3C20

100 W SMART SERVO MOTION CONTROLLER

1.5

This page intentionally not blank

Table of Contents

Table of Contents

Table of Contents

Table of Contents

3C20

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.

GENERAL DESCRIPTION

The 3C20 digital motion controller is a DSP based single-axis DC servo motor controller/amplifier intended to control small (up to 100 W) brush motors with quadrature encoders.

The 3C20 design has a 30 MIPS 16 bit DSP coupled a 100 W Hbridge on a single 2"x2" card.

Position, velocity and acceleration and jerk parameters are all have 32 bit or greater precision. Profile generator has simple built-in trapezoidal profile capability . Elaborate time based S-curve profiles are easily generated with a sequence of jerk values and delay tokens place in one of the FIFOs.

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 (~20 KHz) support small and fast drive systems. Software selectable current (torque) or voltage mode motor drive.

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.

The 3C20 host interface is a RS-422 or RS-485 serial port that supports baud rates up to 1.84 MBaud. Efficient dual FIFO buffered 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.

In addition to the motor control signals, two general purpose inputs, 2 high current (500 mA) outputs and three 10 bit 5V full scale analog inputs are provided.

HARDWARE CONFIGURATION

GENERAL

The 3C20 has just two hardware settable options, setup mode and termination enable. Both options are selectable by moving a shorting jumper. The directions assume that the 3C20 is in normal upright position. This position is with the RJ-45 host connector on the left side and the Encoder/I-O connector on the right.

SETUP MODE JUMPER

W1 is the setup mode jumper. As shipped, 3C20s have setup mode disabled, Setup mode is enabled by moving W1 to the right hand position. When setup mode is enabled, the EEPROM contents are ignored and the following defaults are used:

ADDRESS	00
BAUDRATE	115,200
SERIALMODE	RS-422

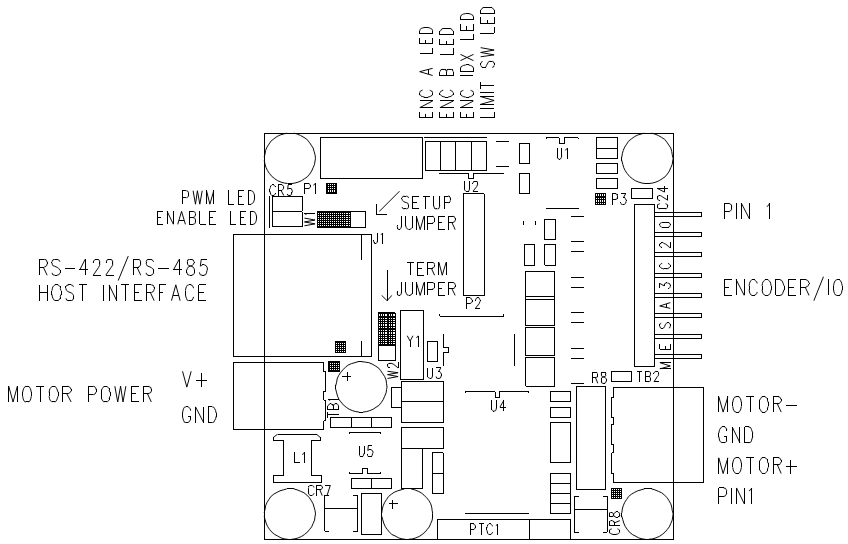
Setup mode allows the 3C20 to be put into a known state to simplify initialization and to recover if the EEPROM is programmed incorrectly.

TERMINATION JUMPER

The RS-422/RS-485 interface on the 3C20 can be terminated with 100 Ohms. Termination is suggested when using baud rates above 460K baud or on long lines. If multiple 3C20s share a single long line, only one 3C20 at the far end of the line should be terminated. W2 controls the termination. When W2 is on the "up" position, termination is enabled.

HARDWARE CONFIGURATION

CONNECTOR LOCATIONS AND DEFAULT JUMPER POSITIONS



CONNECTORS

TB1 MOTOR POWER CONNECTOR

The 2 pin motor power connector supplies power for the servo motor and the 5V regulator that runs the 3C20s DSP. TB1 is a 2 pin 3.5MM inline connector which can be used with crimp type mating connectors or pluggable screw terminals. Mating screw terminal is Weidmüller PN 1638780000. Motor power can be up to 32V for the 3C20A and up to 50V for the 3C20B.

TB1 PINOUT:

PIN	FUNCTION
1	+ MOTOR POWER
2	GND

ENCODER-I/O CONNECTOR

P3 is the encoder-I/O connector. P3 is a 16 pin .1" pitch dual inline header. Suggested mating connector is Amp 746285-3. P3 can supply up to 250 mA of 5V power to the encoder.

