USER MANUAL

Accessory 84E

Universal Serial Encoder Interface

3Ax-603927-xUxx

July 21, 2015

DELTA TAU
Data Systems, Inc.

NEW IDEAS IN MOTION …

Single Source Machine Control ········· Power // Flexibility // Ease of Use

21314 Lassen St. Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Safety Instructions

Qualified personnel must transport, assemble, install, and maintain this equipment. Properly qualified personnel are persons who are familiar with the transport, assembly, installation, and operation of equipment. The qualified personnel must know and observe the following standards and regulations:

IEC364resp.CENELEC HD 384 or DIN VDE 0100
IEC report 664 or DIN VDE 0110
National regulations for safety and accident prevention or VBG 4

Incorrect handling of products can result in injury and damage to persons and machinery. Strictly adhere to the installation instructions. Electrical safety is provided through a low-resistance earth connection. It is vital to ensure that all system components are connected to earth ground.

This product contains components that are sensitive to static electricity and can be damaged by incorrect handling. Avoid contact with high insulating materials (artificial fabrics, plastic film, etc.). Place the product on a conductive surface. Discharge any possible static electricity build-up by touching an unpainted, metal, grounded surface before touching the equipment.

Keep all covers and cabinet doors shut during operation. Be aware that during operation, the product has electrically charged components and hot surfaces. Control and power cables can carry a high voltage, even when the motor is not rotating. Never disconnect or connect the product while the power source is energized to avoid electric arcing.

Warning

A Warning identifies hazards that could result in personal injury or death. It precedes the discussion of interest.

Caution

A Caution identifies hazards that could result in equipment damage. It precedes the discussion of interest.

Note

A Note identifies information critical to the understanding or use of the equipment. It follows the discussion of interest.

MANUAL REVISION HISTORY				
REV	**DESCRIPTION**	**DATE**	**CHANGE**	**APPROVED**
1	Manual Creation.	04/27/10	CP	SS
2	Fixed addresses in EnDat section.	12/01/10	RN	RN
3	Added detailed information about EnDat, Yaskawa, Tamagawa, Panasonic, Mitutoyo, BiSS B/C, Matsushita A/D and Mitsubishi protocols.	07/17/14	SS	SS
4	Added example for incremental EnDat2.2	10/23/14	SS	SS
5	Added XY2-100 Protocol Appendix	05/08/15	SS	SS

Table of Contents

INTRODUCTION

Overview

The ACC-84E Universal Serial Encoder Interface Board provides up to four channels of serial encoders to be read by the UMAC and Ultralite/MACRO Station controllers. The ACC-84E is part of the UMAC or MACRO Pack family of expansion cards and these accessory cards are designed to plug into an industrial 3U rack system. The information from these accessories is passed directly to either the UMAC or MACRO Station CPU via the high speed JEXP expansion bus. ACC-84E supports different serial encoder protocols depending on the option ordered. These protocols are programmed into an on-board FPGA upon manufacturing. Multiple common protocols are supported at the moment and future developments of additional protocols are feasible. Currently, ACC-84E supports the following protocols:

● SSI	Synchronous Serial Interface
● EnDat 2.2	EnDat 2.2 interface from HEIDENHAIN
● Yaskawa	Yaskawa Sigma II/III/V feedback support
● Tamagawa	Tamagawa OAS and SA Absolute Encoders
● Panasonic	A4 and A5 Encoder Series
● Mitutoyo	Mitutoyo ENSIS® high-speed serial protocol (AT503/AT503A/ST70X)
● BiSS B/C	BiSS B/C Unidirectional
● Matsushita	
● Mitsubishi	Mitsubishi Serial Encoder Protocol for HG-X Servo Motors

Each ACC-84E can only support one of the protocols mentioned above for all four channels. If the customer has two different serial protocols in the system, two separate ACC-84E cards should be used. Since ACC-84E is strictly a feedback input card, if the feedback is intended to be used as the feedback for closed loop servo control, the servo command should be sent out to the amplifier using a UMAC axis interface card depending on the signal and control type required by amplifier. Here is a list of possible axis interface cards available for UMAC systems:

● ACC-24E2	Digital amplifier breakout w/ TTL encoder inputs or MLDT
● ACC-24E2A	Analog amplifier breakout w/ TTL encoder inputs or MLDT
● ACC-24E2S	Stepper amplifier breakout w/ TTL encoder inputs or MLDT
● ACC-24E3	Analog/Digital Output (Power PMAC Compatible Only)

Up to 12 ACC-84E boards can be connected to one UMAC providing up to 48 channels of serial encoder feedback. Because each MACRO Station CPU can service only eight channels of servo data, only two fully populated ACC-84E boards can be connected to the MACRO-Station.

The ACC-84E board will take the data from the serial encoder and process it as up to four 24-bit binary parallel words depending on protocol specifications. This data can then processed in the UMAC or MACRO Station encoder conversion table for position and velocity feedback. With proper setup, the information can also be used to commutate brushless and AC induction motors.

Compatibility

The ACC-84E can be used with any type of CPU available for UMAC systems. These CPUs include:

- Power PMAC UMAC CPU
- Turbo PMAC UMAC CPU
- MACRO16 UMAC CPU

SPECIFICATIONS

Environmental Specifications

Description	Specification	Notes
Operating Temperature	0°C to 45°C,	
Storage Temperature	-25°C to 70°C	
Humidity	10% to 95 % non-condensing	

Physical Specifications

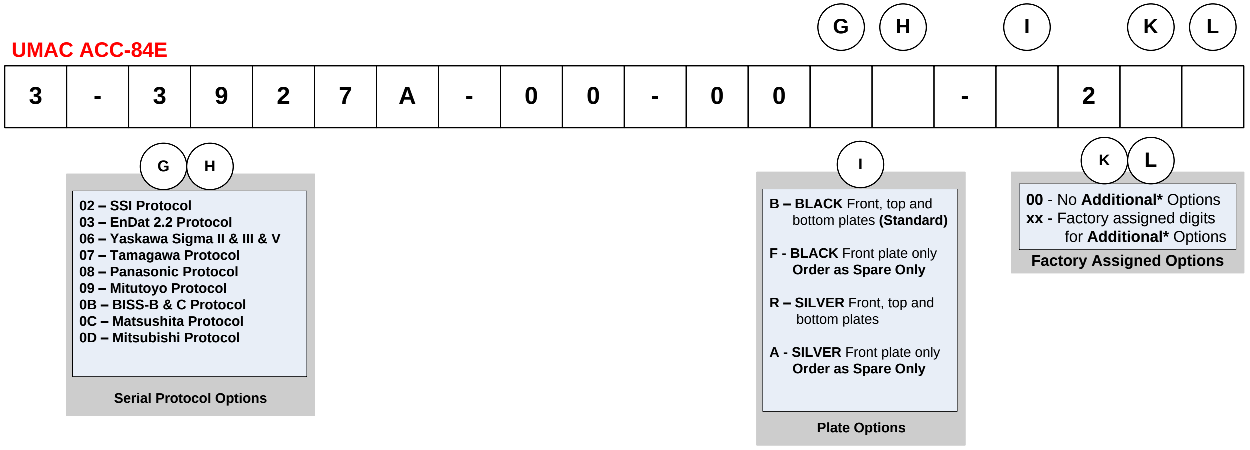
Description	Specification	Notes
Dimensions	Length: 16.256 cm (6.4 in.) Height: 10 cm (3.94 in.) Width: 2.03 cm (0.8 in.)	
Weight		
DB Connectors	DB9 Female	UL-94V0
The width is the width of the front plate. The length and height are the dimensions of the PCB.		

Electrical Specifications

Description	Specification	Notes
ACC-84E Power Requirements	5V @ 360mA (±10%) +15V @ 0 A -15V @ 0 A	5V current requirement mentioned is the consumption of the ACC-84E without any encoders connected.

Configuration

The ACC-84E can support different serial encoder protocols depending on the selected option. The following figure shows its part number scheme:



UMAC ACC-84E

													G	H		I		K	L
3	-	3	9	2	7	A	-	0	0	-	0	0			-		2		

G H

02 – SSI Protocol
03 – EnDat 2.2 Protocol
06 – Yaskawa Sigma II & III & V
07 – Tamagawa Protocol
08 – Panasonic Protocol
09 – Mitutoyo Protocol
0B – BISS-B & C Protocol
0C – Matsushita Protocol
0D – Mitsubishi Protocol

Serial Protocol Options

I

B – BLACK Front, top and bottom plates **(Standard)**

F - BLACK Front plate only **Order as Spare Only**

R – SILVER Front, top and bottom plates

A - SILVER Front plate only **Order as Spare Only**

Plate Options

K L

00 - No **Additional*** Options
xx - Factory assigned digits for **Additional*** Options

Factory Assigned Options

* If Any **Additional Option** is required, contact factory for digits **K** and **L** **(Factory Assigned** digits).

HARDWARE SETUP

The ACC-84E uses expansion port memory locations defined by the type of PMAC (Power, Turbo, or MACRO) to which it is directly communicating.

Addressing the ACC-84E

The Switch 1 (SW1) settings will allow you to select the starting address location for data from the first encoder. Data from encoders 2 through 4 will be placed at +4 memory locations from the base address and so on and so forth.

Chip Select	**Base Address**			**SW1 Positions**					
	TURBO	**MACRO**	**POWER**	**6**	**5**	**4**	**3**	**2**	**1**
CS10	Y:$78C00	Y:$8800	ACC84E[0]	ON	ON	ON	ON	ON	ON
	Y:$79C00	Y:$9800	ACC84E[4]	ON	ON	ON	**OFF**	ON	ON
	Y:$7AC00	Y:$A800	ACC84E[8]	ON	ON	**OFF**	ON	ON	ON
	Y:$7BC00	Y:$B800	ACC84E[12]	ON	ON	**OFF**	**OFF**	ON	ON
CS12	Y:$78D00	Y:$8840	ACC84E[1]	ON	ON	ON	ON	ON	**OFF**
	Y:$79D00	Y:$9840	ACC84E[5]	ON	ON	ON	**OFF**	ON	**OFF**
	Y:$7AD00	Y:$A840	ACC84E[9]	ON	ON	**OFF**	ON	ON	**OFF**
	Y:$7BD00	Y:$B840	ACC84E[13]	ON	ON	**OFF**	**OFF**	ON	**OFF**
CS14	Y:$78E00	Y:$8880	ACC84E[2]	ON	ON	ON	ON	**OFF**	ON
	Y:$79E00	Y:$9880	ACC84E[6]	ON	ON	ON	**OFF**	**OFF**	ON
	Y:$7AE00	Y:$A880	ACC84E[10]	ON	ON	**OFF**	ON	**OFF**	ON
	Y:$7BE00	Y:$B880	ACC84E[14]	ON	ON	**OFF**	**OFF**	**OFF**	ON

Note

ON designates **Closed**. **OFF** designates **Open**. Factory default is all ON.

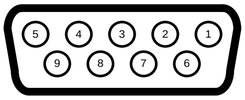
Signal Format

The signal format for the encoder is dependent on the particular protocol, but in all protocols, there is a “strobe” and/or “clock” output from the controller, and a data channel into the processor from the encoder. The encoder is queried synchronously with the Power/Turbo PMAC’s phase or servo clock, and the incoming serial data is latched into a memory-mapped register for the processor to read.

Connections

Encoders are connected to the ACC-84E through four 9-pin D-sub connectors. Two connectors on the top side of the rack for encoders 1 and 2, and two connectors in the bottom side for encoders 3 and 4.

D-Sub DE9 Female
Mating: D-Sub DE9 Male

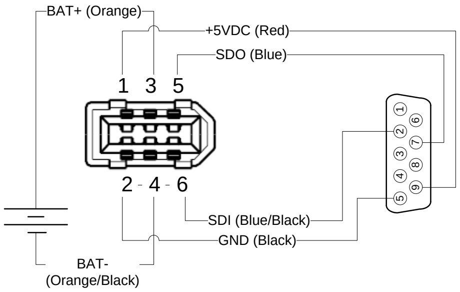


Pin # / Function			SSI / EnDat	Yaskawa	Tamagawa	Panasonic	Mitutoyo/ Mitsubishi	BiSS B/C	Matsushita
1	CLOCK–	OUT	CLK-	–				MA-	–
2	DATA–	IN/OUT	DAT-	SDI BLU/BLK	$\overline{\text{SD}}$	$\overline{\text{PS}}$	MRR (*RQ/*DT)	SLO-	/Rx
3	ENA–	OUT	–		$\overline{\text{SENA}}$	–			
4	GND	COM	**GND**	**GND** **BLK**	**GND**				
5	GND	COM	**GND**	**GND** **BLK**	**GND**				
6	CLOCK+	OUT	CLK+	–				MA+	–
7	DATA+	IN/OUT	DAT+	SDO BLU	SD	PS	MR (RQ/DT)	SLO+	Rx
8	ENA+	OUT	–		SENA	–			
9	+ 5V	OUT	**+ 5V**						

Encoder Specific Connection Information

Yaskawa Sigma II/III/V Encoders

Yaskawa Sigma II/III/V absolute encoders require a 3.6V battery to maintain the multi-turn data while the controller is powered down. This battery should be placed outside of ACC-84E and the Yaskawa Sigma II/III/V encoder, possibly on the cable. The battery should be installed between orange (+3.6V) and orange/black wires (GND). Use of ready-made cables by Yaskawa is recommended. (Yaskawa part number: UWR00650)



The previous diagram shows the pin assignment from mating IEEE 1394 Yaskawa Sigma II connector to ACC-84E encoder input. The Molex connector required for IEEE 1394 can be acquired as receptacle kit from Molex, 2.00mm (.079") Pitch Serial I/O Connector, Receptacle Kit, Wire-to-Wire, Molex Part Number: 0542800609.

Note

Yaskawa Encoder expects a supply voltage of 5V with less than 5% tolerance. Make sure voltage drop is not caused by excessive wire length.

