*SOFT*DMC

FPGA based

Digital Motion Controller

4.4 --- SW Rev. 4.103+ / HW Rev. 30+

This page intentionally not blank

Table of Contents

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 ,Spartan3, Spartan3E and Virtex FPGAs.

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.

*SOFT*DMC supports brush, 2 phase stepper, 3 phase brushless and 3 phase AC induction motors. Steppers and 3 Phase brushless motors can be operated open-loop (no encoder).

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. 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.

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 (>50KHz for 4 axis simultaneous motion, 96 MHz clock) support small and fast drive systems.

Programmable event logic allows real time response to internal (position, time, velocity, flags, etc) and external (limit switches, sensors, etc) events. Event logic combined with the FIFOed host interface allow fully buffered profiling operations and filter changes based on breakpoints or external events.

Efficient dual FIFO host interface allows real time and queued commands to proceed simultaneously. The buffered synchronous design of the host interface allows almost any parameter to be changed during motion. Wait-on-flag tokens allow precise queued command timing to one sample period. 16 bit PC/104, 8 bit microcontroller, PCI, USB, SPI, serial, and other host interface types are available.

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 four 16 bit registers starting at the BASE address (Note that all addresses are *byte* addresses) All host interaction with the *SOFT*DMC is done via these registers. The serially and USB interfaced *SOFT*DMC configurations are similar but are accessed via either an ASCII command set with hexadecimal parameters or the LBP protocol. SPI interfaced versions include the register address, command and data in a single 24 bit serial frame.

BUS INTERFACE DESCRIPTION

REGISTER MAP

BASE ADDRESS	ICD FIFO
BASE ADDRESS +4	QCD FIFO
BASE ADDRESS +8	STATUS REGISTER A
BASE ADDRESS +12	STATUS REGISTER B

COMMAND/DATA FIFOS

Most communication to the *SOFT*DMCs internal processor is done via the two command/data FIFOs. These FIFOs are called the Immediate Command/Data FIFO (ICD FIFO) and the Queued Command/Data FIFO (QCD FIFO). The ICD FIFO and the QCD FIFO function identically though the ICD FIFO is typically smaller than the QCD FIFO. The reason that there are 2 FIFOs is to allow immediate I/O requests to be serviced (via the ICD FIFO) even if the QCD FIFO is busy with queued commands.

With either FIFO, commands and parameters are written sequentially to the desired FIFO, with data following commands in the case of write commands.

Note: *When first configured, the FPGA disables access to the ICD FIFO to prevent the last bytes of the configuration data from being misinterpreted as commands. To enable the ICD FIFO, 0x0000 should be written to status register A.*

READ-BACK FIFOS

When the ICD of QCD FIFO locations are read, they return data from the read back FIFOs. These FIFOs are used to return data from *SOFT*DMC to the host. Since the read-back FIFOs are independent of the Command/Data FIFOs, read and write commands may be mixed. They also allow multiple read commands to be issued before reading back the data.

HOST INTERFACE

BUS INTERFACE DESCRIPTION

STATUS REGISTER A

Status register A is used to determine FIFO and interrupt status. Read only FIFO status is available in the top 8 bits of status register A. All FIFO status bits are active high. The bottom 8 bits are read/write interrupt status bits. When *SOFT*DMC is in the firmware download mode, Status register A is used to write the firmware address.

STATUS REGISTER A READ-ONLY BITS

IFF	IFH	QFF	QFH	IRE	IRH	QRE	QRH	ICR7	ICR6	ICR5	ICR4	ICR3	ICR2	ICR1	ICR0

BIT 15 IFF = Immediate Command/Data FIFO full flag

BIT 14 IFH = Immediate Command/Data FIFO half full flag

BIT 13 QFF = Queued Command/Data FIFO full flag

BIT 12 QFH = Queued Command/Data FIFO half full flag

BIT 11 IRE = Immediate read-back FIFO empty flag

BIT 10 IRH = Immediate read-back FIFO half full flag

BIT 9 QRE = Queued read-back FIFO empty flag

BIT 8 QRH = Queued read-back FIFO half full flag

BIT 7..0 Bits 0 through 7 of status register A reflect the lower 8 bits of the interrupt cause register.

STATUS REGISTER A WRITE-ONLY BITS

RS	CI	X	X	X	X	X	X	CI7	CI6	CI5	CI4	CI3	CI2	CI1	CI0

BIT15 RESETS *SOFT*DMC CPU and enables firmware download mode

BIT14 Clear Interrupt Bit(s).

BIT 7..0 Bits CI0 ..CI7 specify which bits to clear in the ICR when bit 14 (CIB) is high. In other words, to clear ICR0, you would write 0x4001 to status register A, to clear all ICR bits you would write 0x40FF.

HOST INTERFACE

BUS INTERFACE DESCRIPTION

STATUS REGISTER B

Status register B is a general purpose status register used to convey real time information to the host. It is normally used by *SOFT*DMC events. When *SOFT*DMC is in the firmware download mode, Status register B is used to read or write firmware data.

COMMANDS

Host communication consists of sending commands and data and reading returned data from either of the FIFOs. There are three basic command types: read commands, write commands and wait commands.

All commands share a similar structure:

W	S1	S0	A2	A1	A0	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

BIT 15 Write bit - 1 for writes, 0 for reads

BITS 14,13 Size field:

- 0,0 One 16 bit word
- 0,1 Two 16 bit words (one long)
- 1,0 Special for wait commands and block writes
- 1,1 four 16 bit words (one quad)

BITS 12,11,10 Axis specifier

BITS 9..0 Parameter address

BITS 9..0 Word count on block write commands

HOST INTERFACE

COMMANDS

WRITE COMMANDS

All write commands have bit 15 set. Bits 14 and 13 specify the write data size. The least significant 10 bits specify the parameter address where the data is to be written. For write commands with data size greater than one word, the data is written in least significant to most significant order.

WRITE_WORD (16 BITS) COMMAND

1	**0**	**0**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

WRITE_LONG (32 BITS) COMMAND

1	**0**	**1**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

WRITE_QUAD (64 BITS) COMMAND

1	**1**	**1**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

EXAMPLE LONG WRITE PROCEDURE

A polled host procedure for writing a 32 bit *SOFT*DMC parameter via the ICD FIFO is as follows:

1. Check status register A for ICD Half Full Flag. If the flag is clear, there is enough room for a write command and following data.

2. Write WRITE_LONG command to ICD FIFO

3. Write least significant 16 bits of 32 bit parameter to ICD FIFO

4. Write most significant 16 bits of 32 bit parameter to ICD FIFO

HOST INTERFACE

COMMANDS

WRITE BLOCK COMMAND

The write block command is provided for writing larger amounts of data, it differs from the other write commands in that the word count is specified in the least significant bits of the command and that specifying the starting parameter address requires an additional word of data following the write block command in the FIFO.

WRITE BLOCK COMMAND

1	**1**	**0**	**A2**	**A1**	**A0**	**C9**	**C8**	**C7**	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**

Bits 9..0 (C9..C0) are the block write count bits and specify the number of words to write. The number of words written is count +1, That is, a count of 0 specifies a single data write, and a count of 63 would cause 64 words to be written.

The Block write command is always followed by a:

PARAMETER ADDRESS

1	**1**	**0**	**X**	**X**	**X**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

Which specifies the starting parameter address for the block write. The parameter address is incremented for each successive data item.

NOTE: Block writes are always atomic, that is, the host interface code will always write the entire block before starting the next sample period, also a block write will not begin until sufficient data is available in the FIFO to complete the command, therefore large block writes should start at the beginning of the host interface portion of the DSPs cycle. This can be accomplished with a wait command that polls the SYNC flag (see below). Block writes that take longer than the available host interface time will cause a delay in starting of the next sample period.

HOST INTERFACE

COMMANDS

EXAMPLE WRITE BLOCK PROCEDURE

A polled host procedure for writing a block < (ICDFIFOSize/2 -2) of *SOFT*DMC parameters via the ICD FIFO is as follows:

1. Check status register A for ICD Half Full Flag. If the flag is clear, there is enough room for a write command and following data.

2. Write WRITE_BLOCK command with word count to ICD FIFO

3. Write parameter address to ICD FIFO

4. Write count+1 data words to ICD FIFO

READ COMMANDS

All read commands have bit 15 = 0, and bits 14 and 13 specify the read data size. The least significant 10 bits specify the parameter address where the data is to be read from. For read commands with data size greater than one word, the data is read in least significant to most significant order.