P3 PINOUT:

PIN	FUNCTION	PIN	FUNCTION
1	ENCA	2	GND
3	ENCB	4	5V Enc Power (from 3C20)
5	IDX	6	5V Enc Power (from3C20)
7	LIMIT	8	GND
9	AIN0	10	AIN1
11	AIN2	12	/EXTRST
13	IN0	14	IN1
15	OUT0	16	OUT1

CONNECTORS

TB2 MOTOR CONNECTOR

Three pin connectorTB2 is the motor connector TB2 is a 3 pin 3.5MM inline connector which can be used with crimp type mating connectors or pluggable screw terminals. Mating screw terminal is Weidmüller PN 1638790000.

TB2 PINOUT:

PIN	FUNCTION
1	MOTOR+
2	GND
3	MOTOR-

SERIAL HOST INTERFACE

The 3C20 connects to the host via a serial RS-422 or RS-485 interface. This serial connection uses a RJ-45 type jack J1.

J1 PINOUT:

PIN	FUNCTION	PIN	FUNCTION
1	RX-	2	RX+
3	TX-	4	GND
5	GND	6	TX+
7	+5V	8	+5V

Notes:

1. For RS-485 (2 wire) type applications, RX- must be tied to TX- and RX+ must be tied to TX+.

2. Pin names are 3C20 relative, RX pair is input to 3C20 and TX pair is output from 3C20. When connecting to host computer using RS-422 (4 wire) mode, the 3C20's RX pair must connect to the hosts TX pair, and the 3C20's TX pair must connect to the hosts RX pair.

HOST INTERFACE

GENERAL

The 3C20 communicates to the host through a RS-422 or RS-485 serial interface. The 3C20 supports baud rate from 9600 to 1.8432 Mbps. All host communication is done via an ASCII command set with hexadecimal parameters.

SERIAL COMMAND SET

The host serial interface supports several commands. Basic command include reading and writing the Immediate and Queued FIFOs, reading FIFO status and commands for selecting individual 3C20s that share a single serial interface. Most commands consist of two letters and may have hexadecimal parameters. All returned data is hexadecimal or ASCII strings. Hexadecimal data must uppercase, and all commands are case sensitive. Spaces, LineFeed, and commas are ignored in the input. A command is terminated with a Carriage Return (CR). All commands except select commands echo a Carriage Return when complete. Invalid commands will cause the selected 3C20 to echo a "?" .

The following table lists the serial host interface commands. In the table, > means Carriage Return, D means a hexadecimal data nibble. A means a hexadecimal address nibble. C means a hexadecimal count value nibble. Normal text is commands/data sent to 3C20, text in italics is from 3C20.

HOST INTERFACE

SERIAL COMMAND SET

READ COMMANDS

COMMAND	FORMAT	FUNCTION
RE	RE AA > ***DD >***	Read EEPROM byte
Re	Re AAAA > ***DDDD >***	Read EEPROM word
RI	RI > ***DDDD >***	Read IFIFO word
RQ	RQ > ***DDDD >***	Read QFIFO word
RM	RM CC > ***DDDDDD...>***	Read multiple words from IFIFO,CC is data words to read.
Rm	Rm CC > ***DDDDDD...>***	Read multiple words from QFIFO,CC is data words to read.
Rs	Rs > ***DDDD >***	Read FIFO Status register

WRITE COMMANDS

WE	WE AADD > **>**	Write EEPROM location AA with DD
We	We AAAADDDD > **>**	Write EEPROM location AAAA with DDDD
WP	WP DD > **>**	Enable (DD = 0x00) or Disable (DD=0xFF) EEPROM writes
WI	WI DDDD > **>**	Write word to IFIFO
WQ	WI DDDD > **>**	Write word to QFIFO

HOST INTERFACE

SERIAL COMMAND SET

COMMAND	FORMAT	FUNCTION
SELECT COMMANDS		
LS	LS DD >	Select 3C20 unit DD (and deselect all others)
MESASTART	MESASTART >	Enable all 3C20s to listen
MESASTOP	MESASTOP >	Disable all 3C20s

Note: Select commands do not echo a Carriage return.

IDENTIFY COMMANDS		
II	II > ***3C20 >***	Inquire ID
IR	IR > ***REVISION >***	Inquire Revision
FIFO INITIALIZATION COMMANDS		
CI	CI > **>**	Clear ICDFIFO
Ci	Ci > **>**	Clear IRBFIFO
CQ	CQ > **>**	Clear QCDFIFO
Cq	Cq > **>**	Clear QRBFIFO
FIFO COUNT READBACK COMMANDS		
cI	cI > **>**	Read ICDFIFO data count
ci	ci > **>**	Read IRBFIFO data count
cQ	cQ > **>**	Read QCDFIFO data count
cq	cq > **>**	Read QRBFIFO data count

HOST INTERFACE

FIFO DESCRIPTION

COMMAND/DATA FIFOS

Most communication to the 3C20s 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 16 words deep and the QCD FIFO is 64 words deep. The reason that there are two 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.