Note

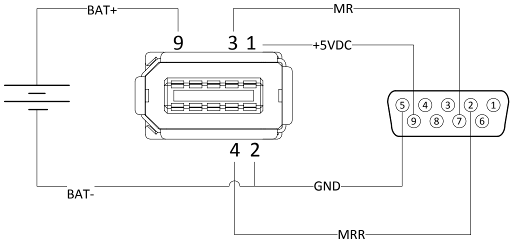
Encoder wire shield must be connected to chassis ground on both encoder and connector ends.

Note

Yaskawa Sigma II/III/V require a 120Ω termination resistor between SDI and SDO twisted pair lines on ACC-84E side.

Mitsubishi HG-□ Servo Motor Encoders

Mitsubishi HG-□ servo motor absolute encoders require a 3.6V battery to maintain the multi-turn data while the controller is powered down. This battery should be placed outside of ACC-84E and the Mitsubishi HG-□ servo motor's encoder, possibly on the cable. The battery should be installed between pin 9 of the motor encoder connector (+6V) and pin 2(GND). Use of ready-made cables by Mitsubishi is recommended. (Mitsubishi part number: UWR00650)



The diagram above shows the pin assignment from mating 3M SCR Receptacle (36110) to ACC-84E encoder input.

SOFTWARE SETUP

ACC-84E supports multiple protocols and for this reason the setup for each of them will be different. For each protocol, depending on the CPU type, the setup steps differs slightly, but the general idea regardless of the protocol is the same.

Position encoders that provide numerical position information in a serial data stream, usually representing absolute position information, are becoming increasingly popular. ACC-84E is an FPGA-based interface which is programmed to support different serial protocols. Multiple serial encoder protocols are supported by ACC-84E.

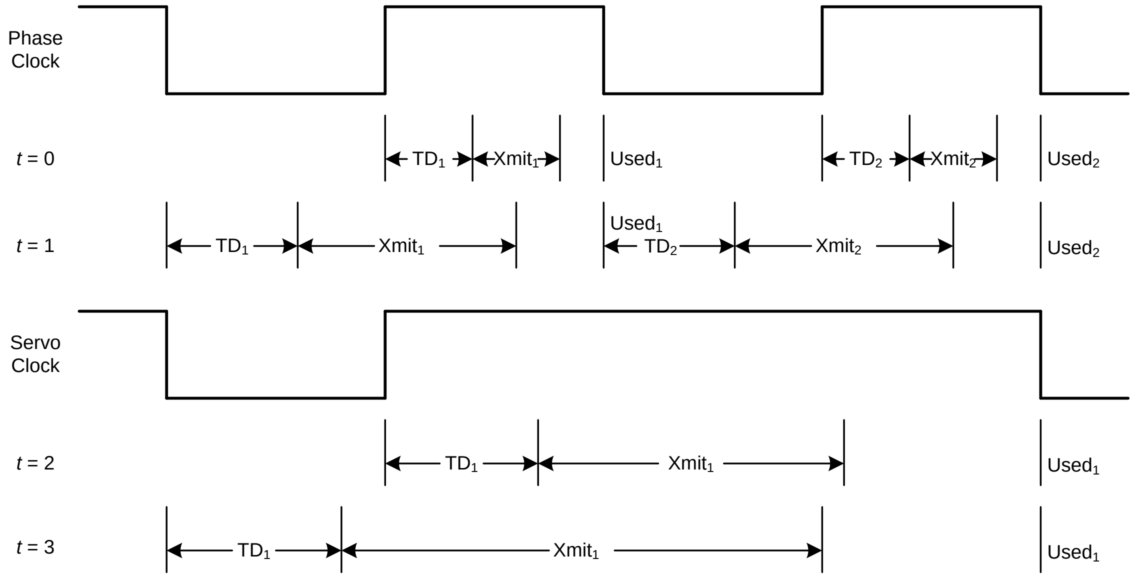
Hardware-Control Parameter Setup

This section describes the Power/Turbo PMAC serial encoder hardware interface in general terms. All of the supported serial encoder interfaces use differential signal pairs at 5V RS-422 levels. All have clock and/or "strobing" outputs, and all have a data signal input. In some protocols, the data line is bi-directional, supporting data output commands to the encoder.

The configuration of the hardware control registers differs slightly between serial protocols in the ACC-84E, however, the principles of setup are the same.

Because of the serial data protocol, the transfer of data from the encoder to the Power/Turbo PMAC interface circuitry takes a significant amount of time. The data must be ready for the processor immediately after the falling edge of the phase and/or servo clock signals, which are the interrupts to the processor telling it to start those respective tasks.

The process of querying the encoder for data must start well before these signal edges, and this timing must be carefully considered. If it starts too late, the data will not be ready in time. If it starts too early, unnecessary delay is introduced into the feedback loop, possibly compromising its performance. In both styles of interface, the multi-channel saved setup element permits the user to optimize the timing by selecting the edge (rising or falling) of the clock signal (phase or servo) that starts the triggering process, and the time delay from this edge until the actual triggering occurs. The following diagram shows the time lines for the possible configurations:



t: Trigger select code – clock and edge

TD$_n$: Trigger delay from edge for cycle *n*

Xmit$_n$: Data transmission for cycle *n*

Used$_n$: Data used by software for cycle *n*

Serial Encoder Interface Timing

The “SEIGATE” FPGA on the ACC-84E UMAC board has a multi-channel setup element that affects all channels on the IC, and a single-channel setup element for each channel.

This section describes the setup elements for the serial encoder interface in general terms. Detailed information for each serial encoder protocol can be found in the following reference chapters of the manual.

Note

This section describes the setup of the FPGA-based elements using the **Acc84E[*i*]** data structure. If you are using the FPGA-base serial encoder interface in the Power Brick, substitute “**Acc84B[*i*]**” for “**Acc84E[*i*]**”.

Multi-Channel Setup Element

The multi-channel setup element **Acc84E[*i*].SerialEncCtrl** (saved element in Power PMAC only and non-saved in Turbo PMAC, must be setup in power/initialization PLC) specifies several aspects of the serial encoder configuration for all four channels of the IC: the protocol, the trigger, and the clock frequency. All three of these aspects must be common to all four channels of the IC, so it is not possible, for instance, to interface to encoders with different protocols from the same IC.

The different components of this 24-bit full-word element cannot be accessed as independent elements, so it is necessary to assemble the full-word value from the values of the individual components. It is easiest to treat the value as a hexadecimal value, so the individual components can be seen independently.

Power PMAC Global Control Register	**Turbo PMAC Global Control Register**	**Switch Position (SW1)**			
		1	**2**	**3**	**4**
ACC84E[0].SerialEncCtrl	X:$78C0F	Close	Close	Close	Close
ACC84E[4].SerialEncCtrl	X:$79C0F	Close	Close	Open	Close
ACC84E[8].SerialEncCtrl	X:$7AC0F	Close	Close	Close	Open
ACC84E[12].SerialEncCtrl	X:$7BC0F	Close	Close	Open	Open
ACC84E[1].SerialEncCtrl	X:$78D0F	Open	Close	Close	Close
ACC84E[5].SerialEncCtrl	X:$79D0F	Open	Close	Open	Close
ACC84E[9].SerialEncCtrl	X:$7AD0F	Open	Close	Close	Open
ACC84E[13].SerialEncCtrl	X:$7BD0F	Open	Close	Open	Open
ACC84E[2].SerialEncCtrl	X:$78E0F	Close	Open	Close	Close
ACC84E[6].SerialEncCtrl	X:$79E0F	Close	Open	Open	Close
ACC84E[10].SerialEncCtrl	X:$7AE0F	Close	Open	Close	Open
ACC84E[14].SerialEncCtrl	X:$7BE0F	Close	Open	Open	Open

Acc84E[*i*].SerialEncCtrl is the full-word element that comprises the multi-channel setup for serial encoder interfaces for the ACC-84E. It is comprised of the following components (which cannot be accessed as independent elements):

Component	**Turbo PMAC/ Power PMAC Script Bits**	**Hex Digit #**	**C Bits**	**Functionality**
SerialClockMDiv	23 – 16	1 – 2	31 – 24	Serial clock linear division factor
SerialClockNDiv	15 – 12	3	23 – 20	Serial clock exponent division factor
(Reserved)	11 – 10	4	19 – 18	(*Reserved for future use*)
SerialTrigClockSel	09	4	17	Serial trigger source select
SerialTrigEdgeSel	08	4	16	Serial trigger source edge select
SerialTrigDelay	07 – 04	5	15 – 12	Serial trigger delay from source edge
SerialProtocol	03 – 00	6	11 – 08	Serial encoder protocol select (*read-only*)

The component *SerialClockMDiv* controls how an intermediate clock frequency is generated from the IC's fixed 100 MHz clock frequency. The resulting serial-encoder clock frequency is then generated from this intermediate clock frequency by the component *SerialClockNDiv*, described below.

The equation for this intermediate clock frequency f_{int} is:

$$f_{\text{int}}(MHz) = \frac{100}{M+1}$$

where *M* is short for *SerialClockMDiv*. This 8-bit component can take a value from 0 to 255, so the resulting intermediate clock frequencies can range from 100 MHz down to 392 kHz.

The component *SerialClockNDiv* controls how the final serial-encoder clock frequency is generated from the intermediate clock frequency set by *SerialClockMDiv*. The equation for this final frequency f_{ser} is:

$$f_{ser}(MHz) = \frac{f_{\text{int}}(MHz)}{2^N} = \frac{100}{(M+1)*2^N}$$

where *N* is short for *SerialClockNDiv*. This 4-bit component can take a value from 0 to 15, so the resulting 2^N divisor can take a value from 1 to 32,768.

For serial-encoder protocols with an explicit clock signal, the resulting frequency is the frequency of the clock signal that is output from the ACC-84E's IC to the encoder. For "self-clocking" protocols without an explicit clock signal, this frequency is the input sampling frequency, and will be 20 to 25 times higher than the input bit rate f_{bit}. Refer to the instructions for the particular protocol for details.

The component *SerialTrigClockSel* controls which Power PMAC clock signal causes the encoder to be triggered. This single-bit component is set to 0, the encoder will be triggered on the phase clock; if it is set to 1, the encoder will be triggered on the servo clock. If the encoder feedback is required for commutation rotor angle feedback, it should be triggered on the phase clock; otherwise it can be triggered on the servo clock.

The component *SerialTrigEdgeSel* controls which edge of the clock signal (phase or servo) selected by *SerialTrigClockSel* initiates the triggering process. If this single-bit component is set to 0, the triggering process starts on the rising edge; if it is set to 1, the triggering process starts on the falling edge.

Power PMAC software expects to have the resulting encoder data available to it immediately after the falling edge of the relevant phase or servo clock signal, which interrupts the processor to initiate the activity that reads this data. Since minimum delay from trigger to use is desirable, it is better to start the triggering on rising clock edge if the data can be fully transferred before the falling edge. If this is not possible, the falling edge should be used to start the triggering process.

It is best to choose the edge that minimizes the delay between the triggering of the encoder and its use by the Power/Turbo PMAC software. The software will use the received encoder value immediately after the falling edge of the phase clock for commutation feedback, and immediately after the falling edge of the servo clock for servo feedback.

If you are using the serial encoder data for commutation feedback, you must trigger using the phase clock in order to get new data every phase cycle. If there is sufficient time to receive the data in one half of a phase clock cycle, you should use the rising edge of the phase clock to trigger. For example, at the default phase clock frequency of 9 kHz, a clock cycle is 110 µsec. If the serial encoder data can be received within 55 µsec, the rising edge should be used. If not, the falling edge must be used.

If you are only using the serial encoder data for servo, and not commutation, feedback, the servo clock can be used for the trigger. However, it is still advisable to use the phase clock if possible to minimize the delay. When using the servo clock, as with the phase clock, use the rising edge if possible for the trigger, and the falling edge only if required.

Remember that the servo clock signal is low only for one half phase clock cycle. For example, with the default 9 kHz phase clock and 2.25 kHz servo clock, the servo clock is low for only a half of 110 µsec phase clock cycle, and the delay from the rising edge to the next falling edge is 385 µsec.

The component *SerialTrigDelay* specifies the delay from the specified clock edge to the actual start of the output signal that will trigger the encoder response, in units of the serial encoder clock. A non-zero value can be used to minimize the delay between triggering the encoder and its resulting use by the Power PMAC.

The triggering does not need to start exactly on the specified clock edge. The trigger delay component *SerialTrigDelay* specifies the number of 20-microsecond intervals after the specified clock edge before the triggering of the encoder actually begins. It can take a value of $0 to $F (0 to 15, or 0 to 300 microseconds). Non-zero values can be used to minimize the delay between triggering of the encoder and the use of its data in the next software cycle.

The component *SerialProtocol* controls which serial-encoder protocol is selected for all channels of the IC. This 4-bit component can take a value from 0 to 15. This component is read-only, as it reflects the protocol interface that was installed in the board at the factory.

Note

The FPGA used here comes with the interface for only a single serial protocol, which was pre-installed at the factory as specified in the order. This component of the element is read-only, simply notifying the user which protocol has been installed.

The following table shows the protocol selected for each value of this component (more protocols may be added):

Value	Protocol	Value	Protocol	Value	Protocol	Value	Protocol
0	(*Reserved*)	4	(*Reserved*)	8	Panasonic	12	Matsushita
1	(*Reserved*)	5	(*Reserved*)	9	Mitutoyo	13	Mitsubishi
2	SSI	6	Sigma II/III	10	(*Reserved*)	14	(*Reserved*)
3	EnDat	7	Tamagawa	11	BiSS-B/C	15	(*Reserved*)

When used in the Script environment (Both Turbo and Power PMAC), **Acc84E[*i*].SerialEncCtrl** is a 24-bit element. When used in the C environment (Power PMAC Only), it is a 32-bit element, with real data in the high 24 bits, so its value in the C environment is 256 times greater than its value in the Script environment.

SSI Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for an SSI encoder.