READ_WORD (16 BITS) COMMAND

0	**0**	**0**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

READ_LONG (32 BITS) COMMAND

0	**0**	**1**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

READ_QUAD (64 BITS) COMMAND

0	**1**	**1**	**A2**	**A1**	**A0**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

HOST INTERFACE

COMMANDS

EXAMPLE LONG READ PROCEDURE

A polled host procedure for reading a 32 bit *SOFT*DMC parameter via the ICD FIFO is as follows:

1. Check status register A for ICD Half Full Flag. If the flag is clear, there is enough room for a read command, otherwise wait.
2. Write READ_LONG command to ICD FIFO
3. Wait until IFIFO Readback Empty flag (IRE) is false
4. Read least significant 16 bits of 32 bit parameter from ICD FIFO
5. Wait until IFIFO Readback Empty flag (IRE) is false
6. Read most significant 16 bits of 32 bit parameter from ICD FIFO

WAIT ON FLAG COMMANDS

In addition to the read and write commands, a wait-on-flag command is available to synchronize FIFO operations to internal *SOFT*DMC conditions. The wait on flag command pauses IFIFO or QFIFO command processing until the desired logical condition is met. The logical condition test is done by first XORing the specified parameter with FLAGXOR parameter, then anding the result with the FLAGAND parameter. If the result of the test is non-zero, the WAIT-ON-FLAG token is removed from the FIFO, and FIFO processing can continue. Note that FLAGXOR and FLAGAND are per axis parameters. Wait on flag commands will work in either the ICD or QCD FIFOs but it is suggested that they only be used in the QCD FIFO. This is because it is possible to generate a deadlock that can only be exited via a reset if a unsatisfied wait command holds up both FIFOs.

0	1	0	A2	A1	A0	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Wait on flag commands are useful for operations like profiling where groups of parameter updates are synchronized to position or sample count break-points. In this case the flag that the wait on flag command is polling would typically be set with a compare event. Wait on flag commands can also be use to reduce host polling when doing indexing type operations, allowing multiple sequential trapezoidal profiles to be queued in the FIFO with a wait on flag(GO) separating the individual motion profiles.

HOST INTERFACE

COMMANDS

HOST SYNCHRONIZATION

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 the 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 operation.

During the host interface part of the cycle the DSP polls both ICD and QCD FIFOs and executes any commands found. Commands are only executed when all the data required by the command is available in the FIFO. Sometimes it is desirable to synchronize host data reads and writes with the internal DSP. For example, it might be desirable that a group of parameter writes be accomplished during a single sample period. Since host FIFO writes are asynchronous to the internal DSP sample timing, a write command might be executed near the end of the host interface cycle. This could mean that subsequent write commands would be executed in the next sample period. To avoid splitting groups of commands between samples, a special flag is available, the SYNC flag.

SYNC FLAG

The SYNC flag is set when the DSP *first* enters the host interface part of its processing cycle and clear for the rest of the host interface time. By using the wait on flag command with the SYNC flag as a parameter, command parsing can be paused until the start of the host interface portion of the DSPs cycle. If a wait on SYNC flag command precedes a group of read or write commands, that group of commands will be held up in the FIFO until the *start* of the next host interface cycle, allowing the full host interface cycle time to execute the group of commands.

The wait on SYNC flag procedure can still fail to keep a group of parameter updates constrained to one sample period in some circumstances. This can happen if the host parameter update rate is too slow or the host is interrupted while writing parameters. In this case processing of the parameter group begins at the start of the host interface time, but not all parameter update data is available in the FIFO by the end of the host interface cycle. To avoid this problem, A separate wait on flag command that precedes the wait on SYNC command is issued. Then a parameter write command sent through the ICD FIFO sets the flag of the first wait on flag command once all the queued parameter writes are in the QCD FIFO. The wait on SYNC flag will then pause the FIFO until the next host interface period.

HOST INTERFACE

COMMANDS

USING THE FIFOS WITH INTERRUPTS

The most efficient way to interface to *SOFT*DMC FIFOs is via interrupts. This is accomplished by placing set interrupt commands (a word parameter write command to the IRQCAUSE Register, the ICR) in the FIFO after a block of commands and data loaded in the FIFO. This way, an interrupt will be generated after the block of commands have been parsed, signaling the host that new commands may be loaded in the Command/Data FIFOs or data may be read back. The interrupt service routine can determine the interrupt source by reading the least significant 8 bits of status register A, which will reflect the data written to the ICR.

The interrupt service routine can clear one or all bits in the ICR by writing to Status register A.

The interrupt hardware can be used in polled mode by using the same technique of putting write to ICR commands in the FIFO after a block of read or write commands, and polling status register A.

For more details on the interrupt hardware, see the INTERNAL HARDWARE section below.

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. Two registers are involved in program downloading: the program address register, and the program data register.

PROGRAM ADDRESS REGISTER @ BASE + 8

L	**CI**	**X**	**X**	**X**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

PROGRAM DATA REGISTER @BASE + 12

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

DOWNLOAD PROCESS

The process for downloading new DSP firmware is as follows: All the words of DSP program are written by first writing the target address Ored with bit 15 (Load bit) 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, the DSP is started is by writing a 0 to the DSP program address register, starting execution of the new code.

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 INCLUDE4.INC and BITS4.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 INCLUDE4.INC file lists all parameters.

MAKEINC

The supplied utility program MAKEINC will translate the INCLUDE4.INC and BITS4.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 INCLUDE4.INC SOFTDMC1.H C

(Create C include file of parameter addresses)

MAKEINC INCLUDE4.INC MOTPARMS.PAS P S Loc

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

PARAMETER TYPES

There are six different types externally useable parameters:

FLAG: Flags are a 16 bit parameters, 0xFFFF is "true" 0 is false.

PTR: 11 bit address pointer range 0 to 2047

INT: 16 bit signed number range -32,768 to 32,767

UINT: 16 bit unsigned number range 0 to 65,535

LONG: 32 bit signed number range -2,147,483,648 to 2,147,483,647

ULONG: 32 bit unsigned number range 0 to 4,294,967,295

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
OPERATION FLAGS			
GO	FLAG	AXIS	Start profile when set true, cleared when done
PID	FLAG	AXIS	Enable PID portion of control loop when true
PROFILE	UINT	AXIS	0= Off, 1=Trapezoidal profile mode, 2= external (cubic) profile mode
DIRINV	FLAG	AXIS	Invert direction output polarity if true
FILTERBLOCK	PTR	AXIS	Pointer to PID filter block (points to default filter block at startup)
PROFILEBLOCK	PTR	AXIS	Pointer to profile control block (points to default profile block at startup)
HOME	FLAG	AXIS	True when home operation is complete
MOTION	FLAG	AXIS	True when in motion (velocity <> 0)
SLEW	FLAG	AXIS	True when slew portion of profile has been reached
ERROR	UINT	AXIS	Bit 0 = Excessive position error Bit 1 = Excessive drive error
ERRORMASK	UINT	AXIS	ANDed with error, if <>0 the fault routine is run
RESET	FLAG	AXIS	Do hardware reset of Axis if cleared
STOPATHOME	FLAG	AXIS	Set DESVEL and GO to 0 on home event if STOPATHOME is set

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MISC MONITORING AND CONTROL FUNCTIONS			
LED	PTR	GLOBAL	Specify which parameter the debug LEDs follow. Note default is to point to SWREVISION
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
AVGVEL	INT	AXIS	(8.8) Average actual (counted) velocity
MAXNEGERR	INT	AXIS	Maximum negative deviation from profile. (Recorded counts)
MAXPOSERR	INT	AXIS	Maximum positive deviation from profile (Recorded counts)
EXPOSERR	UINT	AXIS	Excessive position error limit (counts)
DRIVEERROR	UNIT	AXIS	Excessive drive detection logic accumulator
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)
EVENTS	UINT	AXIS	Number of events (max number depends on free memory)
FIXUP	LONG	AXIS	Difference between profile end position and NEXTPOS (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	ULONG	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	Phasor velocity feed forward
KFF	UINT	AXIS	Friction feed forward / PWM deadzone term
KDFIL	UINT	AXIS	Derivative term filter coefficient
DRIVEPLUS	UINT	AXIS	Added to DRIVEERROR when Drive = MAXPWM
DRIVEMINUS	UINT	AXIS	Subtracted from DRIVEERROR when drive < MAXPWM
NEXTFILBLOCK	PTR	AXIS	Pointer to next filter block