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 3C20 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. The IRBFIFO is 16 words deep and the QRBFIFO is 64 words deep.

HOST INTERFACE

FIFO DESCRIPTION

FIFOSTATUS REGISTER

The FIFOSTATUS register is used to determine FIFO status. Read only FIFO status is available in the top 8 bits of the FIFO status register. All FIFO status bits are active high. The bottom 8 bits are general purpose status bits that reflect the contents of the USERSTATUS register.

FIFO STATUS BITS

IFF	IFH	QFF	QFH	IRE	IRH	QRE	QRH	US7	US6	US5	US4	US3	US2	US1	US0

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 the FIFO status register reflect the lower 8 bits of the USERSTATUS register.

HOST INTERFACE

FIFO COMMANDS

Host communication consists of sending commands and data and reading returned data from either of the FIFOs. The low level commands shown here are sent to the 3C20 as hexadecimal strings that are part of the serial command protocol. 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 (should be 0 for single axis cards like 3C20)	
BITS 9..0	Parameter address	
BITS 9..0	Word count on block write commands	

HOST INTERFACE

FIFO 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**

HOST INTERFACE

FIFO COMMANDS

WRITE COMMANDS

EXAMPLE LONG WRITE PROCEDURE

(We will write 0XABCD1234 to location 0x40 in this example)

A polled host procedure for writing a 32 bit 3C20 parameter via the ICD FIFO is as follows:

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

Rs>**0A00>**

2. Write WRITE_LONG command to ICD FIFO

WIA040>**>**

3. Write least significant 16 bits of 32 bit parameter to ICD FIFO (via "WI" serial command).

WI1234>**>**

4. Write most significant 16 bits of 32 bit parameter to ICD FIFO (via "WI" serial command).

WIABCD>**>**

HOST INTERFACE

FIFO 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

FIFO COMMANDS

EXAMPLE WRITE BLOCK PROCEDURE

A polled host procedure for writing a block < (ICDFIFOSize/2 -2) of 3C20 parameters via the ICD FIFO. In this example we will write 0x20 words starting at parameter address 0x60

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

Rs>**0A00>**

2. Write WRITE_BLOCK command with word count to ICD FIFO (via "WI" serial command).

WIC01F>**>**

3. Write parameter address to ICD FIFO (via "WI" serial command).

WIC060>**>**

4. Write count+1 data words to ICD FIFO (via "WI" serial command).

WI1234>**>**

WI3456>**>**

WI5678>**>**

...

(total of 0x20 data words written)

HOST INTERFACE

FIFO COMMANDS

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

FIFO COMMANDS

EXAMPLE LONG READ PROCEDURE

An example polled host procedure for reading a 32 bit 3C20 parameter at address 0x10 via the ICD FIFO is as follows:

1. Check FIFO status register for ICD Full Flag If the flag is clear, there is enough room for a read command, otherwise wait.

Rs>**0A00>**

2. Write READ_LONG command to ICD FIFO

WI2010>>

3. If Baud rate > 230K, wait until IRB Empty flag is false

Rs>**0200>** (IRB false so continue)

4. Read least significant 16 bits of 32 bit parameter from ICD FIFO

RI>**1234>**

5. Read most significant 16 bits of 32 bit parameter from ICD FIFO

RI>**ABCD>**

HOST INTERFACE

FIFO COMMANDS

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 3C20 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

FIFO 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.

OPERATION

PARAMETERS

GENERAL

The 3C20 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 INCLUDED.INC and BITSD.INC supplied in the 3C20 support file set 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 INCLUDED.INC file lists all parameters.

MAKEINC

The supplied utility program MAKEINC will translate the INCLUDED.INC and BITSD.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 INCLUDED.INC 3C20.H C

(Create C include file of parameter addresses)

MAKEINC INCLUDED.INC MOTPARMS.PAS P S Loc

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

MAKEINC INCLUDED.INC MOTPARMS.ASM A M

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

OPERATION

PARAMETER TYPES

There are different types externally useable parameters:

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

PTR: 10 bit address pointer range 0 to 1023

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

DOUBLE: 64 bit signed number range -1.8E19 to 1.8E19

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE		FUNCTION
OPERATION FLAGS			
GO	FLAG		Start profile when set true, cleared when done
PID	FLAG		Enable PID portion of control loop when true
PROFILE	UINT		0= Off, 1=Trapezoidal profile mode, 2= external (cubic) profile mode
DIRINV	FLAG		Invert direction output polarity if true
HOME	FLAG		True when home operation is complete
MOTION	FLAG		True when in motion (velocity <> 0)
SLEW	FLAG		True when slew portion of profile has been reached
ERROR	UINT		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		Do hardware reset if cleared
STOPATHOME	FLAG		Set DESVEL and GO to 0 on home event if STOPATHOME is set
DRIVEMODE	FLAG		Selects motor drive mode, False = Voltage mode, true = Current mode.
SLOW	FLAG		Selects slow mode velocity scaling if set