SerialClockMDiv:	= (100 / f_{bit}) - 1	// Serial clock frequency = bit transmission frequency
SerialClockNDiv:	= 0	// No further division unless *f* < 400 kHz
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $02	// Shows SSI protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (100 / 2.5) - 1 = 39 ($23) and **Acc84E[*i*].SerialEncCtrl** is set to $230002 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	2				3				0				0				0				2				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	1	0	0	0	1	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	1	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*--*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists a few common Serial clock frequency settings used with SSI protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
49 ($31)	0 ($0)	2.0 MHz
99 ($63)	0 ($0)	1.0 MHz
99 ($63)	1 ($1)	500.0 kHz
99 ($63)	2 ($2)	250.0 kHz

EnDat 2.1/2.2 Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for an EnDat encoder. The serial clock frequency is set 25 times higher than the external clock frequency, which is the bit transmission frequency $f_{bit\ (MHz)}$, to permit oversampling of the input signal.

SerialClockMDiv:	= (4 / f_{bit}) - 1	// Serial clock freq. = 25x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $03	// Shows EnDat protocol is programmed into IC

For example, for a 2.0 MHz bit transmission rate, ***SerialClockMDiv*** = (4 / 2) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $010003 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				3				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	1	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*--*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists a few common Serial clock frequency settings used with EnDat2.1/2.2 protocol:

SerialClockMDiv	*SerialClockNDiv*	Serial Clock Freq.	Bit Transmission Freq.
0 ($00)	2 ($2)	25.0 MHz	1.0 MHz
0 ($00)	3 ($3)	12.5 MHz	500 kHz
0 ($00)	4 ($4)	6.25 MHz	250 kHz

In newer version of the EnDat 2.1/2.2 firmware for ACC-84E (Released Feb. 4, 2014) bit 11 is used as backward compatible switch, which if set allows higher bit transmission frequencies.

Note

Higher serial clock/but transmission frequency is only supported for short encoder cables, since in FPGA implementation of EnDat 2.1/2.2 in ACC-84E does not have the delay compensation feature.

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** based upon the additional clock switch for an EnDat encoder. The the bit transmission frequency $f_{bit\ (MHz)}$ is fixed at 1/4 of the external clock frequency.

SerialClockMDiv:	= (25 / f_{bit}) - 1	// Serial clock freq. = 4x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialDivSelect:	= 0	// Selects higher clock frequency selection
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $03	// Shows EnDat protocol is programmed into IC

For example, for a 5.0 MHz bit transmission rate, ***SerialClockMDiv*** = (25 / 5) - 1 = 4 ($04) and **Acc84E[*i*].SerialEncCtrl** is set to $040803 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				4				0				8				0				3				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	1	-	0	0	0	0	0	0	0	0	1	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*DS*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists a few common Serial clock frequency settings used with EnDat2.1/2.2 protocol:

SerialClockMDiv	*SerialClockNDiv*	Serial Clock Freq.	Bit Transmission Freq.
1 ($01)	0 ($0)	50.0 MHz	12.5 MHz
2 ($02)	0 ($0)	33.33 MHz	8.33 MHz
3 ($03)	0 ($0)	25.0 MHz	6.25 MHz
4 ($04)	0 ($0)	20.0 MHz	5.0 MHz
5 ($05)	0 ($0)	16.66 MHz	4.16 MHz

Yaskawa Sigma II/III/V Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Yaskawa II/III/V encoder.

SerialClockMDiv:	= 0	// 100 MHz serial clock freq. = 25x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $06	// Shows Yaskawa II/III/V protocol is programmed

For example, for the standard 4.0 MHz bit transmission rate, a 100 MHz serial clock frequency is used, and **Acc84E[*i*].SerialEncCtrl** is set to $000006 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				0				0				0				0				6				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	0	0	1	1	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*--*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Yaskawa Sigma II/III/V protocol:

SerialClockMDiv	*SerialClockNDiv*	Serial Clock Frequency
0 ($00)	0 ($0)	100.0 MHz

Note

Yaskawa Sigma II/III/V transmission of position data requires a minimum transfer time of 62.5 µsec. It is important to choose trigger clock and trigger edge to ensure complete transmission of data each cycle before its use by the controller.

Tamagawa Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Tamagawa FA-Coder serial encoder. The serial clock frequency is set 20 times higher than the external clock frequency, which is the bit transmission frequency f_{bit}, to permit oversampling of the input signal.

SerialClockMDiv:	= (5 / f_{bit}) - 1	// Serial clock freq. = 20x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $07	// Shows Tamagawa protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (5 / 2.5) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $010007 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				7				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	1	1	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				--	--	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Tamagawa protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
1 ($01)	0 ($0)	50.0 MHz

Panasonic Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Panasonic encoder. The serial clock frequency is set 20 times higher than the external clock frequency, which is the bit transmission frequency f_{bit}, to permit oversampling of the input signal.

SerialClockMDiv:	= (5 / f_{bit}) - 1	// Serial clock freq. = 20x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $09	// Shows Panasonic protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (5 / 2.5) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $010008 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				8				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	1	0	0	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*--*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Panasonic protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
1 ($01)	0 ($0)	50.0 MHz

Mitutoyo Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Mitutoyo serial encoder. The serial clock frequency is set 20 times higher than the external clock frequency, which is the bit transmission frequency f_{bit}, to permit oversampling of the input signal.

SerialClockMDiv:	= (5 / f_{bit}) - 1	// Serial clock freq. = 20x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $09	// Shows Mitutoyo protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (5 / 2.5) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $010009 for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				9				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	1	0	0	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*--*	*--*	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Mitutoyo protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
1 ($01)	0 ($0)	50.0 MHz

BiSS B/C (Unidirectional) Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a BiSS B or BiSS C (unidirectional) encoder.

SerialClockMDiv:	= (100 / f_{bit}) - 1	// Serial clock frequency = bit transmission frequency
SerialClockNDiv:	= 0	// No further division unless *f* < 400 kHz
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $0B	// Shows BiSS protocol is programmed into IC

For example, for a 1.0 MHz bit transmission rate, ***SerialClockMDiv*** = (100 / 1.0) **-** 1 = 99 ($63) and **Acc84E[*i*].SerialEncCtrl** is set to $63000B for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	6				3				0				0				0				B				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	1	1	0	0	0	1	1	0	0	0	0	-	-	0	0	0	0	0	0	1	0	1	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				--	--	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists a few common Serial clock frequency settings used with BiSS B/C protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
24 ($18)	0 ($0)	4.0 MHz
99 ($63)	0 ($0)	1.0 MHz
99 ($63)	1 ($1)	500.0 kHz
99 ($63)	2 ($2)	250.0 kHz

Matsushita Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Matsushita serial encoder. The serial clock frequency is set 20 times higher than the external clock frequency, which is the bit transmission frequency f_{bit}, to permit oversampling of the input signal.

SerialClockMDiv:	= (5 / f_{bit}) - 1	// Serial clock freq. = 20x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $0C	// Shows Matsushita protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (5 / 2.5) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $01000C for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				C				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	1	1	0	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				--	--	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Matsushita protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
1 ($01)	0 ($0)	50.0 MHz

Mitsubishi Protocol

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a Mitsubishi serial encoder. The serial clock frequency is set 20 times higher than the external clock frequency, which is the bit transmission frequency f_{bit}, to permit oversampling of the input signal.

SerialClockMDiv:	= (5 / f_{bit}) - 1	// Serial clock freq. = 20x bit transmission freq.
SerialClockNDiv:	= 0	// No further division
SerialTrigClockSel:	= 0	// Use phase clock if possible
SerialTrigEdgeSel:	= 0	// Use rising clock edge if possible
SerialTrigDelay:	= 0	// Can increase from 0 if possible to reduce latency
SerialProtocol:	= $0D	// Shows Mitsubishi protocol is programmed into IC

For example, for a 2.5 MHz bit transmission rate, ***SerialClockMDiv*** = (5 / 2.5) - 1 = 1 ($01) and **Acc84E[*i*].SerialEncCtrl** is set to $01000D for triggering on the rising edge of phase clock without delay.

Hex Digit ($)	0				1				0				0				0				D				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	1	1	0	1	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				--	--	*TC*	*TE*	*SerialTrigDelay*				*SerialProtocol*					

The following table lists the only Serial clock frequency setting used with Mitsubishi protocol:

SerialClockMDiv	***SerialClockNDiv***	**Serial Clock Frequency**
1 ($01)	0 ($0)	50.0 MHz

Note

Mitsubishi Serial Encoder on HG-□ type servo motors can only be queried at 55.5µsec±1.0µsec (18 kHz), 111µsec±1.0µsec (9 kHz) and 222µsec±1.0µsec (4.5 kHz). If the request cycle is other than the above cycles the data will not be latched properly.

Single-Channel Setup Element

Each channel of the FPGA has a 24-bit saved setup element **Acc84E[*i*].Chan[*j*].SerialEncCmd** (saved element in Power PMAC only and non-saved in Turbo PMAC) that specifies exactly how the channel's serial encoder interface will operate, given the protocol, trigger timing, and frequency specified by the multi-channel element. It has multiple components that specify different aspects of this interface. Not all components are used in every protocol.

Power PMAC Channel Control Register	Turbo PAMC Base Address	Channel			
		1 (j=0)	2 (j=1)	3 (j=2)	4 (j=3)
ACC84E[0].Chan[j].SerialEncCmd	$78C00	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
ACC84E[4].Chan[j].SerialEncCmd	$79C00	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
ACC84E[8].Chan[j].SerialEncCmd	$7AC00	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
ACC84E[12].Chan[j].SerialEncCmd	$7BC00	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
ACC84E[1].Chan[j].SerialEncCmd	$78D00	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
ACC84E[5].Chan[j].SerialEncCmd	$79D00	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
ACC84E[9].Chan[j].SerialEncCmd	$7AD00	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
ACC84E[13].Chan[j].SerialEncCmd	$7BD00	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
ACC84E[2].Chan[j].SerialEncCmd	$78E00	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
ACC84E[6].Chan[j].SerialEncCmd	$79E00	Y:$79E00	Y:$79E04	Y:$79E08	Y:$79E0C
ACC84E[10].Chan[j].SerialEncCmd	$7AE00	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
ACC84E[14].Chan[j].SerialEncCmd	$7BE00	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C

Acc84E[*i*].Chan[*j*].SerialEncCmd is comprised of the following components. These components cannot be accessed as independent data structure elements, so the value of the element must be "built up" from the value of the individual components.

Component	Script Bits	Hex Digit #	C Bits	Functionality
SerialEncCmdWord	23 – 16	1 – 2	31 – 24	Serial encoder output command
SerialEncParity	15 – 14	3	23 – 22	Serial encoder parity type
SerialEncTrigMode	13	3	21	Serial trigger mode: continuous or one-shot
SerialEncTrigEna	12	3	20	Serial trigger enable
SerialEncGtoB	11	4	19	Serial SSI data Gray-to-binary convert control
SerialEncEna/ *SerialEncDataReady*	10	4	18	Serial encoder circuitry enable (write) Serial encoder received data ready (read)
SerialEncStatusBits	09 – 06	4 – 5	17 – 14	Serial encoder SPI number of status bits
SerialEncNumBits	05 – 00	5 – 6	13 – 08	Serial encoder bit length control

The full element can be viewed in the following format. In the Script environment (Both in Turbo and Power PMAC), it is accessed as a 24-bit element. In the C environment (Only in Power PMAC), it is accessed as a 32-bit element with the real data in the high 24 bits.

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value															-										-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*		*Status*				*NumBits*						

Note

This section provides information about **Acc84E[*i*].Chan[*j*].SerialEncCmd** that is common to all protocols. . For more detailed and protocol-specific information, refer to the corresponding section of the manual.

The 8-bit component *SerialEncCmdWord* is used to define a command value sent to the serial encoder in a protocol-specific manner. This value can be changed during an application for different functionality, such as resetting an encoder. Not all protocols require a command value.

The 2-bit component *SerialEncParity* defines the parity type to be expected for the received data packet (for those protocols that support parity checking). A value of 0 specifies no parity; a value of 1 specifies odd parity; a value of 2 specifies even parity. (A value of 3 is reserved for future use.)

The 1-bit component *SerialEncTrigMode* specifies whether the encoder is to be repeatedly sampled or just one time. A value of 0 specifies continuous sampling (every phase or servo cycle as set by the multi-channel element **Acc84E[*i*].SerialEncCtrl**); a value of 1 specifies one-shot sampling.

The 1-bit component *SerialEncTrigEna* specifies whether the encoder is to be sampled or not. A value of 0 specifies no sampling; a value of 1 enables sampling of the encoder. If sampling is enabled with *SerialEncTrigMode* at 0, the encoder will be repeatedly sampled (every phase or servo cycle as set by the multi-channel element **Acc84E[*i*].SerialEncCtrl**) as long as *SerialEncTrigEna* is left at a value of 1. However, if sampling is enabled with *SerialEncTrigMode* at 1, the encoder will be sampled just once, and the ACC-84E's IC will automatically set *SerialEncTrigEna* back to 0 after the sampling.

The 1-bit component *SerialEncGtoB* specifies whether the data returned in SSI protocol undergoes a conversion from Gray format to numerical-binary format or not. A value of 0 specifies that no conversion is done; a value of 1 specifies that the incoming data undergoes a Gray-to-binary conversion.

The 1-bit component *SerialEncEna / SerialEncDataReady* has separate functions for writing to and reading from the register. When writing to the register, this bit represents *SerialEncEna*, which enables the driver circuitry for the serial encoder. This bit must be set to 1 to use any protocol of serial encoder on the channel. If there is an alternate use for the same signal pins, this bit must be set to 0 so the encoder drivers do not conflict with the alternate use. ***Note that you cannot read back the value you have written to this bit!***

When reading from the register, you get the *SerialEncDataReady* status bit indicating the state of the serial data reception. It reports 0 during the data transmission indicating that valid new data is not yet ready. It reports 1 when all of the data has been received and processed. This is particularly important for slower interfaces that may take multiple servo cycles to complete a read; in these cases, the bit should be polled to determine when data is ready.

The 4-bit component *SerialEncStatusBits* specifies the number of status bits the interface will expect from the encoder in the SPI protocol. The valid range of settings is 0 to 12.

The 6-bit component *SerialEncNumBits* specifies the number of data bits the interface will expect from the encoder in the SSI, EnDat, or BiSS protocol. The valid range of settings for these protocols is 12 – 63. In other protocols, the number of bits is not specified this way, and this value does not matter, so this component is usually left at 0.