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MOTION CONTROL PARAMETERS (Working Block)			
DESPOSF	LONG	AXIS	Fractional part of DESPOS (<1 count)
DESPOS	LONG	AXIS	Desired Position
VELOCITY	LONG	AXIS	Profile velocity (8.24 counts/sample)
ACCELF	INT	AXIS	Fractional part of 48 bit ACCEL (32.16)
ACCEL	LONG	AXIS	Profile Acceleration
JERK	LONG	AXIS	Profile Jerk (Delta ACCEL)
BREAKPOINT	LONG	AXIS	Breakpoint value for this block
NEXTPROBLOCK	PTR	AXIS	Pointer to next profile block
NEXTPOS	LONG	AXIS	Next position for internal profile generator
SLEWLIMIT	LONG	AXIS	Slew speed during move (counts/2^24/sample)

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
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)
DESVEL	LONG	AXIS	Velocity target for velocity follower (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
FLAGXOR	FLAG	GLOBAL	Used by Wait on flag token Note FLAGXOR is broken in Revisions 100..102, if needed, upgrade to 103 or >
FLAGAND	FLAG	GLOBAL	Used by wait on flag token

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
MULTIPHASE RELATED PARAMETERS			
PHASEAI	UINT	AXIS	Max A current for stepper/3 phase open loop current.
PHASEBI	UINT	AXIS	Max B current for stepper
OFFSETI	UINT	AXIS	Current offset for amplifier .
ENCODERCNT	UINT	AXIS	Encoder counts per revolution
ENCFACTOR	UINT	AXIS	(131072*Poles) / ENCODERCNT
SLIPK	INT	AXIS	Constant for generating slip frequency for AC motors
OPENLOOP	FLAG	AXIS	Set true for open loop mode (PID loop will use DESPOS instead of position error)
PHASELEAD	INT	AXIS	Lead angle, normally 256 or -256 (+ or -90 degrees)
PHASESHIFT	FLAG	AXIS	Sets lead angle to 0 or 90 degrees.
MOTORPHASE	UINT	AXIS	Primary encoder count mod ENCODECNT. Manipulated to align encoder count with drive.

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
CONTROLTYPE	UINT	GLOBAL	LSB = Motor phases
			(1 = brush, 2= 2 phase 3= 3 phase)
CPUTYPE	UINT	GLOBAL	DSP CPU type
GPHASEK	LONG	GLOBAL	Phase constant for user timer (GPHASEA=(GPHASEK*sample)
GPHASEA	LONG	GLOBAL	Global phase accumulator timer
GPHASEF	FLAG	GLOBAL	Flag set when GPHASEA overflows
PROCTIMER	UINT	GLOBAL	Process timer, reads SYSCLK*2 cycles for processing all enabled axis
SYSCLK	LONG	GLOBAL	*SOFT*DMC SYSCLK frequency in Hz
AXIS EVENT AND GENERAL PURPOSE RAM (6 words per event)			
EVENT1..EVENT24		AXIS	24 Events or 144 words of RAM
GLOBAL EVENT/MISC RAM			
GEVENT1 through GEVENT36		GLOBAL	36 Events or 216 words of RAM

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
INTERNAL HARDWARE			
CNTCTL	UINT	AXIS	Sets operational mode of quadrature input counter
CNTCLRP	UINT	AXIS	Any write clears primary encoder counter
CNTCLRS	UINT	AXIS	Any write clears secondary encoder 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. POSTSCALE MSB is symmetrical PWM mode enable
PWMGENA	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
PWMGENB	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
PWMGENC	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
DIRA	UINT	AXIS	Single bit in MSB controls direction output bit
DIRB	UINT	AXIS	Single bit in MSB controls direction output bit
ENA	UINT	AXIS	Controls Hbridge/Servo Amp enable bit

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
INTERRUPT REGISTERS			
IRQSETUPREG	UINT	GLOBAL	Controls IRQ channel and mask
IRQCAUSE	UINT	GLOBAL	Writes here set IRQ and OR data written to the IRQ Cause register. Bits 0..7 of IRQ cause register are reflected in Status register A
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

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	G/A	FUNCTION
HARDWARE INFO			
NAXIS	UINT	GLOBAL	Number of Axis for this hardware
HWREVISION	UINT	GLOBAL	Hardware revision number
ICDFIFOSZE	UINT	GLOBAL	Immediate Command/Data FIFO size
QCDFIFOSZE	UINT	GLOBAL	Queued Command/Data FIFO size
IRBFIFOSZE	UINT	GLOBAL	Immediate data Read-Rack FIFO size
QRBFIFOSZE	UINT	GLOBAL	Queued data Read-Back FIFO size

OPERATION

HARDWARE REGISTER DESCRIPTIONS

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 usually 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 10 bit register in the least significant part of the word:

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

Counter control register bits are defined as follows:

B9	CLRO	CLeaR Once, If set, causes the COI bit to be cleared if counter is cleared.
B8	DIR	Sets count DIRection, clear = normal, set = reversed
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. Note that the count clear function is deprecated in favor of CNTCLRX
B1	B	R/O — Reads back realtime B input
B0	A	R/O — Reads back realtime 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.*

OPERATION

INTERNAL HARDWARE

IRQ SETUP 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 Interrupt Cause 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)	

INTERRUPT CAUSE REGISTER.

The interrupt cause register (ICR) is an 8 bit register that can be written by the DSP. When the DSP writes to the ICR, the data written is Ored with the existing data in the ICR. This allows the DSP to set a desired bit in the ICR without disturbing the other bits. The ICR can be read by the host as the least significant 8 bits of Status register A.

IC7	IC6	IC5	IC4	IC3	IC2	IC1	IC0

A host interrupt is asserted whenever IMASK is true and any bit of the ICR is true. This means that separate events or queued commands can set separate bits in the ICR to cause an interrupt, and allow the host to determine the cause or causes of the interrupt.

HOST ACCESS TO ICR

The host can read the ICR as the least significant 8 bits of host Status Register A, and can clear bits in the ICR by writing a clear interrupt command to Host Status Register A. A clear interrupt command consist a of a 16 bit word with bit 14 and the ICR bits that you wish to clear set to one. For example, Writing 0x4003 to Host Status Register A would clear ICR bits 1 and 0.

OPERATION

INTERNAL HARDWARE

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 output. 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. In closed loop mode, they correspond directly with encoder counts. In open loop mode (2 phase and 3 phase BLDC only) the position is in units of microsteps. There are 1024 microsteps per motor pole. For example with a standard 200 step (50 pole) step motor, there are 51200 microsteps per revolution. 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.

VELOCITY IN RPM

The following equation can be used to calculate a DESVEL or SLEWLIMIT value for a desired speed in RPM:

SLEWLIMIT = (RPM * 2^24 * EncoderCountsPerRev) / (60 * SampleRate)

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.

SYMMETRICAL PWM MODE

In addition to the normal PWM mode, a symmetrical PWM mode is available. With normal PWM, the PWM leading edge is the same for all channels, while the trailing edge is determined by the PWM value. With symmetrical PWM, both the leading and trailing edge are determined by the PWM values such that the PWM pulse is centered on the total PWM period. This is often desirable for 3 Phase BLDC motor drive.

The symmetrical PWM mode is selected by writing a 1 to bit 15 of the POSTSCALE register. When symmetrical PWM is selected, the PWM rate will be ½ the value given in the equation above. The SAMPLERATE is not affected by the PWM mode.

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 is used, the maximum sample rate will be decreased by an amount depending on the total number of events. Time per event is approximately 700 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.

Occasional timeouts caused by running near the maximum sample rate do not cause long term timing errors because DSP will ‘catch up’ with missed samples (up to 16 in a row)

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 (DESPOS = *setpoint* position) for the PID loop to track, The PID loop then generates drive and direction signals to control the motor so that the actual motor position matches DESPOS. 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 three main modes of operation, trapezoidal mode velocity mode and external profile mode.

PROFILE TYPES

The PROFILE parameter determines the operation mode of the profile generator. There are three valid values for the PROFILE parameter: 0, 1, or 2. A value of 0 disables the profile generator. A value of 1 enables the trapezoidal profile mode . A value of 2 enables the external profile mode.