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
MISC MONITORING AND CONTROL FUNCTIONS		
ACTVEL	INT	Actual velocity (counts/sample)
MAXPWM	UINT	Maximum PWM drive applied
AVGVEL	INT	Average actual (counted) velocity
MAXNEGERR	INT	Maximum negative deviation from profile. (Recorded counts)
MAXPOSERR	INT	Maximum positive deviation from profile (Recorded counts)
EXPOSERR	UINT	Excessive position error limit (counts)
DRIVEERROR	UNIT	Excessive drive detection logic accumulator
POSENC	PTR	PID loops pointer to position encoder (default = ENCP)
VELENC	PTR	PID loops pointer to position encoder used for velocity calculation (default = ENCP)
EVENTS	UINT	Number of events (max number depends on free memory)
FIXUP	LONG	Difference between profile end position and NEXTPOS (counts)

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
PID LOOP TUNING PARAMETERS (Working block)		
KK	UINT	PWM offset or bias
KP	UINT	Proportional constant
KD	UINT	Derivative constant
KA	UINT	Not used
KI	ULONG	Integral constant
KIL	UINT	Integral limit
KF1	UINT	Velocity feed forward term
KF2	UINT	Acceleration feed forward term
KF3	UINT	Phasor velocity feed forward
KFF	UINT	Friction feed forward / PWM deadzone term
KDFIL	UINT	Derivative term filter coefficient
DRIVEPLUS	UINT	Added to DRIVEERROR when Drive = MAXPWM
DRIVEMINUS	UINT	Subtracted from DRIVEERROR when drive < MAXPWM

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
MOTION CONTROL PARAMETERS		
DESPOSF	LONG	32 bit fractional part of DESPOS =.32
DESPOSD	DOUBLE	All 64 bits (32.32) of DESPOS
DESPOS	LONG	Desired Position (integer part = 32.)
VELOCITY	INT	Fractional part of 48 bit VELOCITY (=.16)
VELOCITY	LONG	Profile velocity (integer part = 32.)
ACCELF	INT	Fractional part of 48 bit ACCEL (=.16)
ACCEL	LONG	Profile Acceleration
JERK	LONG	Profile Jerk (Delta ACCEL)
NEXTPOS	LONG	Next position for internal profile generator
SLEWLIMIT	LONG	Slew speed during move

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
USER PARAMETERS		
ENCP	LONG	Primary encoder position (counts)
FOLLOW	PTR	Pointer to position for PID loop to follow (default = DESPOS)
DESVEL	LONG	Velocity target for velocity follower
HOMEPOSP	LONG	Primary encoder starting position count (loaded when index detected))
PHASEK	LONG	Phase constant for user timer (PHASEA=(PHASEK*sample)
PHASEA	LONG	User phase accumulator timer
PHASEF	FLAG	Set when PHASEA overflows
FLAGXOR	FLAG	Used by Wait on flag token
FLAGAND	FLAG	Used by wait on flag token
USERSTATUS	BYTE	Passed through to host interface on lower 8 bits of FIFOSTATUS register

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
MISCELLANEOUS PARAMETERS		
TIMEOUT	UINT	Count of DSP out of time events
SWREVISION	UINT	Firmware revision number, Major rev. = MSB, Minor rev. = LSB
CONTROLTYPE	UINT	LSB = Motor phases = 1
CPUTYPE	UINT	DSP CPU type
GPHASEK	LONG	Phase constant for user timer (GPHASEA=(GPHASEK*sample)
GPHASEA	LONG	Global phase accumulator timer
GPHASEF	FLAG	Flag set when GPHASEA overflows
PROCTIMER	UINT	Process timer, reads SYSCLK cycles for alll motion processing.
SYSCLK	LONG	3C20 SYSCLK frequency in Hz
EVENT AND GENERAL PURPOSE RAM (6 words per event)		
EVENT1..EVENT20		20 Events or 120 words of RAM

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
INTERNAL HARDWARE		
CNTCTL	UINT	Sets operational mode of quadrature input counter
CNTCLR	FLAG	If set, clears quadrature counter
PRESCALE	UINT	Sample rate prescale divider - sets sample rate as: SYSCLK/PRESCALE
PWMGENA	UINT	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	Single bit in MSB controls direction output bit
ENA	UINT	Controls Hbridge/Servo Amp enable bit
MOTORI	UINT	Selects Current Limit = (MOTORI*3A/65536) in Voltage control mode.