When used in the Script environment, **Acc84E[*i*].Chan[*j*].SerialEncCmd** is a 24-bit element. When used in the C environment, it is a 32-bit element, with real data in the high 24 bits, so its value in the C environment is 256 times greater than its value in the Script environment.

SSI Protocol

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for an SSI encoder.

SerialEncCmdWord:	= 0	// No command word supported for SSI protocol
SerialEncParity:	= ??	// Encoder-specific parity check
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= ??	// Encoder-specific data format
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for SSI protocol
SerialEncNumBits:	= ??	// Encoder-specific number of position bits returned

For example, for an SSI encoder with 25 position bits in Gray-code format with odd parity, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $005C19. (It may report back as $005819 if the data-ready status bit is not set.)

Hex Digit ($)	0				0				5				C				1				9				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	0	1	0	1	1	1	-	-	-	-	0	1	1	0	0	1	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

EnDat2.1/2.2 Protocol

The EnDat interface in the ACC-84E supports four 6-bit command codes that are sent directly to the encoder:

- 000111 ($07) for reporting position (EnDat2.1)
- 101010 ($2A) for resetting the encoder (EnDat2.1)
- 111000 ($38) for reporting position with possible additional information (EnDat2.2)
- 101101 ($2D) for resetting the encoder (EnDat2.2)

These 6 bits fit at the low end of the 8-bit *SerialEncCmdWord* command field of **Acc84E[*i*].Chan[*j*].SerialEncCmd**.

Note

By the EnDat standard, EnDat2.2 encoders should be able to accept and process EnDat2.1 command codes as well. However, not all encoders sold as meeting the EnDat2.2 standard can do this.

For EnDat2.2 encoders, the ACC-84E (starting 1st quarter 2014) also supports controller requests for additional information from the encoder through the use of Memory Range Select (MRS) codes. To implement these, the *SerialEncCmdWord* command field contains the MRS code. (In this mode, the ACC-84E sends the 111000 command code – report position with additional information – to the encoder.)

The following MRS codes are supported in the EnDat2.2 standard:

- $40 – Send additional info 1 w/o data content (NOP)
- $41 – Send diagnostic values
- $42 – Send position value 2 word 1 LSB
- $43 – Send position value 2 word 2
- $44 – Send position value 2 word 3 MSB
- $45 – Acknowledge memory content LSB
- $46 – Acknowledge memory content MSB
- $47 – Acknowledge MRS code
- $48 – Acknowledge test command
- $49 – Send test value word 1 LSB
- $4A – Send test value word 2
- $4B – Send test value word 3 MSB
- $4C – Send temperature 1
- $4D – Send temperature 2
- $4F – Stop sending additional information 1
- $50 – Send additional info 2 w/o data contents (NOP)
- $51 – Send commutation
- $52 – Send acceleration
- $53 – Send commutation & acceleration
- $54 – Send limit position signals
- $55 – Send limit position signals & acceleration
- $56 – Currently not assigned
- $5F – Stop sending additional information 2

The response from the encoder to specific MRS code data requests from the encoder depends on the availability of that data in the encoder. The additional information provided from supported MRS codes will be found in status elements **Acc84E[*i*].Chan[*j*].SerialEncDataC** and **SerialEncDataD**.

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for position reporting from an EnDat encoder.

SerialEncCmdWord:	= *{cmd/MRS code}*	// Command code
SerialEncParity:	= 0	// No parity check supported for EnDat protocol
SerialEncTrigMode:	= 0	// Continuous triggering (EnDat2.2)
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for EnDat protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for EnDat protocol
SerialEncNumBits:	= *{enc spec}*	// Encoder-specific number of position bits returned

For example, for an EnDat2.2 encoder with 37 position bits, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $381425 for continuous position reporting. (It may report back as $381025 if the data-ready status bit is not set.)

pg	3				8				1				4				2				5				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	0	1	1	1	0	0	0	-	-	0	1	-	1	-	-	-	-	1	0	0	1	0	1	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

This same encoder can be reset with a command word value of 45 ($2D) sent in one-shot mode with **Acc84E[*i*].Chan[*j*].SerialEncCmd** set to $2D3425.

Hex Digit ($)	2				D				3				4				2				5				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	0	1	0	1	1	0	1	-	-	1	1	-	1	-	-	-	-	1	0	0	1	0	1	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

For an EnDat2.1 encoder with 24 position bits, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $073418 for one-shot position reporting (at power-up). It will report back as $073018 until the data is received.

Hex Digit ($)	0				7				3				4				1				8				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	0	0	0	1	1	1	-	-	1	1	-	1	-	-	-	-	0	1	1	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

For an EnDat2.2 incremental encoder with 24 position bits, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $421418 for continuous position reporting with additional information of position 2 word 1. (It may report back as $421018 if the data-ready status bit is not set.)

Hex Digit ($)	4				2				1				4				1				8				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	1	0	0	0	0	1	0	-	-	0	1	-	1	-	-	-	-	0	1	1	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

Reading Additional Information from EnDat2.2

The following sequence of settings needs to be followed in order for user to read additional information from an EnDat2.2 compatible encoder:

- Change CommandCode in SerialEncoderCommand register to desired MRS code.
- Once bit 22 of the SerialEncoderCommand is set to 1 (all MRS codes share this property), then the command code changes to 001001 for next cycle of communication.
- Encoder transmits the position data and SEIGATE3 receives it and stores it in SerialEncDataA and SerialEncDataB.
- MRS code will be transmitted after reception of the position data (available in bits 16 to 22 of serial encoder data)
- Depending on the first content bit (the bit after Busy bit in addition package), the package is copied into either SerialEncoderDataC or SerialEncoderDataD register

Yaskawa Sigma II/III/V Protocol

The Yaskawa Sigma II/III/V interface supports position reporting and fault-reset modes. The command code for position reporting is $00; the command code for fault reset is $04.

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for position reporting from a Yaskawa Sigma II/III/V encoder.

SerialEncCmdWord:	= 0	// No command word for position reporting in Yaskawa
SerialEncParity:	= 0	// No parity check supported for Yaskawa protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Yaskawa protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Yaskawa protocol
SerialEncNumBits:	= 0	// Fixed number of position bits returned

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $001400 for continuous position reporting. (It may report back as $001000 if the ready status bit is not set.)

Hex Digit ($)	0				0				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	0	0	0	0	0	0	-	-	0	1	-	1	0	0	0	0	0	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

It is important for user to implement a background process, such as a PLC to check for TimeoutError, CRC_Error and CodingError bits available in **Acc84E[*i*].Chan[*j*].SerialEncDataB** register to ensure validity of connection and position data.

Most alarm conditions in Yaskawa Sigma II/III/V encoder are either level fault (non-latching) or are cleared at cycle power, but absolute encoders have two latching errors (Backup Alarm which is usually caused by loss/drainage of battery or Encoder Error Alarm which is caused by internal circuit bearkdown in the encoder) which are not cleared on cycle power and require special instructions transmitted from controller for clearing them.

To reset the encoder, the components must be set up as follows:

SerialEncCmdWord:	= $04	// $04 : Fault-reset command code // $00 : No-operation command code
SerialEncParity:	= 0	// No parity check supported for Yaskawa protocol
SerialEncTrigMode:	= 1	// Single-shot triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Yaskawa protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 4	// Special reset command for Yaskawa protocol
SerialEncNumBits:	= $01	// "Encoder address" for reset

This means that **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $043501 to start a fault reset. (It may report back as $043101 if the ready status bit is not set.)

Hex Digit ($)	0				4				3				5				0				1				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	0	0	0	1	0	0	-	-	1	1	-	1	0	1	0	0	0	0	0	0	0	1	-	-
Component:			*SerialEncCmdWord*						*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

The following steps show the procedure for clearing the latched alarms on absolute encoders which the user/plc should perform in certain order:

1. Write the value $043501 to **Acc84E[*i*].Chan[*j*].SerialEncCmd.**
2. Wait 10 milliseconds.
3. Wait for the trigger-enable component (Script bit 12) of this element to clear.
4. Wait for the busy signal (Script bit 8) of **Acc84E[*i*].Chan[*j*].SerialEncDataB** to clear. If cleared go to step 7.
5. Clear the command code of this element to $00 by writing $003501 to the element.
6. Repeat steps 2 to 4.
7. Resume continuous position requests by writing $001400 to the element.

Tamagawa FA-Coder Protocol

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a Tamagawa FA-Coder serial encoder.

SerialEncCmdWord:	= $1A	// Command word for position reporting in Tamagawa
SerialEncParity:	= 0	// No parity check supported for Tamagawa protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Tamagawa protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Tamagawa protocol
SerialEncNumBits:	= 0	// Fixed number of position bits returned

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $1A1400 for continuous position reporting. (It may report back as $1A1000 if the ready status bit is not set.)

Hex Digit ($)	1				A				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	1	1	0	1	0	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

If the *SerialEncCmdWord* component is set to $BA and triggered for 10 consecutive cycles with 40 microseconds interval or more, all latched errors (Overspeed, Counter Overflow, Muti-turn error, Counting Error II and Battery Error). This should be done in “one-shot” mode, making the element equal to $BA3400 and triggered for 10 consecutive cycles with 40 microseconds interval or more.

If the *SerialEncCmdWord* component is set to $C2 and triggered for 10 consecutive cycles with 40 microseconds interval or more, the multi-turn position value in the encoder is reset to 0 and also all latched errors are cleared. This should be done in “one-shot” mode, making the element equal to $C23400 and triggered for 10 consecutive cycles with 40 microseconds interval or more.

When the reset operation is done, the component should report as $BA2000, $C22000 respectively.

Panasonic Protocol

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a Panasonic serial encoder.

SerialEncCmdWord:	= $2A	// Command word for multi-turn position in Panasonic
SerialEncParity:	= 0	// No parity check supported for Panasonic protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Panasonic protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Panasonic protocol
SerialEncNumBits:	= 0	// Fixed number of position bits returned

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $2A1400 for continuous position reporting. (It may report back as $2A1000 if the ready status bit is not set.)

Hex Digit ($)	2				A				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	1	0	1	0	1	0	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

If the *SerialEncCmdWord* component is set to $52 for single-turn position reporting with alarm code, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $521400. (It may report back as $521000 if the data-ready status bit is not set.) If the *SerialEncCmdWord* component is set to $52, the encoder ID value is also reported.

If the *SerialEncCmdWord* component is set to $4A and triggered for 10 consecutive cycles with 40 microseconds interval or more Counter Overflow and Battery Alarm flags are cleared. This should be done in “one-shot” mode, making the element equal to $4A3400 and triggered for 10 consecutive cycles with 40 microseconds interval or more. The register should report as $4A2000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $F2 and triggered for 10 consecutive cycles with 40 microseconds interval or more, all latched errors are cleared (Overspeed, Counter Overflow, Mutiple Revolution Error, Count Error II, Battery Alarm and System Down). This should be done in “one-shot” mode, making the element equal to $F23400 and triggered for 10 consecutive cycles with 40 microseconds interval or more. The register should report as $F22000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $DA and triggered for 10 consecutive cycles with 40 microseconds interval or more, the multi-turn position value in the encoder is reset to 0 (single revolution data will not be reset) and also all latched errors are cleared (Overspeed, Counter Overflow, Mutiple Revolution Error, Count Error II, Battery Alarm and System Down). This should be done in “one-shot” mode, making the element equal to $DA3400 and triggered for 10 consecutive cycles with 40 microseconds interval or more. The register should report as $DA2000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $7A and triggered for 10 consecutive cycles with 40 microseconds interval or more single revolution data will be reset to 0°±0.35° (MAX). This should be done in “one-shot” mode, making the element equal to $7A3400 and triggered for 10 consecutive cycles with 40 microseconds interval or more. Notice that this reset command (all 10) should only be sent when the encoder is at rest with no movement. Once reset, the single turn zero location is maintained regardless of connection of external battery after main power source is turned off. The register should report as $7A2000 after completion of a single trigger.

Mitutoyo Protocol

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a Mitutoyo serial encoder.

SerialEncCmdWord:	= $01	// Command word for position reporting in Mitutoyo
SerialEncParity:	= 0	// No parity check supported for Mitutoyo protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Mitutoyo protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Mitutoyo protocol
SerialEncNumBits:	= 0	// Fixed number of position bits returned

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $011400 for continuous position reporting. (It may report back as $011000 if the ready status bit is not set.)

Hex Digit ($)	0				1				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	0	0	0	1	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

If the *SerialEncCmdWord* component is set to $89 and sent 8 times, the encoder is reset (equivalent to power cycle on encoder). This should be done in "one-shot" mode repeated 8 times, making the element equal to $893400. When the reset operation is done, the component should report as $892000.

If this component is set to $9D, the encoder will report its ID value. This should be done in "one-shot" mode, and the IC will hold this value in status element **Acc84E[*i*].Chan[*j*].SerialEncDataC**.

BiSS-B/C (Unidirectional) Protocol

For the BiSS-B and BiSS-C (unidirectional) protocols, the *SerialEncCmdWord* component of **Acc84E[*i*].Chan[*j*].SerialEncCmd** specifies the CRC polynomial used for error detection when the position and status data are reported. This is an 8-bit mask value "M" that can define any 4-bit to 8-bit CRC polynomial. It must be set up to match the polynomial used for the particular BiSS encoder.

The mask bits M_7 to M_0 represent the coefficients for the terms x^8 to x^1, respectively, in the CRC polynomial:

$$M_7x^8 + M_6x^7 + M_5x^6 + M_4x^5 + M_3x^4 + M_2x^3 + M_1x^2 + M_0x^1 + 1$$

The coefficient for x^0 in a CRC polynomial is always 1, and so is not included in the mask. A mask with all zeros is used to indicate that no CRC bits are included with the encoder data.

For example, if the encoder uses a CRC polynomial of $x^6 + x^1 + 1$ (as with the Renishaw Resolute™ encoders), the CRC mask value M should be set to 00100001 (bits M_5 and M_0 set to 1), or $21.