TRAPEZOIDAL PROFILE MODE

In trapezoidal profile 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 ACCEL, SLEWLIMIT and NEXTPOS parameters. Note that JERK is not used in trapezoidal profile mode.

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 >= SLEWLIMIT, 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 called a trapezoidal profile because the velocity profile is a trapezoid. The position profile is a parabolic (square) function of time**.**

CHANGING TRAPEZOIDAL PROFILE PARAMETERS ON-THE-FLY

The determination of the position to start ramp-down is done dynamically by calculating the required acceleration needed to stop at NEXTPOS, A = V^2/2D, When this value of A greater than or equal to the profiles programmed acceleration at the current velocity and position relative to the endpoint, ramp-down is begun. This dynamic calculation allows the host or event logic to change velocity, acceleration, or endpoint during a trapezoidal move and still reach the desired endpoint. Parameters can be changed freely as long as they are not changed during the ramp-down portion of the profile, and that ramp down to the endpoint is possible.

OPERATION

PROFILE GENERATOR

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

Velocity mode is a subset of trapezoidal profile mode. In velocity mode, the motion is controlled by the commanded velocity (DESVEL) parameter, and unlike position mode, motion parameters (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 must be 0), 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.

EXTERNAL PROFILE MODE

In this mode the host is can control the profile by manipulating the JERK and ACCEL, VELOCITY and DESPOS parameters. This allows creation of cubic position profiles. To use the External profile mode, the PROFILE parameter must be set to 2. Only the DDA (adder) portion of the profile generator is active. The DDA does the following calculations every sample:

IAccel <= IAccel + JERK

JERK 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 ACCELF as the least significant 16 bits.

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

IDespos <= IDespos + VELOCITY*256 (64 bit signed add)

IDespos is a 64 bit number consisting of DESPOS as the most significant 32 bits (the integer part) and DESPOSF as the least significant 32 bits (the fractional part).

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. Next the encoder counter is programed to clear its count when index is detected by setting the Clear_On_Index (COI) bit in the counter control register. In most cases it is also desirable to set the ClearOnce bit in the Counter Control Register. Counter Control Register bits are defined as follows:

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

Once the clear on index bit is set, host software should clear the HOME flag and start a slow velocity mode 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 normally until the desired edge of the index signal is detected. When the Index signal is detected,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, and the home FLAG set true. If the STOPATHOME flag is set, DESVEL will be set to 0, starting a controlled deceleration toward 0 velocity at the programmed acceleration rate. If the STOPATHOME flag is not set is not set, motion will continue, the host being responsible for changing the motion parameters.

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. To preset a encoder count without using index, The HOMEPOSP or HOMEPOSS parameter is set to the desired count, and the hardware encoder counter is cleared by writing the CL bit in the appropriate counter control register. The encoder count will be loaded with HOMEPOSP or HOMEPOSS at the next sample time. As above, the DESPOS parameter will be set to the HOMEPOSP value and the HOME flag set when the primary encoder is preset. The presetting the secondary encoder has no such side effects.

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:

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

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

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.

KF1 is the velocity feed forward term. It supplies an amount of drive proportional to the VELOCITY parameter. KF1 centers 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 small positive number A reasonable value for KF1 i ~1 to 100

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. Note that due to fixed point math limitations, KF2 will only work properly for acceleration values up to 2^23. This is normally not a real limitation as this represents and acceleration value of > 200,000 RPM per S^2 with a 500 line encoder an a 4KHz sample rate.

KF3 Is a velocity feed forward term for the drive phasor that is used in 2 and 3 phase motor drive configurations. Normally the drive phasor leads or lags the current motor position by a fixed angle. At high speeds this fixed angle may be too small due to the lag in motor current caused by motor inductance when rotational speeds and hence drive frequencies are high. The KF3 parameter increases the fixed lead or lag angle by an amount proportional to KF3 x Velocity. KF3 should be set to zero when *SOFT*DMC is used with amplifiers or H-bridges that implement Field Oriented Control of drive angle.

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.

OPERATION

PID LOOP

SECONDARY PARAMETERS

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.

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 lag in applying KD 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:

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

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.

MAXPWM While not strictly a tuning parameter, the MAXPWM parameter is part of the PID loop and affects its operation. MAXPWM limits the maximum PWM value applied to the motor drive. This can be used for torque limiting or keeping peak motor current within specified limits. Setting MAXPWM slightly below full scale is also required by some Hbridges to guarantee high side gate driver boostrap refresh. MAXPWM is an unsigned 16 bit number. Setting MAXPWM to 65535 will allow full scale drive. This is the MAXPWM default value. Setting MAXPWM to 16384 would result in a maximum PWM value of 25% of full scale.

MAXPWM is also used for the excessive drive fault detection.

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

ERRORMASK

When an fault occurs, a bit specific to that fault will be set in the ERROR parameter. The ERROR parameter is ANDed with the ERRORMASK parameter and if the result is not zero, a fault sequence will be generated. The default ERRORMASK is 0xFFFF, so all error types will cause a fault.

FAULT SEQUENCE

When an ummasked *SOFT*DMC error occurs, 1. The GO flag will be cleared, 2. The PID flag will be cleared, which disables the PID loop and sets the PWM value to zero, and finally 3. The ENA bit is cleared, allowing external hardware to detect the fault condition.

OPERATION

PID LOOP

EXCESSIVE POSITION ERROR

Excessive position error means that the absolute value of the PID loops error in counts is greater than the EXPOSERR 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 EXPOSERR value of zero will disable excessive position error checking. This saves some time, so if excessive position error detection is not needed, EXPOSERR should be set to zero.

When an excessive position fault occurs, bit 0 is set in the ERROR parameter.

EXCESSIVE DRIVE ERROR

Excessive drive means that the motor drive has reached its maximum allowed value (MAXPWM) for a programmable number of sample periods. If the motor drive equals MAXPWM, the DRIVERROR parameter is incremented by the DRIVEPLUS parameter. If the motor drive is less than the MAXPWM, the DRIVEERROR parameter is decremented by the DRIVEMINUS parameter. When DRIVEERROR is decremented, the count is bounded at zero so that it does not underflow. If the 16 bit DRIVEERROR parameter overflows, bit 1 is set in the ERROR parameter.

The separate DRIVEPLUS and DRIVEMINUS parameters allow the excessive drive detection time constant to be tailored to the motion control system so that the fault is detected in minimum time, but without generating false triggers. For example a DRIVEPLUS parameter of 656 and a DRIVEMINUS parameter of 65535 would require 100 sample periods where drive equaled MAXPWM (and no periods when drive was less than MAXPWM) to generate an error.

The excessive drive detection is valuable because it can detect fault conditions such as a bad (non counting) encoder and shut down the affected motor.

OPERATION

PID LOOP

RECOVERY FROM FAULT CONDITIONS

To recover from a fault sequence, the fault cause must be cleared, The ERROR parameter must be cleared, and then the motion system 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.

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 uses four environment variables to determine I/O type. These parameters are PROTOCOL, COMPORT, BAUDRATE and BAUDRATEMUL. Valid values for PROTOCOL are BUS, HEX, and LBP. Valid values for COMPORT are COM1 through COM99 Valid values for BAUDRATE are 9600,19200,38400,57600, or 115200. Valid values for BAUDRATEMUL are 1 through 16.

If PROTOCOL IS BUS, DMCTUNE will attempt to access a PCI or PC/104 interfaced motion controller. For use with serial or USB interfaced *SOFT*DMC controllers, the PROTOCOL parameter should be set to HEX for ASCII HEX serial interfaces and LBP for interfaces that use the Little Binary Protocol. COMPORT and BAUDRATE parameters need to be set appropriately for serially interfaced cards.

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
Delete	Erase trace screen
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

OPERATION

DMCTUNE COMMANDS

I/i	Import all parameters from file
L/l	Print all parameters to ASCII list file
T/t	Dump trace buffer to file
ALTX/Q/q	Exit program

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. KF1 in most needed with straight voltage output PWM amplifiers (with no current feedback). With straight PWM amplifiers, KF1 compensates for the motor back EMF, emulating current (torque) control. KF! Is adjusted by setting the acceleration to something that the system can follow and setting am moderate velocity, perhaps ½ full system speed. Then KF1 is slowly increased until the servo system response matches as closely as possible to the profile it is following.