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
STATUS REGISTERS		
USERSTATUS	UINT	Writes here are reflected in lower 8 bits of FIFO status register
USER I/O PORTS		
INPORT	UINT	Input bits
OUTPORT	UINT	Output bits
ANALOG0	UINT	Analog in 0 read here
ANALOG1	UINT	Analog in 1 read here
ANALOG1	UINT	Analog in 2 read here

OPERATION

PARAMETER DESCRIPTIONS

PARAMETER	TYPE	FUNCTION
HARDWARE INFO		
NAXIS	UINT	Number of Axis for this hardware = 1
HWREVISION	UINT	Hardware revision number
ICDFIFOSZE	UINT	Immediate Command/Data FIFO size.
QCDFIFOSZE	UINT	Queued Command/Data FIFO size
IRBFIFOSZE	UINT	Immediate data Read-Rack FIFO size
QRBFIFOSZE	UINT	Queued data Read-Back FIFO size
EEPROM PARAMETERS		
ADDRESS	BYTE	3C20 card address (default = 0)
BAUDSEL	BYTE	Baud rate selector (see operation section)
SERIALMODE	BYTE	0 = RS-422, 1= RS-485
LOADPARAM	BYTE	If magic # = 0xDD then load params from EEPROM at startup (see operation section)

OPERATION

EEPROM PARAMETERS

The 3C20s processor has non-volatile EEPROM storage for some parameters. These include basic communication control parameters like Baud rate and serial communication mode. Also the 3C20 has the capability to save all volatile parameters to EEPROM and automatically restore the values at startup. Note: EEPROM parameters take effect only after a reset or power cycle.

BYTE EEPROM PARAMETERS

ADDRESS

When the 3C20 is used in multidrop applications (Multiple 3C20s share a single serial host connection), each 3C20 must be given a different address. The address is set by writing the ADDRESS EEPROM parameter.

BAUDSEL

The BAUD EEPROM parameter is a byte parameter that selects the communication baud rate. The following table show the 3C20 Baud rate versus BAUD value:

BAUDSEL	BAUD RATE	BAUDSEL	BAUDRATE
0	110	1	150
2	300	3	600
4	1,200	5	2,400
6	4,800	7	9,600
8	19,200	9	38,400
10	57,600	11	115,200
12	230,400	13	460,800
14	921,600	15	1,843,200

SERIALMODE

The SERIALMODE EEPROM parameter sets the communication mode to either RS-422 or RS-485 mode. When SERIALMODE is 0 , RS-422 mode is used, when SERIALMODE is 1, RS-485 mode is used. Both RS-422 and RS-485 modes support multidrop operation, but RS-422 mode must be use a 4 wire (Full duplex) serial link. RS-485 mode can be used with 2 wire (Half duplex) or 4 wire (Full duplex) serial links.

OPERATION

WORD EEPROM PARAMETERS

In addition to the few byte EEPROM setup parameters, the 3C20 can save motion and tuning parameters in its EEPROM. These parameters are accessed via the EEPROM word read and write commands. The word parameters have their own address space and do not conflict with the byte parameters. The EEPROM word parameters have the same addresses as the RAM parameters that they save. For example, writing -1 to the PID parameter address via the EEPROM word write command will write the EEPROM backup copy of the RAM PID parameter.

LOADPARAM

The LOADPARAM EEPROM parameter determines if the 3C20 loads its motion control parameters from EEPROM backup copies at startup. If LOADPARAM = 0xDD then all motion control parameters will be loaded from EEPROM. Any other value will cause all motion control parameters to be initialized to zero.

OPERATION

SERIAL COMMUNICATION BASICS

GENERAL

In order to communicate with the 3C20, you must first have a RS-422 or RS485 port on the host computer, the correct baud rate and correct connections to the 3C20. Instead of a RS-422/RS-485 port, a RS-232 serial port and a RS-232 -> RS-422 adapter like Mesas quad port Q422 can be used. The Q422 has the advantage that it uses the same RJ-45 pinout as the 3C20 so allows standard CAT5 Ethernet cables to be used between the Q422 and the 3C20(s)

MAKING THE 3C20 LISTEN

At startup the 3C20 will not listen to any commands except MESASTART. This allows coexistence with other RS-422/RS485 devices that share the same serial line. The MESASTART command is deliberately long so that it unlikely to be found in serial data transmitted or received from non-Mesa device on the serial line. Once enabled by MESASTART, the 3C20 needs to be selected by a LiSten ("LS") command. For example if you have a single 3C20 with default EEPROM programming (ADDRESS = 0) you would issue a:

LS00(CR) command

To select the 3C20. Once selected, the 3C20 will respond to all serial commands. To select another 3C20 sharing the same serial line at address 03, you would issue a

LS03(CR) command

This command will deselect 3C20 number 0 and select 3C20 number 3. At this point, all serial commands would be processed by 3C20 number 3.

If there are non-Mesa devices sharing the RS-422/ RS-485 interface, you will have to disable the 3C20s if you wish to communicate to the non-MESA devices. You do this by issuing the MESASTOP command. This puts all 3C20s in their non-listening state so only a MESASTART command will be accepted.