For the BiSS protocol, the *SerialEncParity* component of **Acc84E[*i*].Chan[*j*].SerialEncCmd** is used to distinguish between the BiSS-B and BiSS-C protocol variants. Bit 1 of the component (bit 15 of the 24-bit element) is set to 0 for BiSS-C, and to 1 for BiSS-B. (BiSS-C provides a zero bit between the start bit and the position data; BiSS-B does not.) Hengstler Acuro®-Drive™ is an example of encoders supporting BiSS-B (unidirectional) protocol.

Bit 0 of the component (bit 14 of the 24-bit element) is only used for BiSS-B. If it is set to 1, it permits the acceptance of a "Multi-Cycle Data" (MCD) bit from the encoder by providing an extra clock cycle output. The MCD bit is not captured or used.

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a BiSS C encoder.

SerialEncCmdWord:	= $21	// CRC polynomial of $x^6 + x^1 + 1$
SerialEncParity:	= 0	// BiSS-C protocol variant
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for EnDat protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= *{enc spec}*	// Encoder-specific number of position bits returned
SerialEncNumBits:	= *{enc spec}*	// Encoder-specific number of status bits returned

For example, for a BiSS-C encoder with 36 position bits, 2 status bits, and a CRC polynomial of $x^6 + x^1 + 1$ (as with the Renishaw Resolute™ encoders), **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $2114A4 for continuous position reporting. (It may report back as $2110A4 if the data-ready status bit is not set.)

Hex Digit ($)	2				1				1				4				A				4				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	1	0	0	0	0	1	0	0	0	1	-	1	-	0	1	0	1	0	0	1	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

For a BiSS-B encoder with 32 position bits, 4 status bits, an MCD bit, and a CRC polynomial of $x^4 + x^1 + 1$, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $09D520 for continuous position reporting. (It may report back as $09D120 if the data-ready status bit is not set.)

Hex Digit ($)	0				9				D				5				2				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	0	1	0	0	1	1	1	0	1	-	1	-	1	0	0	1	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*						*NumBits*					

Matsushita Protocol

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a Matsushita serial encoder.

SerialEncCmdWord:	= $CA	// Command word for multi-turn position in Matsushita
SerialEncParity:	= 0	// No parity check supported for Matsushita protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Matsushita protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Matsushita protocol
SerialEncNumBits:	= 0	// first 4 bits represent the Encoder ID

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $CA1400 for continuous position reporting. (It may report back as $CA1000 if the ready status bit is not set.)

Hex Digit ($)	C				A				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	1	1	0	0	1	0	1	0	0	0	0	1	-	1	-	-	-	-	-	-	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Reserved*						*EncoderID*					

Matsushita protocol supports two modes of communication: independent and continuous. In Delta Tau's implementation of this protocol, only the independent communication is supported where the ID of the request packet should match the ID of the encoder.

If the *SerialEncCmdWord* component is set to $9A for single-turn position reporting (first 16-bits of single turn data), **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $9A1400. (It may report back as $9A1000 if the data-ready status bit is not set.)

If the *SerialEncCmdWord* component is set to $A2 for multi-turn position plus MSB of single turn data reporting (15 bits of multiple turn data + bit 17 of single turn data), **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $A21400. (It may report back as $A21000 if the data-ready status bit is not set.)

If the *SerialEncCmdWord* component is set to $AA for Alarm code data reporting, **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $AA1400. (It may report back as $AA1000 if the data-ready status bit is not set.)

If the *SerialEncCmdWord* component is set to $CA for single-turn + multi-turn + alarm reporting (17-bits of single turn data+15-bits of multi-turn data + 8bits of alarm data), **Acc84E[*i*].Chan[*j*].SerialEncCmd** would be set to $CA1400. (It may report back as $CA1000 if the data-ready status bit is not set.)

If the *SerialEncCmdWord* component is set to $E2 (Reset I) and triggered for 10 consecutive cycles with 7 microseconds interval or more battery alarm, system down and over speed flags are cleared. This should be done in "one-shot" mode, making the element equal to $E23400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. The register should report as $E22000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $EA (Reset II) and triggered for 10 consecutive cycles with 7 microseconds interval or more, multi-turn data and counter overflow are reset. This reset mode should only be called when the motor speed is less than 300 RPM. This should be done in "one-shot" mode, making the element equal to $EA3400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. The register should report as $EA2000 after completion of a single trigger.

Note

For both $E2 and $EA reset modes:

- To check whether the reset is made correctly, Send a request signal $CA to the encoder' and check the multi-turn data and ALC on the output signal data field.
- Single-turn absolute data is prohibited from resetting.

If the *SerialEncCmdWord* component is set to $B2 and triggered for 10 consecutive cycles with 7 microseconds interval or more single revolution data will be reset to 0° and shaft position will be written to EEPROM. This should be done in "one-shot" mode, making the element equal to $B23400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. Notice that this reset command (all 10) should only be sent when the encoder is at rest with no movement. Once reset, the single turn zero location is maintained regardless of connection of external battery after main power source is turned off. The register should report as $B22000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $BA and triggered for 10 consecutive cycles with 7 microseconds interval or more single revolution data will be reset to its initial data and EEPROM data will be reset. This should be done in "one-shot" mode, making the element equal to $BA3400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. Notice that this reset command (all 10) should only be sent when the encoder is at rest with no movement. Once reset, the single turn zero location is maintained regardless of connection of external battery after main power source is turned off. The register should report as $BA2000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $F2/$FA and triggered for 10 consecutive cycles with 7 microseconds interval or more, The Encoder ID present in lower 4 bits of **Acc84E[*i*].Chan[*j*].SerialEncCmd** is written to EEPROM on the encoder. This should be done in "one-shot" mode, making the element equal to $F23400/$FA3400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. The register should report as $F22000/$FA2000 after completion of a single trigger.

If the *SerialEncCmdWord* component is set to $D2/$DA and triggered for 10 consecutive cycles with 7 microseconds interval or more, The Encoder ID stored in EEPROM of the encoder is fored into "S8-shaft" (111). This should be done in "one-shot" mode, making the element equal to $D23400/$DA3400 and triggered for 10 consecutive cycles with 7 microseconds interval or more. The register should report as $D22000/$DA2000 after completion of a single trigger.

Note

Only the encoders that are written with an encoder ID “S8-shaft” permit to change the setting of the encoder ID.

If an encoder has an ID, other than S8 (111), it is necessary to assign it the ID S8 using $D2/DA mode before assigning it its final ID.

The difference between $F2 and $FA is similar to the difference between $D2 and $DA command modes. The command modes $F2 and D2 despite different functionality, cause the overflow flag to be reflected in the ea0 status bit. The command modes $FA and DA, prevent the reflection of overflow flag in the ea0 status bit.

Mitsubishi Protocol

The Mitsubishi encoder has 8 request codes defined for the Request Field transmitted from the controller to the encoder. To transmit a specific ID code to the encoder, program the *SerialEncCmdWord* register with the appropriate Command Code listed below:

- $02 – for reporting single-turn data (single-turn data in lower 18 bits of 24-bit word. 18-bit single-turn data in bits 0 to 17 and bits [18:23] report 0)
- $8A – for reporting multi-turn data
- $92 – for reporting Encoder-ID
- $A2 – for reporting single-turn and multi-turn data (single-turn data in lower 20 bits of 24-bit word. 18-bit single-turn data in bits 2 to 19 and bits [0:1] and [18:23] report 0)
- $2A – for reporting single-turn and multi-turn data (single-turn data in lower 18 bits of 24-bit word. 18-bit single-turn data in bits 0 to 17 and bits [18:23] report 0)
- $32 – for reporting single-turn and multi-turn data (single-turn data in upper 20 bits of 24-bit word. 18-bit single-turn data in bits 6 to 23 and bits [0:5] report 0)
- $BA – for clearing alarms and reporting single-turn data(single-turn data in lower 18 bits of 24-bit word. 18-bit single-turn data in bits 0 to 17 and bits [18:23] report 0)
- $7A – for reporting encoder/motor ID

The following list shows typical settings of **Acc84E[*i*].Chan[*j*].SerialEncCmd** for a Mitsubishi HG type servo motor's serial encoder.

SerialEncCmdWord:	= $32	// Command word for position reporting in Mitsubishi
SerialEncParity:	= 0	// No parity check supported for Mitsubishi protocol
SerialEncTrigMode:	= 0	// Continuous triggering
SerialEncTrigEna:	= 1	// Enable triggering
SerialEncGtoB:	= 0	// No Gray code supported for Mitsubishi protocol
SerialEncEna:	= 1	// Enable driver circuitry
SerialEncStatusBits:	= 0	// No status bits supported for Mitsubishi protocol
SerialEncNumBits:	= 0	// Fixed number of position bits returned

Acc84E[*i*].Chan[*j*].SerialEncCmd would be set to $321400 for continuous position reporting. (It may report back as $321000 if the ready status bit is not set.)

Hex Digit ($)	3				2				1				4				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	1	1	0	0	1	0	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	-	-
Component:	*SerialEncCmdWord*								*Parity*		*TM*	*TE*	*GB*	*Ena*	*Status*				*NumBits*							

If the *SerialEncCmdWord* component is set to $BA and triggered for 10 consecutive cycles with 55.5 microseconds interval or more, the ABS lost alarm is cleared. This should be done in "one-shot" mode, making the element equal to $BA3400 and triggered for 10 consecutive cycles with 55.5 microseconds interval or more.

If the *SerialEncCmdWord* component is set to $7A and triggered for 10 consecutive cycles with 222±1.0µsec interval, the encoder/motor ID is sent back from encoder to controller. In order to exit from this mode, any other mode command ($02,$8A,$92,$A2,$2A,$32) should be triggered for 10 consecutive cycles with 222±1.0µsec interval after which normal cyclic position reporting could be resumed. This should be done in "one-shot" mode, making the element equal to $7A3400 and triggered for 10 consecutive cycles with 222 microseconds interval. Once encoder ID is retrieved, any other mode command (for example $323400) should be triggered for 10 consecutive cycles with 222±1.0µsec interval to exit the encoder ID reporting mode.

When the reset operation is done, the component should report as $BA2000, $7A2000 respectively.

Hardware-Status Data Structure

Status elements of ACC-84E are read only elements where the received data and status flags are written by FPGA at every trigger event. There are no global status registers and only channel specific registers are defined.

Single-Channel Status Elements

Some aspects of the serial-encoder, such as position data, alarm, encoder ID and some additional information, can be read individually for each channel.

The type of data which can be read from these single-channel status elements, is dependent on each specific protocol and mode of operation within the same protocol as different modes of operation could result in different contents in these status elements. Each channel of the FPGA has four 24-bit status elements:

- **Acc84E[*i*].Chan[*j*].SerialEncDataA**
- **Acc84E[*i*].Chan[*j*].SerialEncDataB**
- **Acc84E[*i*].Chan[*j*].SerialEncDataC**
- **Acc84E[*i*].Chan[*j*].SerialEncDataD**

Depending on each protocol mode setting in **Acc84E[*i*].SerialEncCtrl** and

Acc84E[*i*].Chan[*j*].SerialEncCmd that specifies exactly how the channel's serial encoder interface will operate, different data formats will be presented in the data registers. Not all data registers are used in every protocol.

POWER	TURBO	Power PMAC ACC-84E Data Register	Channel			
			1 (j=0)	2 (j=1)	3 (j=2)	4 (j=3)
ACC84E[0]	$78C00	Chan[j].SerialEncDataA	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
		Chan[j].SerialEncDataB	Y:$78C01	Y:$78C05	Y:$78C09	Y:$78C0D
		Chan[j].SerialEncDataC	Y:$78C02	Y:$78C06	Y:$78C0A	Y:$78C0E
		Chan[j].SerialEncDataD	Y:$78C03	Y:$78C07	Y:$78C0B	Y:$78C0F
ACC84E[4]	$79C00	Chan[j].SerialEncDataA	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
		Chan[j].SerialEncDataB	Y:$79C01	Y:$79C05	Y:$79C09	Y:$79C0D
		Chan[j].SerialEncDataC	Y:$79C02	Y:$79C06	Y:$79C0A	Y:$79C0E
		Chan[j].SerialEncDataD	Y:$79C03	Y:$79C07	Y:$79C0B	Y:$79C0F
ACC84E[8]	$7AC00	Chan[j].SerialEncDataA	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
		Chan[j].SerialEncDataB	Y:$7AC01	Y:$7AC05	Y:$7AC09	Y:$7AC0D
		Chan[j].SerialEncDataC	Y:$7AC02	Y:$7AC06	Y:$7AC0A	Y:$7AC0E
		Chan[j].SerialEncDataD	Y:$7AC03	Y:$7AC07	Y:$7AC0B	Y:$7AC0F
ACC84E[12]	$7BC00	Chan[j].SerialEncDataA	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
		Chan[j].SerialEncDataB	Y:$7BC01	Y:$7BC05	Y:$7BC09	Y:$7BC0D
		Chan[j].SerialEncDataC	Y:$7BC02	Y:$7BC06	Y:$7BC0A	Y:$7BC0E
		Chan[j].SerialEncDataD	Y:$7BC03	Y:$7BC07	Y:$7BC0B	Y:$7BC0F
ACC84E[1]	$78D00	Chan[j].SerialEncDataA	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
		Chan[j].SerialEncDataB	Y:$78D01	Y:$78D05	Y:$78D09	Y:$78D0D
		Chan[j].SerialEncDataC	Y:$78D02	Y:$78D06	Y:$78D0A	Y:$78D0E
		Chan[j].SerialEncDataD	Y:$78D03	Y:$78D07	Y:$78D0B	Y:$78D0F
ACC84E[5]	$79D00	Chan[j].SerialEncDataA	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
		Chan[j].SerialEncDataB	Y:$79D01	Y:$79D05	Y:$79D09	Y:$79D0D
		Chan[j].SerialEncDataC	Y:$79D02	Y:$79D06	Y:$79D0A	Y:$79D0E
		Chan[j].SerialEncDataD	Y:$79D03	Y:$79D07	Y:$79D0B	Y:$79D0F
ACC84E[9]	$7AD00	Chan[j].SerialEncDataA	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
		Chan[j].SerialEncDataB	Y:$7AD01	Y:$7AD05	Y:$7AD09	Y:$7AD0D
		Chan[j].SerialEncDataC	Y:$7AD02	Y:$7AD06	Y:$7AD0A	Y:$7AD0E
		Chan[j].SerialEncDataD	Y:$7AD03	Y:$7AD07	Y:$7AD0B	Y:$7AD0F
ACC84E[13]	$7BD00	Chan[j].SerialEncDataA	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
		Chan[j].SerialEncDataB	Y:$7BD01	Y:$7BD05	Y:$7BD09	Y:$7BD0D
		Chan[j].SerialEncDataC	Y:$7BD02	Y:$7BD06	Y:$7BD0A	Y:$7BD0E
		Chan[j].SerialEncDataD	Y:$7BD03	Y:$7BD07	Y:$7BD0B	Y:$7BD0F
ACC84E[2]	$78E00	Chan[j].SerialEncDataA	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
		Chan[j].SerialEncDataB	Y:$78E01	Y:$78E05	Y:$78E08	Y:$78E0D
		Chan[j].SerialEncDataC	Y:$78E02	Y:$78E06	Y:$78E09	Y:$78E0E
		Chan[j].SerialEncDataD	Y:$78E03	Y:$78E07	Y:$78E0A	Y:$78E0F
ACC84E[6]	$79E00	Chan[j].SerialEncDataA	Y:$79E00	Y:$79E04	Y:$79E0B	Y:$79E0C
		Chan[j].SerialEncDataB	Y:$79E01	Y:$79E05	Y:$79E0C	Y:$79E0D
		Chan[j].SerialEncDataC	Y:$79E02	Y:$79E06	Y:$79E0D	Y:$79E0E
		Chan[j].SerialEncDataD	Y:$79E03	Y:$79E07	Y:$79E0E	Y:$79E0F
ACC84E[10]	$7AE00	Chan[j].SerialEncDataA	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
		Chan[j].SerialEncDataB	Y:$7AE01	Y:$7AE05	Y:$7AE09	Y:$7AE0D
		Chan[j].SerialEncDataC	Y:$7AE02	Y:$7AE06	Y:$7AE0A	Y:$7AE0E
		Chan[j].SerialEncDataD	Y:$7AE03	Y:$7AE07	Y:$7AE0B	Y:$7AE0F
ACC84E[14]	$7BE00	Chan[j].SerialEncDataA	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C
		Chan[j].SerialEncDataB	Y:$7BE01	Y:$7BE05	Y:$7BE09	Y:$7BE0D
		Chan[j].SerialEncDataC	Y:$7BE02	Y:$7BE06	Y:$7BE0A	Y:$7BE0E
		Chan[j].SerialEncDataD	Y:$7BE03	Y:$7BE07	Y:$7BE0B	Y:$7BE0F