OPERATION

TUNING

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 integral term is not used, the integral limit (KIL) should be set to 0. This has the advantage of bypassing the Integral part of the firmwares PID loop, speeding up the loop and allowing higher sampling rates.

OPERATION

MULTIPHASE MOTOR OPERATION

GENERAL

*SOFT*DMC supports 2 phase and 3 phase motors. A specific *SOFT*DMC configuration is required to support 2 phase or 3 phase motors and motor types cannot currently be mixed in a single *SOFT*DMC configuration. The *SOFT*DMC configuration type can be determined by reading the CONTROLTYPE parameter. A value of 1 indicates support for brush type motors, a value of 2 for 2 phase and 3 for 3 phase. Additional parameters need to be initialized to setup multiphase motors. Two phase step motors and 3 phase BLDC motors can also be operated in open loop mode (no encoder).

STEP MOTOR SETUP

The first thing to initilaize is the motor current. This is set with parameters PHASEAI and PHASEBI. These correspond to the A and B winding currents. When using the 7I32 Microstepping H-Bridge, a value of 65535 is equivalent to full scale current (either 1A or 3A depending on 7I32 jumpering). Normally the A and B currents are the same but can be adjusted individually to compensate for driver or motor winding imbalances. For example to set a motor current of 1.5A, you would select the 3A range of the 7I32 and then set PHASEAI and PHASEBI to 32768. If you need to reverse the motor direction this can be done VIA PHASESHIFT. PHASESHIFT is a flag that is normally false. Setting PHASESHIFT true is the equivalent or reversing one of the motor leads. Due to non-linearities of 7I32 drive current at low current levels, there is a dead zone in microstepping position on open loop mode. This can be compensated for by adding a small fixed offset current. This constant offset current is controlled by the OFFSETI parameter. The units of OFFSETI are the same as PHASEAI and PHASEBI. A suggested value of OFFSETI for the 7I32 is ~5% of the full scale current. The sum of OFFSETI and PHASEAI or PHASEBI must not exceed 65535.

OPENLOOP MODE

Stepper motors and 3 phase BLDC motors can be operated in open loop mode. This mode is the default for 2 phase motors. The OPENLOOP flag determines the operational mode. When it is true, the motor phase currents are determined by DESPOS. A DESPOS position change of 1024 results in a full phase rotation, which will rotate the motor 1 pole position. The encoder count is still available in open loop mode if desired to verify stepper position. PID must be enabled in open loop mode. Note that there is no position error or excessive drive checking in open loop mode.

CLOSED LOOP MODE

Closed loop mode is enabled by setting OPENLOOP false. In closed loop mode the motor drive currents and lead angle are determined by the magnitude and sign of the position error. There are some parameters that must be initialized properly for multi-phase close loop mode operation. The ENCODERCNT parameter is normally set to the number of encoder counts per revolution. For 3 phase synchronous motors, the ENCFACTOR is set to (131072*MotorPoles)/ENCODERCNT.

OPERATION

MULTIPHASE MOTOR OPERATION

CLOSED LOOP MODE

For step motors ENCFACTOR is set to RND((SINETABLESIZE*64*Motor Steps_Per_Rev)/ENCODERCNT, where Steps_Per_Rev is the number of full steps per motor revolution.

MECHANICAL PHASE ALIGNMENT

Each time *SOFT*DMC is started the encoder count must be aligned with the drive current phase angle. Here is one simple way of doing this alignment for two phase step motors:

First disable PID so that the PWM values can be set by the host., then turn on motor enable (ENA), next set the PWM value of PWMGENA to 0 and then PWMGENB to the same value as PHASEBI. This will set the motor phase angle to 0. Now you must wait some motor/load dependent time for the motion to settle. After the motion has settled, MOTORPHASE should be set to 0. After this is done ENCP and DESPOS can be set to any desired starting count value.

The procedure is slightly different for three phase synchronous motors. First disable PID so that the PWM values can be set by the host., then turn on motor enable (ENA), next set the PWMA, PWMB, PWMC to these values:

PWMA = SIN(0)*DRIVE*FS/2 + FS/2 = 32767

PWMB =SIN(120)*DRIVE*FS/2 + FS/2 = 61146 for 100% drive

PWMC =SIN(240)*DRIVE*FS/2 + FS/2 = 4389 for 100% drive

This will set the motor phase angle to 0. Now you must wait some motor/load dependent time for the motion to settle. After the motion has settled, MOTORPHASE should be set to 0. After this is done ENCP and DESPOS can be set to any desired starting count value. Note that the above PWM values assume locked anti-phase Hbridge operation where a PWM value of ½ full scale = 32767 = 0 drive. For a simple Locked anti-phase driver like our 7I39, DRIVE should be limited to .1 or less, since there is no working feedback at this point and current will be the same as locked rotor currents.

ELECTRICAL PHASE ALIGNMENT

There are several conditions that are required for correct closed loop operation of multiphase moors. The first thing is to wire the encoder so that the encoder counts in desired direction. Next, the PHASELEAD may need to be inverted to be able to close the feedback loop. For example the default value of PHASELEAD is 256 for 2 phase configuration. In order to reverse the drive, PHASELEAD can be changed to -256. Once the feedback loop is closed the PHASESHIFT flag may need to set if the drive is "coggy". Changing PHASESHIFT may require re-inverting PHASELEAD.

OPERATION

EVENT LOGIC

GENERAL

*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 an event can be starting a motion profile, aborting a motion profile, loading new acceleration or velocity parameters, Motion register block pointer updates, PID filter block pointer updates, interrupt generation, I/O bit manipulation, do gearing with a 32 bit ratio between axis. etc etc. Events can also be used to change the way the Profile generator or PID loop operate.

WHAT EVENTS DO

Events can perform logical operations, 16 bit and 32 bit addition, 16 and 32 bit subtraction, 16 and 32 bit multiplication, 32 bit division and 32 bit square root.. Subtraction events can be used to perform 16 or 32 bit compares, and the addition events can be used to perform 16 or 32 bit copies.

All events *can be* unconditional or conditionally executed. Normally, unconditional events are used for operations that must occur every sample period, or for test or compare operations. Unconditional events set two flags (Zero and Carry) that can be used by subsequent conditional events. Conditionally executed events are used for operations that should only happen when specific conditions exist. Conditionally executed events do not change the state of the flags, allowing multiple conditional events to use the flags generated by a single unconditional operation.

Conditional events are only executed if specified condition is matched. Conditional events can be Level or Delta triggered. Level triggering means the conditional operation is always performed (once per sample) when the desired condition is matched. Delta triggering means that the if the condition was false in the previous sample period and true in the current period, the operation will be performed (This is sometimes referred to as edge triggered).

Delta triggering is useful for operations that should only happen once, when a particular condition *becomes* true, for example, generating an interrupt when a limit switch is actuated. Level triggering is useful for operations that should continue as long as the tested condition is true, for example turning on an output bit to control a paint solenoid only when the velocity is greater than a desired setpoint velocity.

OPERATION

EVENT LOGIC

LOGICAL EVENTS

Logical events perform 16 bit wide logical operations on a chosen parameter. Logical events have two parameter pointers, a source and a destination. Logical events consist of a block of six 16 bit words:

EventOpcode	Opcode for the logical event
EventSRC1	Source address pointer
EventXOR	XOR mask
EventAND	AND mask
EventOR	OR mask
EventDest	Destination address pointer

Logical operations proceed as follows:

1. If the operation is conditional, check the flags for the desired condition. If the condition is not met got step 8 otherwise continue at step 2

2. If the event is delta triggered, check the history bit in the event, if it is true, goto step 8, otherwise continue at step 3.

3. The parameter is fetched via the EventSRC1 pointer to TEMP

4. TEMP is XORed with the EventXOR parameter

5. TEMP is ANDed with then EventAND parameter

6. TEMP is Ored with the EventOR parameter

7. TEMP is written back to the parameter pointed to by EventDest

8. If the tested condition was true, set the history bit in the event, otherwise clear it.

Logical events can be used to set, clear or toggle bits in control registers or I/O ports.