OPERATION

HARDWARE REGISTER DESCRIPTIONS

GENERAL

Most of the internal hardware in the 3C20 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	x	FILTER	CLRD	COI	IDXPOL	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

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 IDX — R/O — Read back realtime index input status

B1 B — R/O — Reads back realtime B input

B0 A — R/O — Reads back realtime A input

COUNT MODE

The encoder counter counts in quadrature mode, This is sometimes called the 4X mode, since a X line encoder will generate 4X counts per revolution in this mode.

OPERATION

INTERNAL HARDWARE

USERSTATUS REGISTER.

The UserStatus register can be written by the DSP. When the DSP writes to the UserStatus register, the low 8 bits of the data written is merged with the FIFO status and made available in FIFO status register, allowing events to pass information to the host without going through the FIFO.

US7	US6	US5	US4	US3	US2	US1	US0

OPERATION

INTERNAL HARDWARE

I/O PORTS

The 3C20 has 2 high current output bits and 3 input bits, and three analog inputs the input bits and analog in can be read at the INPORT and AIN(x) locations. The output bits can be controlled by writing to the OUTPORT location.

INPUT PORT

The three input bits are read by reading the INPORT parameter. Two of the input bits (IN1, IN0) are general purpose TTL level inputs with pullup resistors. One of the inputs (INL) is conditioned with a Schmitt trigger.

INL	X	X	X	X	X	X	X	X	X	X	X	X	X	IN1	IN0

OUTPUT PORT

Two output bits are available. These output bits drive external open drain MOSFETs meaning a 'high' output bit will result in a enabled MOSFET and a low output at the connector.

X	X	X	X	X	X	X	X	X	X	X	X	X	X	O1	O0

ANALOG IN

The 3C20 provides three 10 bit analog inputs (AIN0,AIN1,AIN2). All three analog inputs are sampled once per motion controller sample period. The 10 bit data is left justified and the least significant 6 bits are zeroed. Scaling is AIN = 65536*VIN/5V.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	0	0	0	0	0	0

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. The encoder counter operates in quadrature (4X) 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.

VELOCITY AND ACCELERATION UNITS

Dynamic units are a little more complex because they involve the sample period, and the SLOW flag. Slow mode is selected by setting the SLOW flag. The DESVEL, VELOCITY, and SLEWLIMIT parameters are 32 bit numbers with units of encoder_counts / (2^24) /sample_period in normal mode or encoder_counts/(2^32)/sample_period in slow mode.

In normal mode you can look at the velocity number 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 (= 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. In normal mode, acceleration units are in encoder_counts/2^24/sample_period^2.

In slow mode, the velocity is a 32 bit fraction, and maximum velocity is ~1 count per sample, With a 500 line encoder (= 2000 counts/rev) this is 300 RPM In slow mode, acceleration units are in encoder_counts/2^32/sample_period^2. Slow mode is useful for application that require very high velocity resolution at low speeds, such as telescope drives.

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)

(Normal mode)

SLEWLIMIT = (RPM * 2^32 * EncoderCountsPerRev) / (60 * SampleRate)

(Slow Mode)

OPERATION

MOTION UNITS

PWM AND SAMPLE RATE

The PID loop and Profile Generator operate at a fixed sample period determined by the setting of the PRESCALE parameter. The sample period is SYSCLK/PRESCALE. PWM rate is fixed at 14.4 KHz

MAXIMUM SAMPLE RATE

The 3C20 firmware is capable of running at ~20 KHz sample rate. 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 1 uSec.

If the sample rate is set faster than the DSP can process the motion calculations, 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 counts used by the DSP for motion processing. For example, at the default system clock of 29.152 Mhz, so a value of 700 would be equal to 700*34 nS = ~24 uSec. *Note that if a timeout event happens, the PROCTIMER parameter will be invalid for that cycle.*

If the sample rate is set too high, the DSP will not be able to complete motion processing in its alloted time, the motion controller will still work, but the sample time will be determined by the (variable) processing time instead the sample rate generator. Since velocity and acceleration values will be variable in this case, you should not normally run the motion controller in this mode.

OPERATION

PROFILE GENERATOR

GENERAL

There are two main parts of the motion controller firmware, the PID loop and the PROFILE generator. The basic job of the profile generator is to provide position information (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**.**

OPERATION

PROFILE GENERATOR

ERRORS AT HIGH SPEEDS

In trapezoidal mode, The profile generator can have small endpoint errors when more than one encoder count occurs between samples. These errors can be minimized by running the DSP at the highest possible sampling rate. The maximum error is ~(encoder-counts/sample). The error is fixed at the end of the move.

ABORTING A MOVE

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

NOTE: It is very important to first clear the go flag before setting DESVEL to 0 as otherwise the trapezoidal profile generator will assume a motion profile is still in effect, and do a large position fixup, possibly causing a large uncontrolled motion.

