	Pointer to output parameter
	Bit 15 high selects edge mode / low selects static mode
	Bit 14 high selects add mode/ low selects logical operation mode
	Bit 13 high selects copy mode
OPERXOR	Operation XOR (polarity) mask or ADDed value
OPERAND	Operation AND (bit selection)
OPEROR/SRC	Operation OR (output data) or pointer to copy source

COMPARE EVENTS *CAN* USE A POINTER TO A POINTER

Notice that the condition (A) and compared value (B) can accessed by a pointer to a pointer if CONDPTR bit 13=1. This is to allow reference to parameters (such as breakpoints) that are in turn accessed by pointer, for example the breakpoint value and compared parameter in a motion control block. By using the double indirection, the sense of the compare can be reversed by reversing the pointers in the motion control block so that a linked list of motion control blocks can use a single compare event to swap motion control blocks, which will update the breakpoint, *and* simultaneously change the sense of the compare.

OPERATION

EVENT LOGIC

CONDITIONS

There are 3 types of detectable CONDITIONs:

1. Logical (16 bit bit field)

2. Arithmetic compare (16 bit unsigned)

3. Arithmetic compare (32 bit signed)

Logical CONDITIONs are evaluated by XORing the condition parameter with the CONDXOR mask and then ANDing the result with the CONDAND mask. If the result is <> 0 then the OPERATION occurs. Logical conditions could include limit switch detection, detection of internal flags (GO,HOME,SLEW, other event generated flags), etc.

Compare CONDITIONs work by comparing two 16 or 32 bit parameters (A and B). First the two numbers are compared. The compare result is true when A<B.The result of the compare is then XORed with the CONDXOR mask CONDXOR can be 0x0000 for normal compares or 0xFFFF to change the direction of the compare. By swapping A and B and selecting 0x0000 or 0xFFFF as the CONDXOR mask, four compare types are possible:

A<B (Direct)

A>=B (Result inverted)

A>B (Pointers swapped)

A<=B (Pointers swapped, result inverted)

Compare CONDITIONS are useful for time, position, velocity or acceleration breakpoints, timers, event counters etc.

OPERATION

EVENT LOGIC

OPERATIONS

There are 3 operations possible as the result of an CONDITION being met.

1. Logical (OPERPTR bit 14=0 and bit 13=0)

2. Add (OPERPTR bit 14=1 and bit13=0)

3. Copy (OPERPTR bit 14=0 and bit 13=1)

Logical operations read the output parameter into a variable, XORs the variable with the OPERXOR mask, ANDs the variable with the OPERAND mask, ORs the variable with the OPEROR mask and then writes this variable back to the output parameter. This allows setting, clearing or toggling any bit or sets of bits in the output parameter (where OPERPTR points).

The Add operation reads the output parameter into a variable, adds the OPERXOR parameter to the variable ANDs the variable with the OPERAND mask, ORs the variable with the OPEROR mask and then writes this data back to the output parameter. This is useful for event counting, timing etc.

Copy operations fetch the OPEROR/SRC parameter, XORs the fetched data with the OPERXOR parameter, ANDs it with the OPERAND parameter and writes the result to the output data (where OPERPTR points)

All three operations can be edge triggered or static. Edge triggered mode is selected by setting OPERPTR bit15 =1.

OPERATION

MOTION PARAMETER BLOCKS

GENERAL

The main motion controlling registers are duplicated in a series of parameter blocks that allow linked list type sequential motions or profiling commands to be made without host involvement. The standard implementations of SOFTDMC have 9 motion control or filter parameter blocks available, blocks 1 through 9. These are shared between Motion control blocks, filter parameter blocks and events. Make sure that you do not allocate overlapping blocks! The motion register blocks are the same size as filter register blocks and are allocated from high memory down to low memory, to allow a mix of events and register blocks to co-exist.

Note that there is an additional block, the ‘working block ‘ that is always be used for all motion control. When a SWAPMOT command is issued, (by setting the SWAPMOT flag), the working block will be overwritten by the block pointed to by the NEXTBLOCK parameter in the working block. The SWAP command can be issued by the host or can be internally generated by the EVENT LOGIC. *Note, If you do not need high speed operations and command buffering, the linked list nature of the motion control blocks can be ignored, and only the working block will be used.*

Each motion control block contains the following parameters:

MOTION CONTROL PARAMETERS

NEXTPOS	LONG	AXIS	Next position for move
SLEWLIMIT	LONG	AXIS	Slew speed during move
DESVEL	LONG	AXIS	Commanded velocity for velocity or profile mode
ACCEL	LONG	AXIS	Acceleration value
DELA	LONG	AXIS	Delta acceleration for cubic profiles
BREAKPOINT	LONG	AXIS	Local breakpoint value
BREAKA	PTR	AXIS	Normally points to value compared to BREAKPOINT
BREAKB	PTR	AXIS	Points to breakpoint value (normally BREAKPOINT but can point to anything)
NEXTMOTBLOCK	PTR	AXIS	Pointer to next motion control block
USER	UINT	AXIS	User variable or pointer

OPERATION

MOTION PARAMETER BLOCKS

MOTION CONTROL PARAMETERS

The first six parameters in the block are standard motion control parameters and are discussed in the PID LOOP and PROFILE sections of this manual. The last 4 parameters control the operation of the motion control blocks.