SSI Protocol

For an SSI encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	P 23	P 22	P 21	P 20	P 19	P 18	P 17	P 16	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0	-	-
Component:	*Single/Multi-Turn Position*																									

Acc84E[*i*].Chan[*j*].SerialEncDataB is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Data	E 0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	P 31	P 30	P 29	P 28	P 27	P 26	P 25	P 24	-	-
Component:	*PE*																*Single/Multi-Turn Position*									

Acc84E[*i*].Chan[*j*].SerialEncDataC and **Acc84E[*i*].Chan[*j*].SerialEncDataD** status registers are not used in SSI Protocol.

Bits P*n* represent the bits of single-turn and multi-turn position.
Bit E0 represents the parity error bit.

EnDat 2.1/2.2 Protocol

For an EnDat 2.1/2.2 encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	P 23	P 22	P 21	P 20	P 19	P 18	P 17	P 16	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0	-	-

Component: *Single/Multi-Turn Position*

Bit	Bit Data	Component	Description
[23:0]	P*n*	-	Single/Multi-Turn Position

Acc84E[*i*].Chan[*j*].SerialEncDataB is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Data	E 5	E 4	E 3	E 2	E 1	E 0	-	-	P 39	P 38	P 37	P 36	P 35	P 34	P 33	P 32	P 31	P 30	P 29	P 28	P 27	P 26	P 25	P 24	-	-

Component: *TE CE CE1 CE2 EB1 EB2* *Single/Multi-Turn Position*

Bit	Bit Data	Component	Description
[15:0]	P*n*	-	Single/Multi-Turn Position
18	E0	EB2	"Error bit 1" reported by the encoder [2.2 only]
19	E1	EB1	"Error bit 2" reported by the encoder
20	E2	CE2	CRC error detected by the IC for the 2nd additional information word [2.2 only]
21	E3	CE1	CRC error detected by the IC for the 1st additional information word [2.2 only]
22	E4	CE	CRC error detected by the IC for the position information word
23	E5	TE	Timeout error detected by the IC

For the EnDat2.2 protocol, **Acc84E[*i*].Chan[*j*].SerialEncDataC** is used for the first additional information word if this is requested of the encoder with an MRS code in **Acc84E[*i*].Chan[*j*].SerialEncCmd**. This register is never used with the EnDat2.1 protocol.

For an EnDat 2.2 encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataC** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Data	W 0	R 0	B 0	A 4	A 3	A 2	A 1	A 0	I 15	I 14	I 13	I 12	I 11	I 10	I 9	I 8	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0	-	-

Component: *WN RM BY MRS Acknowledge* *Additional Information 1 Word*

Bit R0 is the "RM" (reference mark) status bit.
Bit W0 is the "warning" status bit.

Bit	Bit Data	Component	Description
[15:0]	I*n*	-	*Additional Information 1 Word*
[20:16]	A*n*	MRS Acknowledge	An represent acknowledgement bits for the MRS code in Acc84E[i].Chan[j].SerialEncCmd. The value of the acknowledgement code is $40 (64) less than the value of the commanded MRS code. For example, if the MRS code is $42 (66), the acknowledgement code is $02 (2).
21	B0	BY	Busy status bit
22	R0	RM	Reference mark detection status bit.
23	W0	WN	Warning status bit

The following table provides details of the information and acknowledgement words for each of the MRS command codes for the 1st additional information word.

Serial Encoder Data Register C: Additional Information 1				
Command code	Acknowledgement of MRS Code Bit[20:16], decimal	Information/Type	Byte 1 [I15:I8]	Byte 2 [I7:I0]
$41	1	Diagnosis	Address	Data
$42	2	Position Value 2 Word 1 LSB	MSB data	LSB data
$43	3	Position Value 2 Word 2	MSB data	LSB data
$44	4	Position Value 2 Word 3 MSB	MSB data	LSB data
$45	5	Memory parameter	Address	LSB data
$46	6	Memory parameter	Address	MSB data
$47	7	MRS Code	MRS Code	Any
$48	8	Acknowledge of test command	Port Address	Any
$49	9	Test values word 1 LSB	MSB data	LSB data
$4A	10	Test values word 2	MSB data	LSB data
$4B	11	Test values word 3 MSB	MSB data	LSB data
$4C	12	Temperature sensor 1	MSB data	LSB data
$4D	13	Temperature sensor 2	MSB data	LSB data
$4F	15	Stop additional information 1	Any	Any

For the EnDat2.2 protocol, **Acc84E[*i*].Chan[*j*].SerialEncDataD** is used for the second additional information word if this is requested of the encoder with an MRS code in **Acc84E[*i*].Chan[*j*].SerialEncCmd**. This register is never used with the EnDat2.1 protocol. For an EnDat 2.2 encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataD** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Value	W 0	R 0	B 0	A 4	A 3	A 2	A 1	A 0	I 15	I 14	I 13	I 12	I 11	I 10	I 9	I 8	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0	-	-

Component: WN RM BY MRS Acknowledge Additional Information 2 Word

Bit	Bit Data	Component	Description
[15:0]	In	-	Additional Information 2 Word
[20:16]	An	MRS Acknowledge	An represent bits of the acknowledgement of the MRS code in Acc84E[i].Chan[j].SerialEncCmd. The value of the acknowledgement code is $40 (64) less than the value of the commanded MRS code. For example, if the MRS code is $53 (83), the acknowledgement code is $13 (19).
21	B0	BY	Busy status bit
22	R0	RM	Reference mark detection status bit.
23	W0	WN	Warning status bit

The following table provides details of the information and acknowledgement words for each of the MRS command codes for the 2nd additional information word.

Serial Encoder Data Register D: Additional Information 2

Command code	Acknowledgement of MRS Code Bit[20:16], binary	Information/Type	Byte 1 [15:8]				Byte 2 [7:0]
$51	17	Commutation	U	V	W	Not assigned	Not assigned
			15	14	13	[12:8]	
$52	18	Acceleration	MSB data				LSB data
$53	19	Commutation & Acceleration	U	U	W	MSB Acceleration data	LSB acceleration data
			15	14	13	[12:8]	
$54	20	Limit position signals	L1	L2	Not assigned		Not assigned
			15	14	[13:8]		
$55	21	Limit position signals & Acceleration	L1	L2		MSB Acceleration data	LSB acceleration data
			15	14	13	[12:8]	
$5F	31	Stop additional information 2	Any				Any

When used in the Script environment, **Acc84E[*i*].Chan[*j*].SerialEncDataD** is a 24-bit element. When used in the C environment, it is a 32-bit element, with real data in the high 24 bits, so its value in the C environment is 256 times greater than its value in the Script environment.

In Power PMAC, **Acc84E[*i*].Chan[*j*].SerialEncDataD** will report as “nan” (not-a-number) if no board with this index is present.

Yaskawa Sigma II/III/V Protocol

For the Yaskawa Sigma II/III/V protocol, the data format in this element depends on the particular type of the encoder and its reporting mode.

Yaskawa Sigma II absolute encoder with 17 bits per revolution

For an Absolute Yaskawa Sigma II encoder with 17 bits per revolution and 16 bits of turns count in position-reporting (P1) mode, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	M 2	M 1	M 0	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-	-	-	-	-

Component: *Multi-Turn Pos* *Single-Turn Position*

Bit	Bit Data	Component	Description
[20:4]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position
[23:21]	M*n*	Multi-Turn Position	Bits Mn represent bits of the multi-turn position.

Acc84E[*i*].Chan[*j*].SerialEncDataB is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Data	E 2	E 1	E 0	-	-	-	-	-	-	-	-	M 15	M 14	M 13	M 12	M 11	M 10	M 9	M 8	M 7	M 6	M 5	M 4	M 3	-	-

Component: *TE CE EB* *Multi-Turn Position*

Bit	Bit Data	Component	Description
[12:0]	M*n*	Multi-Turn Position	Bits Mn represent the bits of multi-turn position.
21	E0	EB	Coding error reported by the encoder
22	E1	CE	CRC error detected by the IC
23	E2	TE	Timeout error detected by the IC

Yaskawa Sigma II incremental encoder with 17 bits per revolution

For an in position-reporting (P1) mode, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	-	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-	U	V	W	Z	-	-

Component: *Single-Turn Position* *Hall & Index*

Bit	Bit Data	Component	Description
0	Z	Index	Z represents the encoder's "zero" (index) pulse marker signal state.
[3:1]	UVW	Hall	U, V, and W represent the commutation "Hall" sensor signal states
[22:6]	S*n*	Single-turn Position	Bits Sn represent the bits of single-turn position

Acc84E[*i*].Chan[*j*].SerialEncDataB is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	E 2	E 1	E 0	-	-	-	-	-	-	-	-	-	-	C 16	C 15	C 14	C 13	C 12	C 11	C 10	C 9	C 8	C 7	C 6	-	-

Component: *TE CE EB* *Compensation Position*

Bit	Bit Data	Component	Description
[10:0]	C*n*	Compensation Position	Bits Cn represent the bits of "compensation" position, captured on the first index pulse.
21	E0	EB	Coding error reported by the encoder
22	E1	CE	CRC error detected by the IC
23	E2	TE	Timeout error detected by the IC

Yaskawa Sigma III or V absolute encoder with 20 bits per revolution

For an Absolute Yaskawa Sigma III or V encoder with 20 bits per revolution and 16 bits of turns count in position-reporting (P1) mode, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	S 19	S 18	S 17	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-	-	-	-	-

Component: *Single-Turn Position*

Bits S*n* represent the bits of single-turn position.

Bit	Bit Data	Component	Description
[23:4]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position

Acc84E[*i*].Chan[*j*].SerialEncDataB is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Data	E 2	E 1	E 0	-	-	-	-	-	M 15	M 14	M 13	M 12	M 11	M 10	M 9	M 8	M 7	M 6	M 5	M 4	M 3	M 2	M 1	M 0	-	-
Component:	*TE*	*CE*	*EB*														*Multi-Turn Position*									

Bit	Bit Data	Component	Description
[15:0]	M*n*	Multi-Turn Position	Bits Mn represent the bits of multi-turn position.
21	E0	EB	Coding error reported by the encoder
22	E1	CE	CRC error detected by the IC
23	E2	TE	Timeout error detected by the IC

For a Yaskawa Sigma II/III/V encoder in position-reporting (P1) mode, **Acc84E[*i*].Chan[*j*].SerialEncDataC** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	A7	A6	A5	A4	A3	A2	A1	A0	T7	T6	T5	T4	T3	T2	T1	T0	-	-
Component:									*Alarm Code*								*Temperature*									

Bits T*n* represent bits of the returned temperature value (in degrees C); bits A*n* represent bits of the alarm code.