9. If the event was unconditional, set the Zero flag based on the value in TEMP

OPERATION

EVENT LOGIC

ARITHMETIC EVENTS

Arithmetic events perform an arithmetic operation. All parameters in arithmetic events are indirect, that is the event block contains pointers to the source1, source2 and destination of the operation. Arithmetic event blocks consist or six 16 bit words:

EventOpcode	Event opcode
EventSrc1	Source1 parameter pointer
EventSrc2	Source2 parameter pointer or copy word count
EventFree1	Unused, can be use to store 16 bit data
EventFree2	Unused, can be use to store 16 bit data
EventDest	Destination parameter pointer

Arithmetic operations proceed as follows:

1. If the operation is conditional, check the flags for the desired condition. If the condition is not met got step 7 otherwise continue at step 2

2. If the event is delta triggered, check the history bit in the event, if it is true, goto step 8, otherwise continue at step 3.

3. The Source1 parameter is fetched via the EventSrc1 pointer

4. The Source2 parameter is fetched via the EventSrc2 pointer

5. The arithmetic operation is performed - for subtracts, Source2 is subtracted from Source1.

6. The result of the arithmetic operation is written to the location pointed to by the EventDest pointer.

7. If the tested condition was true, set the history bit in the event, otherwise clear it. .

8. If the event was unconditional, set the Zero and Carry flags based on the result of the arithmetic operation. Multiply operations do not change the flags.

OPERATION

EVENT LOGIC

ARITHMETIC EVENTS

Arithmetic events can be used for 16 or 32 bit compares (by using subtract), gearing (One event for multiply and one for 32 bit add), Allowing hand control of motion via the secondary encoder inputs, event counting, and adding new features to the Profile generator or PID loop.

For 32 bit Arithmetic events, the source and destination pointers point to the least significant word of the data, the most significant words being at pointer +1 (This is the standard word order for all *SOFT*DMC parameters).

COPY EVENTS

Copy events allow from 1 to 1024 words to be block copied from the EventSrc1 location to the EventDest location. EventSrc2 is used as the count of words to be copied. Words are copied in low to high order, so for example a copy of 3 words would proceed as follows:

* EventSrc1 -->* EventDest

*EventSrc1+1 --> *EventDest+1

*EventSrc1+2 --> *EventDest+2

USING CONDITIONAL EVENTS

Conditional events require a preceding unconditional event to set the flags. The most common unconditional events used to set the flags are logical and subtract events. Logical events only set the Zero flag, the carry flag is indeterminate after an unconditional logical event. Unconditional Add and subtract events update both the Zero and Carry flags. Flags are indeterminate after unconditional Multiply events. If an unconditional event is used to set flags and the result is not needed, the EventDest pointer should point to NullLoc.

The conditional event Opcodes contain mask and complement bits that operate on the flag bits to determine whether the conditional event should be executed.

OPERATION

EVENT LOGIC

EVENT OPCODES

The header files supplied with *SOFT*DMC include opcodes for common unconditional and conditional events.

UNCONDITIONAL EVENTS

EventAdd	Unconditional 16 bit add
EventAdd32	Unconditional 32 bit add
EventSub	Unconditional 16 bit subtract
EventSub32	Unconditional 32 bit subtract
EventMul16x16to16	Unconditional 16x16 bit multiply - lo 16 bits of result stored
EventMult32x32Lo	Unconditional 32x32 bit multiply - lo 32 bits of result stored
EventMult32x32Hi	Unconditional 32x32 bit multiply - hi 32 bits of result stored
EventMult32x32to64	Unconditional 32x32 bit multiply - all 64 bits of result stored
EventDiv	Unconditional 64/32 bit divide - 32 bit result
EventSqrRoot	Unconditional 64 bit square root - 32 bit result
EventLogical	Unconditional 16 bit Logical
EventCopy	Unconditional copy

OPERATION

EVENT LOGIC

CONDITIONAL EVENTS

EventAddIf	Conditional 16 bit add
EventAdd32If	Conditional 32 bit add
EventSubIf	Conditional 16 bit subtract
EventSub32If	Conditional 32 bit subtract
EventMul16x16to16If	Conditional 16x16 bit multiply - lo 16 bits of result stored
EventMult32x32LoIf	Conditional 32x32 bit multiply - lo 32 bits of result stored
EventMult32x32HiIf	Conditional 32x32 bit multiply - hi 32 bits of result stored
EventMult32x32to64If	Conditional 32x32 bit multiply - all 64 bits of result stored
EventDivIf	Conditional 64/32 bit divide - 32 bit result
EventSqrRootIf	Conditional 64 bit square root - 32 bit result
EventLogicalIf	Conditional 16 bit Logical
EventCopyIf	Conditional copy

OPERATION

EVENT LOGIC

DELTA TRIGGERED CONDITIONAL EVENTS

EventAddIfDel	DelConditional 16 bit add
EventAdd32IfDel	DelConditional 32 bit add
EventSubIfDel	DelConditional 16 bit subtract
EventSub32IfDel	DelConditional 32 bit subtract
EventMul16x16to16IfDel	DelConditional 16x16 bit multiply - lo 16 bits of result stored
EventMult32x32LoIfDel	DelConditional 32x32 bit multiply - lo 32 bits of result stored
EventMult32x32HiIfDel	DelConditional 32x32 bit multiply - hi 32 bits of result stored
EventMult32x32to64IfDel	DelConditional 32x32 bit multiply - all 64 bits of result stored
EventDivIfDel	DelConditional 64/32 bit divide - 32 bit result
EventSqrRootIfDel	DelConditional 64 bit square root - 32 bit result
EventLogicalIfDel	DelConditional 16 bit Logical
EventCopyIfDel	DelConditional copy

OPERATION

EVENT LOGIC

CONDITION MODIFIERS

The opcodes of conditional events are Ored with specific flag modifiers to select the desired condition:

EventZero	if result of logical or arithmetic operation was Zero
EventNotZero	if result of logical or arithmetic operation was not Zero
EventCarry	if the add or subtract caused a carry
EventNotCarry	if the add or subtract did not cause a carry

The following composite constants assume that the unconditional event that set the flags was a subtract:

EventLT	if @EventSRC1 < @EventSRC2
EventGTEQ	if @EventSRC1 is >= @EventSRC2
EventLTEQ	if @EventSRC1 is <= @EventSRC2
EventEQ	if @EventSRC1 is = @EventSRC2
EventNEQ	if @EventSRC1 is <> @EventSRC2

Note: EventGT can be created by using the EventLTEQ and reversing the parameters for the subtract.

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 six 16 bit words. 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 = ~700 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.

MULTI-AXIS GEARING WITH EVENTS

The multiply event and the subtract event can be combined to gear one axis to another. If gearing is started at location 0, no subtract is needed unless a fixed offset is required. Normally for gearing, the fastest axis is used as the master and the slower axis is the slave. Assuming both master and slave start at 0, gearing is set up as follows:

An unconditional multiply event (EventMult32X32Hi) is created that takes the masters DESPOS (*EventSrc1), multiplies it it by the gearing ratio (*EventSRC2/2^32) and stores the result in a free global 32 bit location (*EventDest). Then the FOLLOW parameter for the slave axis is set to point to *EventDest. This will give gearing ratios from 0 to almost 1 with 32 bit accuracy.

For integer gear ratios that require absolute accuracy, both master and slave axis can be scaled by different ratios. For example for an absolute 5/7 ratio, The master axis could be geared with the ratio (**7***K) to its own DESPOS parameter, and the slave axis could be geared to (**5***K). K is calculated so that the master axis self-gearing ratio is close to one: Trunc((2^32-1)/MasterRatio) or 4294967295 / 7 in the example above.

To start gearing from arbitrary positions, a subtract event is needed along with the multiply event so that the geared positions start out matching their ungeared positions. *If this is not done, a large uncontrolled move will be generated when the FOLLOW pointer is changed.*

OPERATION

MOTION PARAMETER BLOCKS

GENERAL

The main motion controlling registers are accessed via a pointer and therefore it is possible to allocate multiple motion control blocks, and change between them by changing a single pointer.. The standard implementations of *SOFT*DMC can have room for up to 8 motion control blocks in addition to the default block. Make sure that you do not allocate overlapping blocks! Register blocks are allocated from high memory down to low memory, to allow a mix of events and register blocks to co-exist. The default motion control block is allocated in axis memory and does not use any event/user RAM.

Note: if you do not need to quickly change all motion parameters for an axis, the indirect block nature of the motion parameters can be ignored.