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. IVelocity is a signed 48 bit number consisting of VELOCITY as the most significant 32 bits and VELOCITYF as the least significant 16 bits.

IVelocity <= IVelocity +IAccel (48 bit signed add)

DESPOSD <= DESPOSD + IVelocity*256 (64 bit signed add normal mode)

DESPOSD <= DESPOSD+ IVelocity (64 bit signed add slow mode)

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.

TIME BASED S CURVE PROFILES

More sophisticated cubic profiles can be generated in external profile mode by using delays in the FIFO, and manipulating JERK. These time base profiles do not require real time host involvement, as they are done completely with wait-on-flag based delays and queued data in the FIFO. FIFO delays are easily generated by using one of the PHASE accumulators.

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	x	FILTER	CLRD	COI	IDXPOL	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 (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.

OPERATION

PROFILE GENERATOR

HOMING

To preset a encoder count without using index, The HOMEPOSP parameter is set to the desired count, and the hardware encoder counter is cleared by setting CNTCLRP location. The encoder count will be loaded with HOMEPOSP at the next sample time. As above, the DESPOS parameter will be set to the HOMEPOSP value and the HOME flag set when the encoder is preset.

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 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 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 *(SETPOINT* POSITION - *ACTUAL* POSITION) .

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

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 ~500 to ~10000

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.

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 3C20 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 3C20 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. 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 3C20 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 3C20 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.

CURRENT AND VOLTAGE MODES

While not strictly a PID parameter, DRIVEMODE affects the PID loop. DRIVEMODE determines whether the 3C20 output Hbridge works in voltage (Straight PWM control) or current feedback mode. When DRIVEMODE is set, current mode drive is selected, when DRIVEMODE is cleared, voltage mode drive is selected. Both modes have advantages and disadvantages.

CURRENT MODE

In current mode, motor current and therefore torque is proportional to PWM drive. This allows use of the MAXPWM parameter to control maximum applied torque. This is useful for torque limiting and constant torque applications. When current mode is used, motor torque is a more linear function of PWM drive and independent of motor back EMF. This linearity and motor speed/back EMF independence means that in current mode lower values of Velocity feed forward (KF1) are needed. The main disadvantage of current mode is that the motor ‘sees’ a high impedance from the HBridge, resulting in a low intrinsic damping factor. This low intrinsic damping factor means that a higher PID loop damping factor (KD) must be used for stability at a given gain (KP).

VOLTAGE MODE

In voltage mode, straight PWM is used to drive the Hbridge. When voltage mode is used, the MOTORI parameter can be use to set the maximum motor current. The main advantage of voltage mode is high intrinsic damping, as the motor ‘sees’ a low impedance from the HBridge, allowing a lower PID loop damping factor (KD). Because there is no current feedback in voltage mode, the motor back EMF will result in lower drive at higher motor speeds. This back EMF effect can be compensated with the velocity feed forward (KF1) parameter.

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 3C20 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 BUSINTFC, COMPORT, BAUDRATE. And BAUDRATEMUL. Valid values for BUSINTFC are TRUE and FALSE. Valid values for COMPORT are COM1,COM2,COM3 or COM4. Valid values for BAUDRATE are 110 through 115200. Certain serial ports allow baud rates high than 115200 by use of a preset baud rate multiplier. The BAUDRATEMUL parameter allows DMCTUNE to calculate the actual baud rate of such serial ports. For standard serial ports, BAUDRATEMUL should be set to 1.

If BUSINTFC is set to TRUE, DMCTUNE will attempt to access a PCI or PC/104 interfaced motion controller. For use with serial interfaced 3C20 controllers, the BUSINTFC parameter should be set to 'FALSE' and the COMPORT, BAUDRATE, BAUDRATEMUL parameters set appropriately.

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

OPERATION

TUNING

TUNING PROCEDURE

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

EVENT LOGIC

GENERAL

3C20 has aflexible 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, I/O bit manipulation, 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, and 16 and 32 bit multiplication. 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),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 3C20 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 3C20 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
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
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
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:

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

OPERATION

FILTER PARAMETER BLOCKS

GENERAL

To allow quick changes to the set of PID filter parameters, 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 3C20 can have room for up to 7 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

OPERATION

PHASE ACCUMULATOR

There are two user accessible phase accumulators available. These 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 user phase accumulator, 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 axis phase constant is called PHASEK and the axis phase accumulator is called PHASEA. The global phase constant is called GPHASEK and the global phase accumulator is called GPHASEA. Each phase accumulator has a flag that is set when the phase accumulator overflows.

As an example of phase accumulator usage, here is one way to setup a rate generator: Say we want to set a flag 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.

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.