For an absolute encoder, the alarm-code bits have the following meanings:

Bit	Error Name	Type	Alarm Type	Clear Action	Notes
8	Backup Battery Alarm	Alarm	EEPROM	RESET command	Backup battery was down, internal data was lost
9	Encoder Error	Alarm	EEPROM	RESET command	Error in encoder
10	Battery Level Warning	Warning	Flag	-	Battery voltage drop
11	Absolute Error	Alarm	Session Flag	Power cycle	Possible error in position, if doesn't get cleared in one revolution
12	Over Speed	Alarm	Session Flag	Power cycle	
13	Overheat	Alarm	Flag	-	
14	Reset Complete	Warning	Session Flag	Power cycle	Indicates a reset due to RESET command
15	Fixed at "0"				Set at zero

For an incremental encoder, the alarm-code bits have the following meanings:

Bit	Error Name	Type	Alarm Type	Clear Action	Notes
8	Fixed at "1"	-	-	-	-
9	Encoder Error	Alarm	Session Flag	Power cycle	Error in encoder
10	Fixed at "0"	-	-	-	-
11	Position Error	Alarm	Session Flag	Power cycle	Possible error in position or Hall sensor
12	Fixed at "0"	-	-	-	
13	Fixed at "0"	-	-	-	
14	Origin not passed flag	Warning	-	-	The origin has not been passed in this session yet
15	Fixed at "0"				Set at zero

The temperature comes back in centigrade units but it rolls over to negative numbers at 215°C. Here is an example of how to read the information:

```
if (Temp<=215)
{
       Temperature = Temp;
}
Else
{
       Temperature = Temp - 256;
}
```

Tamagawa Protocol

For a Tamagawa FA-Coder encoder with 17 bits per revolution, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0	-	-
Component:								*Single-Turn Position*																		

Bit	Bit Data	Component	Description
[16:0]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position

For a Tamagawa FA-Coder encoder with 16 bits of multi-turn count, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	-	-
Component:									*Multi-Turn Position*																	

Bit	Bit Data	Component	Description
[16:0]	M*n*	Multi-turn Position	Bits Mn represent the bits of multi-turn position.

For a Tamagawa encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataC** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	E1	E0	-	-	S7	S6	S5	S4	A7	A6	A5	A4	A3	A2	A1	A0	I7	I6	I5	I4	I3	I2	I1	I0	-	-
Component:	*TE*	*CE*			*Status Code*				*Alarm Code*								*Encoder ID*									

Bit	Bit Data	Component	Description
[7:0]	In	Encoder ID	Bits In represent bits of the returned encoder ID code (Fixed to $11)
[15:8]	An	Alarm Code	Bits An represent bits of the alarm code
[19:16]	Sn	Status Code	Bits Sn represent bits of the status code
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

Alarm Code Description

Bit	Bit Data	State when error is occurred	Description
8	A0	1	Over-speed (OS) When the shaft of the encoder is rotated over the electric spec of Multi-Turn signal, after Main Power is turned off and during External Battery is on, logic "1." is generated and can be transmitted after Main Power is turned on. As this error may not be detected, it is useful for only ref. Use error reset mode(s) to clear this latched error.
9	A1	1	Full Absolute Status (FS) When Main Power, is turned on during the shaft of the encoder is rotated at more than 100 rpm, logic "1" comes out. The accuracy of One Revolution data is 5 bits during logic "1" comes out. As One Revolution data returns to 17 bits resolution, this error status is automatically released. For clearing the alarm bit, slow rotational speed down less than 100* rpm, and wait to release the error status.
10	A2	1	Counting Error (CE) When One Revolution data is wrong because of any mal-function or defect during Main Power is on, logic "1" comes out depending upon the following I or II. I. Error is detected at each mechanical angle of 45 during the shaft rotational speed is more than 100 rpm. The error status is automatically released by returning the deviation of mechanical angle within ± 22.5 ' (typ.). II. Error is always detected during the shaft rotational speed is less than 100 rpm. If the deviation of mechanical angle is more than ± 0.7 ' (typ.), logic "1" comes out. In this case, use error reset mode(s) to clear this latched error.
11	A3	1	Counter Over-flow (OF) When Multi-Turn counter is overflown, logic "1" comes out. Detecting it during Main Power is off, it can be transmitted after Main Power is turned on. Detecting it once, it is kept till Error Reset. While the counter counts 0-65,535 cyclically.
12	A4	0	-
13	A5	1	Multi-turn Error (ME) When any bit of Multi-Turn signal is jumped during Main Power is on, logic "1" comes out. During Main Power is off, it is not executed. The check for bit jumping is performed in each 12.8μsec. Use error reset mode(s) to clear this latched error.
14	A6	1	Battery Error (BE) When the voltage of capacitor integrated in the encoder is 2.5 ± 0.2 V or less during Main Power is off, logic "1" is generated and can be transmitted after Main Power is turned on. Multi- Turn error nay be occurred at same time with it. Error Reset and Multi-turn Data Reset. Needed to check or replace Battery.
15	A7	1	Battery Alarm (BA) When the voltage of External Battery is 3.1 ±0.1 V or less during Main Power is on, logic "1" comes out. Returning the voltage of External battery to normal, the error status is automatically released.

Status Code Description

Bit	Bit Data	State when error is occurred	Description
16	S4	1	ca1: Demiliter error in Request Frame
17	S5	1	ca0 Parity error in Request Frame.
18	S6	1	ea1: Logic OR of Multi-turn error, Battery error or Battery alarm (check alarm word for identification of exact alarm)
19	S7	1	ea0: Counting error

Panasonic Protocol

For a Panasonic encoder with 17 bits per revolution, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows (if single turn data has more resolution, higher bits include data):

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-
Component:								*Single-Turn Position*																		

Bit	Bit Data	Component	Description
[16:0]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position

For a Panasonic encoder with 16 bits of multi-turn count, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	M 15	M 14	M 13	M 12	M 11	M 10	M 9	M 8	M 7	M 6	M 5	M 4	M 3	M 2	M 1	M 0	-	-
Component:									*Multi-Turn Position*																	

Bit	Bit Data	Component	Description
[16:0]	M*n*	Multi-turn Position	Bits Mn represent the bits of multi-turn position. Multi-turn position is only reported if the SerialEncCmdWord component of **Acc84E[*i*].Chan[*j*].SerialEncCmd** is set to $2A, in which case the encoder ID and alarm code are not reported.

Acc84E[*i*].Chan[*j*].SerialEncDataC is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	E 1	E 0	-	-	S 7	S 6	S 5	S 4	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0	-	-
Component:	*TE*	*CE*			*Status Code*				*Alarm Code*								*Encoder ID*									

Bit	Bit Data	Component	Description
[7:0]	I*n*	Encoder ID	Bits In represent bits of the returned encoder ID code (Fiexd to $11)
[15:8]	A*n*	Alarm Code	Bits An represent bits of the alarm code
[19:16]	S*n*	Status Code	Bits Sn represent bits of the status code
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

The encoder ID and alarm code values are only reported if the *SerialEncCmdWord* component of **Acc84E[*i*].Chan[*j*].SerialEncCmd** is set to $52, in which case multi-turn data is not reported.

Alarm Code Description

Bit	Bit Data	State when error is occurred	Description
8	A0	1	Over-speed (OS) When the shaft of the encoder is rotated over the electric spec of Multi-Turn signal, after Main Power is turned off and during External Battery is on, logic "1." is generated and can be transmitted after Main Power is turned on. As this error may not be detected, it is useful for only ref. Use error reset mode(s) to clear this latched error.
9	A1	1	Full Absolute Status (FS) When Main Power, is turned on during the shaft of the encoder is rotated at more than 100 rpm, logic "1" comes out. The accuracy of One Revolution data is 5 bits during logic "1" comes out. As One Revolution data returns to 17 bits resolution, this error status is automatically released. For clearing the alarm bit, slow rotational speed down less than 100* rpm, and wait to release the error status.
10	A2	1	Counting Error (CE) When One Revolution data is wrong because of any mal-function or defect during Main Power is on, logic "1" comes out depending upon the following I or II. I. Error is detected at each mechanical angle of 45 during the shaft rotational speed is more than 100 rpm. The error status is automatically released by returning the deviation of mechanical angle within ± 22.5 ' (typ.). II. Error is always detected during the shaft rotational speed is less than 100 rpm. If the deviation of mechanical angle is more than ± 0.7 ' (typ.), logic "1" comes out. In this case, use error reset mode(s) to clear this latched error.
11	A3	1	Counter Over-flow (OF) When Multi-Turn counter is overflown, logic "1" comes out. Detecting it during Main Power is off, it can be transmitted after Main Power is turned on. Detecting it once, it is kept till Error Reset. While the counter counts 0-65,535 cyclically.
12	A4	0	-
13	A5	1	Multi-turn Error (ME) When any bit of Multi-Turn signal is jumped during Main Power is on, logic "1" comes out. During Main Power is off, it is not executed. The check for bit jumping is performed in each 12.8µsec. Use error reset mode(s) to clear this latched error.
14	A6	1	Battery Error (BE) When the voltage of capacitor integrated in the encoder is 2.5 ± 0.2 V or less during Main Power is off, logic "1" is generated and can be transmitted after Main Power is turned on. Multi- Turn error nay be occurred at same time with it. Error Reset and Multi-turn Data Reset. Needed to check or replace Battery.
15	A7	1	Battery Alarm (BA) When the voltage of External Battery is 3.1 ±0.1 V or less during Main Power is on, logic "1" comes out. Returning the voltage of External battery to normal, the error status is automatically released.

Status Code Description

Bit	Bit Data	State when error is occurred	Description
16	S4	0	0
17	S5	0	0
18	S6	1	ea1: Logic OR of Multi-turn error, Battery error or Battery alarm (check alarm word for identification of exact alarm)
19	S7	1	ea0: System Down

Mutitoyo Protocol

For a Mitutoyo encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P 23	P 22	P 21	P 20	P 19	P 18	P 17	P 16	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0	-	-
Component:	*Single/Multi-Turn Position*																									

Bit	Bit Data	Component	Description
[23:0]	P*n*	Position Data	Bits Pn represent the bits of single-turn and multi-turn position.

For a Mitutoyo encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	P 31	P 30	P 29	P 28	P 27	P 26	P 25	P 24	-	-
Component:																	*Single/Multi-Turn Position*									

Bit	Bit Data	Component	Description
[7:0]	P*n*	Position Data	Bits Pn represent the bits of single-turn and multi-turn position.

Acc84E[*i*].Chan[*j*].SerialEncDataC is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	E 1	E 0	-	-	S 7	S 6	S 5	S 4	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0	-	-
Component:	*TE*	*CE*			*Status Code*				*Alarm Code*								*Encoder ID*									

Bit	Bit Data	Component	Description
[7:0]	I*n*	Encoder ID	Bits In represent bits of the returned encoder ID code (Fixed to $11)
[15:8]	A*n*	Alarm Code	Bits An represent bits of the alarm code
[19:16]	S*n*	Status Code	Bits Sn represent bits of the status code
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

Alarm Code Description for AT303A Encoders

Bit	Bit Data	State when error is occurred	Description
8	A0	1	Initialized error
9	A1	1	Disagreement data of photoelectric type and capacitance type
10	A2	1	Photoelectric error
11	A3	1	Electrostatic capacitance error
12	A4	1	CPU error
13	A5	1	EEPROM error
14	A6	1	ROM, RAM error
15	A7	1	Overspeed

Alarm Code Description for AT503A Encoders

Bit	Bit Data	State when error is occurred	Description
8	A0	1	Initialized error
9	A1	1	Disagreement data of photoelectric type and capacitance type
10	A2	1	Photoelectric error
11	A3	1	Electrostatic capacitance error
12	A4	1	CPU error / ROM, RAM error
13	A5	1	EEPROM error
14	A6	1	Communication error
15	A7	1	Overspeed

Status Code Description

Bit	Bit Data	State when error is occurred	Description
16	S4	1	System error Set “1 ” if fatal error generates in encoder. If this error occurs, turning off the servo is necessary because data itself may have problems. The scale needs restart.(Either cycle power or execute the reset process using *SerialEncCmdWord* set to $89 and follow the instructions in the Single-Channel Setup Element section)
17	S5	0	Fixed “0”
18	S6	1	Communication error Set “1 ” if mistake the data request (command field) cmd from controller. At this case, transmit a request data by setting *SerialEncCmdWord* set to $01. If there are not mistake in next command field, set “0".
19	S7	0	Fixed “0”

BiSS-B/C Protocol

For a BiSS-B/C encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P 23	P 22	P 21	P 20	P 19	P 18	P 17	P 16	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0	-	-
Component:	*Single/Multi-Turn Position*																									

Bit	Bit Data	Component	Description
[23:0]	P*n*	Position Data	Bits Pn represent the bits of single-turn and multi-turn position.

For a BiSS-B/C encoder, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7-4	3-0
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	-	-
Bit Value	E 1	E 0	S 5	S 4	S 3	S 2	S 1	S 0	P 39	P 38	P 37	P 36	P 35	P 34	P 33	P 32	P 31	P 30	P 29	P 28	P 27	P 26	P 25	P 24	-	-
Component:	*TE*	*CE*							*Single/Multi-Turn Position*																	

Bit	Bit Data	Component	Description
[15:0]	P*n*	Position Data	Bits Pn represent the bits of single-turn and multi-turn position.
[21:16]	S*n*	Status Bits	Bits Sn represent encoder-specific status bits.
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

Acc84E[*i*].Chan[*j*].SerialEncDataC and **Acc84E[*i*].Chan[*j*].SerialEncDataD** status registers are not used in BiSS B/C Protocol.

Status Code Description for Renishaw BiSS encoders

Bit	Bit Data	State when error is occurred	Description
16	S0	0 (Active Low)	Warning: The Warning bit indicates that the encoder scale should be cleaned.
17	S1	0 (Active Low)	Error: The Error bit indicates that the absolute position data may not be valid, or the temperature is above the maximum operating temperature of the encoder.

Matsushita Protocol

For a Matsushita encoder with 17 bits per revolution, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows (if single turn data has more resolution, higher bits include data):

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	M 6	M 5	M 4	M 3	M 2	M 1	M 0	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-
Component:	*Multi-Turn Position*							*Single-Turn Position*																		

Bit	Bit Data	Component	Description
[16:0]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position
[23:17]	M*n*	Multi-Turn Position	Bits Mn represent the bits of multi-turn position

For a Panasonic encoder with 16 bits of multi-turn count, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M 14	M 13	M 12	M 11	M 10	M 9	M 8	M 7	-	-
Component:																	*Multi-Turn Position*									

Bit	Bit Data	Component	Description
[7:0]	M*n*	Multi-turn Position	Bits Mn represent the bits of multi-turn position.