Each motion control block contains the following parameters:

MOTION CONTROL PARAMETERS

DESPOSF	LONG	AXIS	Fractional part of DESPOS (<1 count)
DESPOS	LONG	AXIS	Desired Position
VELOCITY	LONG	AXIS	Profile velocity (8.24 counts/sample)
ACCELF	INT	AXIS	Fractional part of 48 bit ACCEL (32.16)
ACCEL	LONG	AXIS	Profile Acceleration
JERK	LONG	AXIS	Profile Jerk (Delta ACCEL)
BREAKPOINT	LONG	AXIS	Breakpoint value for this block
NEXTPROBLOCK	PTR	AXIS	Pointer to next profile block (or user variable)
NEXTPOS	LONG	AXIS	Next position for internal profile generator
SLEWLIMIT	LONG	AXIS	Slew speed during move (counts/2^24/sample)

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.

OPERATION

FILTER PARAMETER BLOCKS

GENERAL

Like the motion control registers, The filter parameters are accessed via a pointer and therefore it is possible to allocate multiple blocks of filter parameters, and change between them by changing a single pointer.. The standard implementations of *SOFT*DMC can have room for up to 9 filter parameter blocks in addition to the default filter block.

Note: if you do not need to quickly change all filter parameters for an axis, the indirect block nature of the filter parameters can be ignored.

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	Phasor velocity Feed forward
KFF	UINT	AXIS	Friction feed forward term
KDFIL	UINT	AXIS	Derivative term filter coefficient
DRIVEPLUS	UINT	AXIS	Amount added to DRIVEERROR
DRIVEMINUS	UINT	AXIS	Amount subtracted from DRIVERROR
NEXFILBLOCK	PTR	AXIS	Pointer to next filter block or user variable.

OPERATION

USER PARAMETERS

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

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.

OPERATION

DEMONSTRATION SOFTWARE

RP, WP AND WF and EVENT COMMAND LINE UTILITIES

Four simple command line utilities are provided for manually reading and writing *SOFT*DMC parameters, installing FIFO wait tokens, and installing events These utilities are RP, WP, WF, and EVENT. RP reads a parameter, WP writes a parameter, WF installs a Wait token in the FIFO and EVENT installs an event.

ENVIRONMENT VARIABLES

RP, WP, WF, and EVENT rely on four environment variables to determine I/O type. These parameters are PROTOCOL, COMPORT, BAUDRATE, and BAUDRATEMUL Valid values for PROTOCOL are BUS, HEX, and LBP. Valid values for COMPORT are COM1 through COM99 Valid values for BAUDRATE are 9600,19200,38400,57600, or 115200. Valid values for BAUDRATEMUL are 1 through 16.

If PROTOCOL is BUS, the utilities will attempt to access a PCI or PC/104 interfaced motion controller. For use with serial or USB interfaced *SOFT*DMC controllers, the PROTOCOL parameter should be set to HEX for ASCII HEX serial interfaces and LBP for interfaces that use the Little Binary Protocol. COMPORT and BAUDRATE parameters need to be set appropriately for serially interfaced cards. For example to access a serial card with HEX ASCII interface and a standard speed serial card:

SET PROTOCOL=HEX

SET BAUDRATE=115200

SET COMPORT=COM2

SET BAUDRATEMUL=1

Would cause the RP, WP, or WF utility to use the serial interface on COM2 at 115200 baud to communicate to *SOFT*DMC, while:

SET PROTOCOL=BUS

Would cause the utilities to communicate with the *SOFT*DMC hardware on the host computers PC/104 or PCI bus.

OPERATION

DEMONSTRATION SOFTWARE

RP

The RP utility reads a parameter from *SOFT*DMC. It uses symbolic names for the parameters so numeric constants do not need to be memorized, for example

RP DESPOS 3

Would read the DESPOS parameter for Axis 3. If the axis parameter is omitted, RP reads the parameter from Axis 0, so

RP ENCP

Would read the ENCP parameter from Axis 0. Also note that the axis value is don't care for global parameters, so the axis value can be omitted for all global parameter reads.

The RP utility uses the IFIFO by default but can optionally use the QFIFO for communicating with *SOFT*DMC. This is done by putting a 'Q' on the command line:

RP MAXPWM 6 Q

RP GO Q

RP can print parameter values in Hexdecimal if desired by putting an 'H' on the command line after the parameter name:

RP PORTA H

RP knows the parameter size (16 or 32 bit) and parameter type (signed or unsigned)from its symbol table so it reads and prints parameters in the proper format.

OPERATION

DEMONSTRATION SOFTWARE

RP

RP can also read and print a parameter continuously if desired. This is done by putting a 'R' on the command line after the parameter name:

RP ENCP R

All the command modifiers can be combined in a single command if desired:

RP PORTA 2 R H Q

WP

The WP utility writes a parameter to *SOFT*DMC. It uses symbolic names for the parameters so numeric constants do not need to be memorized, for example

WP KP 300 5

Would set the KP parameter for Axis 5. If the axis parameter is omitted, WP writes the parameter to Axis 0, so

WP GO 65535

Would set GO for Axis 0.

The WP utility has the same QFIFO option as RP

WP NEXTPOS 1234 3 Q

Would set the NEXTPOS parameter for axis 3 to 1234, using the QFIFO. WP also has the H (Hex) option to allow the use of Hexadecimal parameter values:

WP PORTA 0FFE H

Would write the Hexdecimal value 0xFFE to I/O port A.

WP knows the parameter size (16 or 32 bit) and parameter type (signed or unsigned) from its symbol table so it writes parameters in the proper format.

OPERATION

DEMONSTRATION SOFTWARE

WF

The WF utility writes a WaitOn FLAG to the specified FIFO. For example:

WF GO 2

Would write a Wait-On-(Axis 2 GO) flag in the IFIFO.

WF MOTION 3 Q

Would write a Wait-On(Axis 3 MOTION) flag in the QFIFO. Note that the behavior of the Wait FLAGS depends on the per axis parameters FLAGXOR and FLAGAND.

WF PHASEA+1 1

Would write a Wait-On(Axis 1 PHASEA) flag in the IFIFO. Note that an offset can be appended to the parameter name (with no spaces). We are able to access the high word of PHASEA by using an offset of 1 in this example

The following example sequence does a axis 2 move, waits for the move to complete, waits 100000 sample times, and then does another move:

WP FLAGXOR FFFF H 2

(FLAGXOR is set to FFFF: we are waiting for the watched parameter to become 0)

WP FLAGAND 8000 H 2

(8000 hex works for sensing flags and also looking for MSB of PHASEA)

WP NEXTPOS 50000 2

(we are going to position 50000)

WP GO -1 2

(start a motion)

WF GO 2

(wait for GO to clear when motion is done)

OPERATION

DEMONSTRATION SOFTWARE

WF
WP PHASEA 8001869F 2

(99,999 more than 80000000 H)

WP PHASEK -1 2

(Count down)

WF PHASEA+1 2

(wait for PHASEA MSB to become 0 = 100000 counts = 100000 sample times)

WP NEXTPOS 0 2

(we are going to 0 next)

WP GO -1 2

(start move towards 0)

WF GO 2

(wait for move to complete)

Note that running the previous sequence of commands would place the commands in *SOFT*DMC's FIFO, after which the host is not involved in the motions or delay, that is the command line example programs are asynchronous and do not wait for *SOFT*DMC other than to pause processing if the QCDFIFO or ICDFIFO is half full.

OPERATION

DEMONSTRATION SOFTWARE

EVENT

The EVENT utility allows command line installation of events. EVENT uses symbolic names for source and destination addresses and event op codes. Because logical and arithmetic events have different syntax, the command line parameters are parsed differently depending on event type. For logical events, the command line syntax is:

EVENT OPCODE COND SRC XOR AND OR DEST NUMBER TYPE AXIS

Where OPCODE is a symbolic event opcode, COND is a symbolic opcode modifier if the OPCODE is conditional, SRC is a symbolic parameter name, XOR, AND, OR are 16 bit Hexadecimal values, DEST is a symbolic parameter name, NUMBER is the event number, TYPE is G for Global events and A for Axis events, and AXIS is the axis number for axis events. For example the following sequence:

WP EVENTS 0 0

(Disable events when we are loading them)

EVENT EventLogical ERROR 0000 FFFF 0000 NULL 1 A 0

(Unconditional logical event to set the flags based on ERROR)

EVENT EventLogicalIfDel EventNotZero NULL 0000 0000 0001 IRQCause 2 A 0

(Conditional logical event to set bit 0 of IRQCause register if ERROR is true)

WP EVENTS 2 0

(Enable both events when we are done loading them)

Would install 2 events, the first event is an unconditional logical event that monitors ERROR and sets the Event flags accordingly, and the second event, a conditional event that writes data to the IRQCause register if the Event flags indicate a non zero result, that is, if ERROR is not zero.