USING PHASE ACCUMULATOR SAMPLE COUNT DELAYS

A common use of a phase accumulator is to provide simple integral sample count delays. For example FIFO processing can be paused a programmable number of samples by using the phase accumulator as a timer, and writing a WaitOnFlag token that polls the phase accumulator. For example, if PHASEAK is set to 1 (increment by one - this need only be done once) writing (2^31 - delay_count) to PHASEA, and then inserting a Wait_On_Flag token that waits for the MSB of PHASEA to be set, a sample count delay 0f 0 to 2^31 samples can be created in the FIFO.

OPERATION

DEMONSTRATION SOFTWARE

RP, WP, WF AND EVENT COMMAND LINE UTILITIES

Four simple command line utilities are provided for manually reading and writing 3C20 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 3 environment variables: COMPORT, BAUDRATE and BUSINTFC. COMPORT specifies the COM port used for serially accessed 3C20 hardware. COMPORT is a hex port address or one of COM1, COM2, COM3, or COM4. BAUDRATE is the decimal baudrate of the selected COM port. BUSINTFC is a flag to specify whether the serial or bus interfaced 3C20 hardware is to be accessed. BUSINTFC can be TRUE of FALSE. For example:

SET BUSINTFC=FALSE

SET BAUDRATE=115200

SET COMPORT=COM2

Would cause the RP, WP, or WF utility to use the serial interface on COM2 at 115200 baud to communicate to 3C20, while:

SET BUSINTFC=TRUE

Would cause the utilities to communicate with the 3C20 hardware on the host computers PC/104 or PCI bus.

OPERATION

DEMONSTRATION SOFTWARE

RP

The RP utility reads a parameter from 3C20. 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 3C20. 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 3C20. It uses symbolic names for the parameters so numeric constants do not need to be memorized, for example

WP KP 300

Would set the KP parameter

WP GO -1

Woudl set the GO flag

The WP utility has the same QFIFO option as RP

WP NEXTPOS 1234 Q

Would set the NEXTPOS parameter 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

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

WF MOTION 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

Would write a Wait-On 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 move, waits for the move to complete, waits 100000 sample times, and then does another move:

WP FLAGXOR FFFF H

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

WP FLAGAND 8000 H

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

WP NEXTPOS 50000

(we are going to position 50000)

WP GO -1

(start a motion)

WF GO

(wait for GO to clear when motion is done)

OPERATION

DEMONSTRATION SOFTWARE

WF
WP PHASEA 8001869F

(99,999 more than 80000000 H)

WP PHASEK -1

(Count down)

WF PHASEA+1

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

WP NEXTPOS 0

(we are going to 0 next)

WP GO -1

(start move towards 0)

WF GO

(wait for move to complete)

Note that running the previous sequence of commands would place the commands in 3C20'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 3C20 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

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, for example the following sequence:

WP EVENTS 0

(Disable events when we are loading them)

EVENT EventLogical ERROR 0000 FFFF 0000 NULL 1 A

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

EVENT EventLogicalIfDel EventNotZero NULL 0000 0000 0001 IRQCause 2 A

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

WP EVENTS 2

(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 OUTPORT whenever 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

( Unconditional subtract event to set the flags)

EVENT EventLogicalIf  EventGT OUTPORT 0000 FFFE 0001 OUTPORT 2 A

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

EVENT EventLogicalIf  EventLTEQ OUTPORT 0000 FFFE 0000 OUTPORT 3 A

( 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 )
```

SPECIFICATIONS

	MIN	MAX	NOTES
SUPPLY VOLTAGE (A REV)	10	32	Note 1
SUPPLY VOLTAGE (B REV)	10	50	Note 1
QUIESCENT POWER	—	500mW	
CONTINUOUS MOTOR CURRENT	---	2.00 A.	
PEAK MOTOR CURRENT	---	3.00 A.	10 Sec
ENCODER 5V POWER	---	250 mA.	
ENCODER INPUT HIGHV	2.0V	10V	TTL levels
ENCODER INPUT LOWV	-5V	.8V	Note 2
IN0,IN1 INPUT BIT HIGHV	2.0V	5.0V	TTL levels
INPUT BIT LOWV	-.5	.8V	Note 3
OUTPUT BIT VOLTAGE	-.3	50V	
OUTPUT BIT CURRENT	—	500 mA	Note 4
ANALOG IN	0V	5V	0-5V full scale
			Note 5

Note1: SUPPLY VOLTAGE MAX is absolute maximum. System design must guarantee that this limit is not exceeded. Running close to the limit may require voltage clamps.

Note 2: ENCODER/INDEX/LIMIT inputs are TTL level Schmitt triggers with 1K series resistors. Maximum input range -5 to +10V

Note 3: IN0 and IN1 have 10 K series resistor to DSP input pins, this will protect the DSP from voltages of +-25V on these pins, but these voltages may cause temporary malfunction.

Note 4: Output MOSFETS require a external clamp diode if used with inductive load

Note 5: Analog input range is 0 to +5V, inputs are protected with 10K series resistors which will protect the DSP from +-25V inputs, but these voltages may cause temporary malfunction.