NEXTMOTBLOCK is a pointer the the block to be loaded when the SWAP command is executed. By initializing NEXTMOTBLOCK pointers in each block to point to the next block, a circular list of blocks can be created. This list of blocks can then be loaded sequentially by a programmed event. The motion register block firmware does not actually handle the breakpoint comparison itself, rather the compare is done with an event (see EVENT LOGIC). programmed to compare the parameter pointed to by BREAKA and the parameter pointed to by BREAKB (normally the BREAKPOINT value in the working motion control block).

Note that both the variable parameter that is monitored and the breakpoint value are accessed via pointers. This allows the breakpoint value and monitored parameter to be swapped to change the sense of the event logics compare from > to <=.

OPERATION

FILTER PARAMETER BLOCKS

GENERAL

Like the motion control registers, the PID loop filter parameters are duplicated in a series of parameter blocks that allow linked list type sequential updating of all filter parameters between samples. The standard implementations of *SOFT*DMC have 9 filter parameter or motion control blocks available, blocks 1 through 9. Note that there is an additional block that is the working block, that the PID loop uses for its parameters. When a SWAPFIL command is issued, (by setting the SWAPFIL flag), the working filter parameter block will be overwritten by the block pointed to by the NEXTBLOCK parameter in the current working filter block. The SWAPFIL command can be issued by the host or can be internally generated by the EVENT LOGIC.

Note, If you do not need high speed operations and command buffering, the linked list nature of the filter parameter blocks can be ignored, and only the working block will be used.

Each filter parameter block contains the following parameters:

PID LOOP TUNING PARAMETERS

KK	UINT	AXIS	PWM offset or bias
KP	UINT	AXIS	Proportional constant
KD	UINT	AXIS	Derivative constant
KA	UINT	AXIS	Not used
KI	LONG	AXIS	Integral constant
KIL	UINT	AXIS	Integral limit
KF1	UINT	AXIS	Velocity feed forward term
KF2	UINT	AXIS	Acceleration feed forward term
KF3	UINT	AXIS	Not used
KFF	UINT	AXIS	Friction feed forward term

OPERATION

FILTER PARAMETER BLOCKS

PID LOOP TUNING PARAMETERS

KDFIL	UINT	AXIS	Derivative term filter coefficient
POSENC	PTR	AXIS	PID loops pointer to position encoder (default = ENCP)
VELENC	PTR	AXIS	PID loops pointer to position encoder used for velocity calculation (default = ENCP)
NEXTFILBLOCK	PTR	AXIS	Pointer to next filter block

A common use of the filter parameter blocks is to have a set of filter parameters used for static conditions and another set used for dynamic (in motion) conditions. The blocks can be changed automatically with two events, one programmed to look for the GO flag being set and one looking for the GO flag being cleared.

OPERATION

USER PARAMETERS AND UTILITIES

GENERAL

A number of parameters and utility functions are available for debugging and user applications. The hardware utilities (LEDs and speaker) are only available on certain platforms

LEDS

Some *SOFT*DMC FPGA platforms have debug LEDS on the circuit card that can be useful for debugging and monitoring. The LEDs can be programmed to display the N least significant bits (where N is the number of available LEDs) of any global or axis parameter. Two parameters control the LEDs, LEDAXIS and LED. LEDAXIS specifies which axis is monitored, it is a dont-care value for global parameters. LED is a pointer the specifies the parameter to monitor. If LED is zero (a null pointer) the LEDs will not be driven. The LEDs are updated once per sample period.

SPEAKER

Some *SOFT*DMC FPGA platforms have a simple I/O bit controlled Speaker. A single pointer (BEEPER) determines which parameter drives the speaker. The most significant bit (B15) of the parameter that BEEPER points to determines whether the speaker current is enabled or disabled. If BEEPER is zero (a null pointer) the speaker will not be driven. The speaker bit is updated once per sample period

PHASE ACCUMULATOR

The user phase accumulators are 32 bit accumulators that have a constant 32 bit value added every sample period. They can be used for general timing tasks, or in conjunction with the event logic for rate generation, timeouts etc, etc. There is one phase accumulator available per axis, plus one global phase accumulator. Each phase accumulator has two 32 bit parameters, the phase constant that is added every sample, and the actual accumulator. The per axis phase constant is called PHASEK and the per axis phase accumulator is called PHASEA. The global phase constant is called GPHASEK and the global phase accumulator is called GPHASEA.

OPERATION

USER PARAMETERS AND UTILITIES

PHASE ACCUMULATOR

As an example of phase accumulator usage, here is one way to setup a rate generator: Say we want to generate a host interrupt once per second, and that we have a 10 KHz sample rate. The MSB (bit 31) of the phase accumulator will toggle at a frequency of:

$$SAMPLERATE * PHASEK / 2^{32}$$

So we choose a PHASEK of 2^32/10 KHz (~429497) for a one second toggle rate of the PHASEA MSB. Then we initialize the interrupt control register (IRQREG) to select the desired interrupt. Finally we set up a edge mode logical event that sets the SETIRQ flag on the rising edge of the PHASEA MSB. The interrupt service routine would then write to the CLRIRQ flag to clear the IRQ.

Note that the though average frequency of the generated 1 Hz signal is very accurate (XTAL accuracy basically) there is a one sample period jitter (1/10KHz = 100 uSec in this example) in the generated rate.

Another example of phase accumulator usage is the DMCTUNE program. DMCTUNE uses the global phase accumulator to set the sample rate for collecting accurately timed samples of the internal parameters during motion, offloading this critical timing task from the host.