OPERATION

DEMONSTRATION SOFTWARE

EVENT

The following set of events would set the LSb of PORTA whenever (axis 0) DESPOS was greater than MYBREAK :

```
WP EVENTS 0 0

( Disable events when we are loading them )

EVENT EventSub32 DESPOS MYBREAK 0000 0000 NULL 1 A 0

( Unconditional subtract event to set the flags)

EVENT EventLogicalIf  EventGT PORTA 0000 FFFE 0001 PORTA 2 A 0

( Conditional logical event to set I/O bit if DESPOS > MYBREAK )

EVENT EventLogicalIf  EventLTEQ PORTA 0000 FFFE 0000 PORTA 3 A 0

( Conditional logical event to clear I/O bit if DESPOS <= MYBREAK )

WP EVENTS 3 0

( Enable all three events when we are done loading them )
```

REFERENCE INFORMATION

PINOUTS

Most Mesa FPGA cards use 50 pin headers as I/O connectors. The following tables show the pin order for the more common *SOFT*DMC configuration when used with Mesa FPGA cards and Mesa interface/driver cards.

When a pin function has for example a (0,4) suffix, this means that the on the first 50 pin connector, the pin would connect to axis 0 and on a second connector, the same pin would connect to axis 4.

BRUSH MOTOR PINOUT

This pinout is used for the 7I30 and 7I40 brush motor drivers, plus the 7I33 servo amplifier interface card. Four axis are supported by each 50 pin connector. Note that the 7I40 is only a 2 axis driver card but the 7I40 can select whether it connects to motor 0,1 signals or motor 2,3 signals, allowing two 7I40s to be daisy chained on a single 50 pin cable.

FUNCTION	DIR	HEADER 50 PIN
B(1,5)	TO FPGA	1
A(1,5)	TO FPGA	3
B(0,4)	TO FPGA	5
A(0,4)	TO FPGA	7
IDX(1,5)	TO FPGA	9
IDX(0,4)	TO FPGA	11
PWM(1,5)	FROM FPGA	13
PWM(0,4)	FROM FPGA	15
DIR(1,5)	FROM FPGA	17
DIR(0,4)	FROM FPGA	19
ENA(1,5)	FROM FPGA	21
ENA(0,4)	FROM FPGA	23

REFERENCE INFORMATION

PINOUTS

BRUSH MOTOR PINOUT (continued)

FUNCTION	DIR	HEADER 50 PIN
B(3,7))	TO FPGA	25
A(3,7)	TO FPGA	27
B(2,6)	TO FPGA	29
A(2,6)	TO FPGA	31
IDX(3,7)	TO FPGA	33
IDX(2,6)	TO FPGA	35
PWM(3,7)	FROM FPGA	37
PWM(2,6)	FROM FPGA	39
DIR(3,7)	FROM FPGA	41
DIR(2,6)	FROM FPGA	43
ENA(3,7)	FROM FPGA	45
ENA(2,6)	FROM FPGA	47

REFERENCE INFORMATION

PINOUTS

7I32 STEP MOTOR PINOUT

This pinout is used by the 7I32 microstepping drive. The 7I32 uses sine and cosine PWM drive from the controller that set the step motor drive current. The 7I32 pinout supports 2 step motors per 50 pin connector.

FUNCTION	DIR	HEADER 50 PIN
CCCOS(0,2,4,6)	FROM FPGA	1
CCSIN(0,2,4,6)	FROM FPGA	3
B(0,2,4,6)	TO FPGA	5
A(0,2,4,6)	TO FPGA	7
ALTIDX(0,2,4,6)	FROM FPGA	9
IDX(0,2,4,6)	FROM FPGA	11
PWMCOS(0,2,4,6)	FROM FPGA	13
PWMSIN(0,2,4,6)	FROM FPGA	15
DIRCOS(0,2,4,6)	FROM FPGA	17
DIRSIN(0,2,4,6)	FROM FPGA	19
ENACOS(0,2,4,6)	FROM FPGA	21
ENASIN(0,2,4,6)	FROM FPGA	23

REFERENCE INFORMATION

PINOUTS

7I32 STEP MOTOR PINOUT (Continued)

FUNCTION	DIR	HEADER 50 PIN
CCCOS(1,3,5,7)	FROM FPGA	25
CCSIN(1,3,5,7)	FROM FPGA	27
B(1,3,5,7)	TO FPGA	29
A(1,3,5,7)	TO FPGA	31
ALTIDX(1,3,5,7)	FROM FPGA	33
IDX(1,3,5,7)	FROM FPGA	35
PWMCOS(1,3,5,7)	FROM FPGA	37
PWMSIN(1,3,5,7)	FROM FPGA	39
DIRCOS(1,3,5,7)	FROM FPGA	41
DIRSIN(1,3,5,7)	FROM FPGA	43
ENACOS(1,3,5,7)	FROM FPGA	45
ENASIN(1,3,5,7)	FROM FPGA	47

REFERENCE INFORMATION

PINOUTS

THREE PHASE PINOUT

This pinout is used by the 7I39 three phase Hbridge. Two axis are supported per 50 pin connector. If I/O port A/B is not available on the FPGA configuration, the secondary encoder inputs are available on the HALLA through HALLC inputs.

FUNCTION	DIR	HEADER 50 PIN
A(0,2,4,6)	TO FPGA	1
B(0,2,4,6)	TO FPGA	3
IDX(0,2,4,6)	TO FPGA	5
HALLA/SA(0,2,4,6)	TO FPGA	7
HALLB/SB(0,2,4,6)	TO FPGA	9
HALLC/SIDX(0,2,4,6)	TO FPGA	11
SENSEA(0,2,4,6)	TO FPGA	13
SENSEB(0,2,4,6)	TO FPGA	15
ENA(0,2,4,6)	FROM FPGA	17
PWMA(0,2,4,6)	FROM FPGA	19
PWMB(0,2,4,6)	FROM FPGA	21
PWMC(0,2,4,6)	FROM FPGA	23

REFERENCE INFORMATION

PINOUTS

THREE PHASE PINOUT (Continued)

FUNCTION	DIR	HEADER 50 PIN
A(1,3,5,7)	TO FPGA	25
B(1,3,5,7)	TO FPGA	27
IDX(1,3,5,7)	TO FPGA	29
HALLA/SA(1,3,5,7)	TO FPGA	31
HALLB/SB(1,3,5,7)	TO FPGA	33
HALLC/SIDX(1,3,5,7)	TO FPGA	35
SENSEA(1,3,5,7)	TO FPGA	37
SENSEB(1,3,5,7)	TO FPGA	39
ENA(1,3,5,7)	FROM FPGA	41
PWMA(1,3,5,7)	FROM FPGA	43
PWMB(1,3,5,7)	FROM FPGA	45
PWMC(1,3,5,7)	FROM FPGA	47

REFERENCE INFORMATION

PINOUTS

IO PORT PINOUT

Some *SOFT*DMC configurations support I/O ports. *SOFT*DMC I/O ports are 12 bit ports with individual (per bit) direction control. In addition, secondary encoder inputs usually share the I/O A/B port pins. This is a typical IO port pinout, specific *SOFT*DMC configurations may vary. Note that Ports C/D,E/F, G/H are similar.

FUNCTION	HEADER 50 PIN
PORTB11/SENCB1	1
PORTB10/SENCA1	3
PORTB9/SENCB0	5
PORTB8/SENCA0	7
PORTB7/SIDX1	9
PORTB6/SIDX0	11
PORTB5	13
PORTB4	15
PORTB3	17
PORTB2	19
PORTB1	21
PORTB0	23

REFERENCE INFORMATION

PINOUTS

IO PORT PINOUT (Continued)

FUNCTION	HEADER 50 PIN
PORTA11/SENCB3	25
PORTA10/SENCA3	27
PORTA9/SENCB2	29
PORTA8/SENCA2	31
PORTA7/SIDX3	33
PORTA6/SIDX2	35
PORTA5	37
PORTA4	39
PORTA3	41
PORTA2	43
PORTA1	45
PORTA0	47