Acc84E[*i*].Chan[*j*].SerialEncDataC is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	E 1	E 0	-	-	-	-	-	-	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	S 3	S 2	S 1	S 0	I 3	I 2	I 1	I 0	-	-
Component:	*TE*	*CE*							*Alarm Code*								*Status Code*				*Encoder ID*					

Bit	Bit Data	Component	Description
[7:0]	I*n*	Encoder ID	Bits In represent bits of the returned encoder ID code (Fiexd to $11)
[15:8]	A*n*	Alarm Code	Bits An represent bits of the alarm code
[19:16]	S*n*	Status Code	Bits Sn represent bits of the status code
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

The encoder ID and alarm code values are only reported if the *SerialEncCmdWord* component of **Acc84E[*i*].Chan[*j*].SerialEncCmd** is set to $52, in which case multi-turn data is not reported.

Alarm Code Description

Bit	Bit Data	State when error is occurred	Description
8	A0	1	Over-speed (OS) Function: When revolving speed exceeds (6600r/min), "1" is output. Detecting timing: During normal operation Output: Latch Reset: Reset I or Re-turn ON the main power supply.
9	A1	1	Preset Status (PS): Function: Function: In case a logical error is included in scan data of M-sensors; or, in case scan data detected three times do not coincide each other, "1" is output. During that period or time, every data is unfixed. After the scanning has completed properly and the data has been established it i3 reset to "0". Detection timing: During normal operation (at preset initial operation only) Output: Non latch Reset: Auto reset (Revolving speed is reduced to 300t/min or less) When ABS signal fails to be read correctly power supply ON, due to a problem of the encoder at such a foreign object or a failure of optical sensor etc., even when revolting speed lowers 300 r/min or less, it may not be reset to "0".
10	A2	1	Count Error 1 (CE1) Function: Specific 1 bit of M-sensors and specific. 1 bit within ASIC are compared and when the data are different from each other, "1" is output. Detecting timing: During normal operation Output: Latch Reset: Re-turn ON the main power supply.
11	A3	1	Count Error 2 (CE2) Function: Consistency between "multi-turn count block" and "single-turn count block" is monitored. To be more concretely; 1. Multi-turn count obtained by magnetic encoder 2. Multi-turn count obtained proximately by processing carry/borrow on single-turn absolute value By comparing 1 and 2 above, in case (difference between 1 and 2) >= 2 turns, on alarm is emitted. Detecting timing: During normal operation Output: Latch Reset: Re-turn ON the main power supply.
12	A4	0	-
13	A5	1	Overflow (OF) Function: In case amount of revolution exceeds "-16383- + 16382", "1" is output. Alter overflow, the multi-tum counter functions as cyclic counter overflow flag allows to set mask to encoder error bit: ea0 on output field. Detecting timing: During normal/backup operation Output: latch Reset: Reset II
14	A6	1	System Down (SD) Function: When an encoder gets into an emergency operation status and disables to perform its function, "1" is output To be more concretely, voltage of the backup capacitor within the encoder towers 2.8 V (TYP) or less. Detecting timing: During backup operation Output: Latch Reset: Reset I (External battery was be checked or replaced with an new one.)
15	A7	1	Battery Alarm (BA) Function: When voltage of the external battery decreases below 3.0V (TYP), "1" is outputted. Detecting timing: During backup operation Output Latch Reset: Reset I (External battery was be checked or replaced with an new one.)

Status Code Description

Bit	Bit Data	State when error is occurred	Description
4	S0	1	ea0: Logic OR of Over-speed (OS), System Down (SD), Batter Alarm (BA)
5	S1	1	ea1: Logic OR of Count Error 1 (CE1) and Count Error 2 (CE2) (check alarm word for identification of exact alarm)
6	S2	0	0
7	S3	1	Preset Status (PS)

`

Mitsubishi Protocol

Note

Mitsubishi Serial Encoder on HG-□ type servo motors can only be queried at 55.5µsec±1.0µsec (18 kHz), 111µsec±1.0µsec (9 kHz) and 222µsec±1.0µsec (4.5 kHz). If the request cycle is other than the above cycles the data will not be latched properly.

For a Mitsubishi HG-□ encoder with 17 bits per revolution, **Acc84E[*i*].Chan[*j*].SerialEncDataA** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	S 23	S 22	S 21	S 20	S 19	S 18	S 17	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-
Component:	*Single-Turn Position*																									

Bit	Bit Data	Component	Description
[23:0]	S*n*	Single-Turn Position	Bits Sn represent the bits of single-turn position

Depending on the *SerialEncCmdWord* in single channel setup register, the single turn data is reported in 3 different modes:

Single-Turn Data in Lower 18 Bits of 24-Bit Word

$02 –single-turn data
$BA – for clearing alarms and reporting single-turn data
$2A – for reporting single-turn and multi-turn data

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	S 17	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-
Component:	*Single-Turn Position*																									

Single-Turn Data in Lower 20 Bits of 24-Bit Word

$A2 – for reporting single-turn and multi-turn data

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	S 17	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-	-	-
Component:	*Single-Turn Position*																									

Single-Turn Data in Upper 20 Bits of 24-Bit Word

$32 – for reporting single-turn and multi-turn data

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	S 17	S 16	S 15	S 14	S 13	S 12	S 11	S 10	S 9	S 8	S 7	S 6	S 5	S 4	S 3	S 2	S 1	S 0	-	-	-	-	-	-	-	-
Component:	*Single-Turn Position*																									

This method of reporting provides the best contiguous data between single-turn and multi-turn data which can be very useful in both Turbo and Power PMAC Power-on Servo Position retrieval process using built-in functionality.

For a Mitsubishi HG-□ encoder with 16 bits of multi-turn count, **Acc84E[*i*].Chan[*j*].SerialEncDataB** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	-	-	-	-	-	-	-	-	M 15	M 14	M 13	M 12	M 11	M 10	M 9	M 8	M 7	M 6	M 5	M 4	M 3	M 2	M 1	M 0	-	-
Component:									*Multi-Turn Position*																	

Bit	Bit Data	Component	Description
[16:0]	M*n*	Multi-turn Position	Bits Mn represent the bits of multi-turn position.

For a Mitsubishi HG-□, **Acc84E[*i*].Chan[*j*].SerialEncDataC** is configured as follows:

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	E 1	E 0	-	-	S 7	S 6	S 5	S 4	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0	-	-
Component:	*TE*	*CE*			*Status Code*				*Alarm Code*								*Encoder ID*									

Bit	Bit Data	Component	Description
[7:0]	I*n*	Encoder ID	Bits In represent bits of the returned encoder ID code. The Encoder ID is transmitted by the encoder for *SerialEncCmdWord* codes $92 and $7A. After issuing one of those two commands, the Encoder ID is latched by the ACC-84E and provided in bits [07:00] of this register. Once latched, it is always readable in this register.
[15:8]	A*n*	Alarm Code	Bits An represent bits of the alarm code. The Alarm Code is not transmitted by the encoder for *SerialEncCmdWord* codes $92 and $7A. Bits [15:08] are cleared for these two commands.
[19:16]	S*n*	Status Code	Bits Sn represent bits of the status code. The 4-bit data from the Status Field (SF) is returned for all command codes.
22	E0	CE	CRC error detected by the IC
23	E1	TE	Timeout error detected by the IC

Alarm Code Description

Bit	Bit Data	State when error is occurred	Description			
			Alarm Name	Check timing	Details	Action
8	A0	1	CPU Alarm	Checked at power on (latched)	Encoder internal data damaged (latched alarm)	Stop operation
9	A1	0	-	-	-	-
10	A2	1	Data Alarm	Checked at each request	Data per revolution error (Cleared when cause of alarm is eliminated)	Stop operation when received. Discard received data
11	A3	1	Encoder Thermal Alarm	Check during operation (10 second average)	Encoder section hot (100±5°C) (Cleared when cause of alarm is eliminated)	Stop operation
12	A4	1	Encoder Thermal Warning	Check during operation (30 minute continuous average)	Encoder section hot (85±5°C) (Cleared when cause of alarm is eliminated)	Only warning, no need to stop operation. Take necessary action to cool down encoder environment
13	A5	1	Multi-Revolution Alarm	Checked at each request	Multi-revolution count data error	Only a warning. No need to stop operation. Note: at next power on, multi-revolution data may be deviated
14	A6	1	ABS Lost Alarm	Checked at power on	Multi-revolution backup data damaged (latched alarm)	Stop operation Clear with request $BA
15	A7	1	Battery Disconnected Alarm	Checked during operation (10-second average)	Battery cable disconnected. Battery voltage low (Cleared when cause of alarm is eliminated)	Only a warning. No need to stop operation. Note: at next power on, ABS lost alarm may occur

Status Code Description

Bit	Bit Data	State when error is occurred	Description
16	S4	1	ca1: Delimiter error in Request Frame
17	S5	1	ca0 Parity error in Request Frame.
18	S6	1	ea1: Logic OR of following error alarms: CPU Alarm, Data Alarm, Encoder Thermal Alarm, Multi-Revolution Alarm, ABS Lost Alarm (check alarm word for identification of exact alarm)
19	S7	1	ea0: Logic OR of following warning alarms: Encoder Thermal Warning, Battery Disconnected Alarm (check alarm word for identification of exact alarm)

USING THE RESULTING POSITION INFORMATION

Serial encoder position information is commonly used for both absolute power-on position and ongoing position, and both for the servo and commutation algorithms. The following sections discuss the general method of using the resulting position information for Power PMAC, Turbo PMAC and MACRO CPUs.

Using the ACC-84E with Power PMAC

Ongoing Commutation Phase Position

For the commutation algorithm's ongoing phase position, Power PMAC reads the entire 32-bit register specified by Motor[x].pPhasePos every phase cycle. In order to be able to handle rollover of this data properly, the most significant bit (MSB) of this data must end up in bit 31 of the 32-bit result, shifted if necessary. With most protocols, no shifting is necessary, but some will require a net "left shift" to achieve this result.

To use serial encoder position from an ACC-84E FPGA-based interface for ongoing phase position, the following saved setup elements must be specified:

- **Motor[*x*].pPhaseEnc** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **Motor[*x*].PhaseEncRightShift** = 8// If encoder LSB in Register 8
- **Motor[*x*].PhaseEncLeftShift** = (*32* **-** *# of bits*)
- **Motor[*x*].PhasePosSf** = 2048 / (*Register LSBs per commutation cycle*)

In the 24-bit ACC-84E, the LSB of the encoder data is generally found in bit 8 on the 32-bit data bus, with unpredictable values in the lowest 8 bits of the bus. While this low "phantom" data is not known to affect actual commutation performance in real systems, some users will want to remove this data with an 8-bit "shift right" operation. When this is done, a "shift left" operation must also be done to leave the MSB of encoder data in bit 31 of the result. For purposes of computing the scale factor, the LSB of the resulting (post-shift) 32-bit value should be used as the "LSB".

Power-On Commutation Phase Position

Because most serial encoders provide absolute position information, especially over one motor revolution, they are commonly used to provide the absolute rotor-angle position at power-up for the commutation algorithms. Doing this requires assigning proper values to several saved setup elements.

This section gives an overview of those settings; details can be found in the element descriptions in the Software Reference Manual, the *Setting Up Commutation* chapter of the User's Manual, and the Hardware Reference Manual for the interface. In addition the motor setup routines in the IDE software will walk you through this setup.

To use serial encoder position from an ACC-84E FPGA-based interface for absolute power-on phase position, the following saved setup elements must be specified:

- **Motor[*x*].pAbsPhasePos** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **Motor[*x*].AbsPhasePosFormat** = $*aabbccdd*   // Protocol-specific settings
- **Motor[*x*].AbsPhasePosSf** = 2048 / (*LSBs per commutation cycle*)
- **Motor[*x*].AbsPhasePosOffset** = (*Difference between sensor zero and commutation zero*)

For the format variable, the LSB of the encoder data is typically found in bit 8 of the 32-bit register, and only enough bits to cover a single commutation cycle need to be used. (However, it does not hurt to specify more bits than are required.) It is seldom required to use data from the next register.

Ongoing Servo Position

To use the serial encoder position for ongoing servo position, the data must first be processed in the encoder conversion table. This is done with a "Type 1" single-register-read conversion from **SerialEncDataA**. In order to be able to handle rollover of this data properly, the most significant bit (MSB) of this data must end up in bit 31 of the 32-bit result, shifted if necessary. With most protocols, no shifting is necessary, but some will require a net "left shift" to achieve this result.

To use serial encoder position from an ACC-84E FPGA-based interface for ongoing servo position, the following saved setup elements must be specified:

- **EncTable[*n*].Type** = 1
- **EncTable[*n*].pEnc** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **EncTable[*n*].index1** = (*32 - # of bits*)   // Shift left # of bits
- **EncTable[*n*].index2** = 8   // Shift right # of bits
- **EncTable[*n*].ScaleFactor** = 1/ ($2^{32\text{ - \# of bits}}$) // For result in encoder LSBs
- **Motor[*x*].pEnc** = **EncTable[*n*].a** // Use table result for position-loop feedback
- **Motor[*x*].pEnc2** = **EncTable[*n*].a** // Use table result for velocity-loop feedback

Power-On Servo Position

Many serial encoders can provide absolute position over the entire range of travel of the motor. If so, Power PMAC can execute an absolute power-on read of the encoder to establish the reference position, eliminating the need for a homing search move.

This section gives an overview of those settings; details can be found in the element descriptions in the Software Reference Manual, the *Basic Motor Setup* chapter of the User's Manual, and the Hardware Reference Manual for the interface. In addition the motor setup routines in the IDE software will walk you through this setup.

To use serial encoder position from an ACC-84E FPGA-based interface for absolute power-on phase position, the following saved setup elements must be specified:

- **Motor[*x*].pAbsPos** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **Motor[*x*].AbsPosFormat** = $*aabbccdd* // Protocol-specific settings
- **Motor[*x*].AbsPosSf** = (*Motor units per sensor LSB*)
- **Motor[*x*].AbsPosOffset** = (*Difference between sensor zero and motor zero*)

For the format variable, the LSB of the encoder data is typically found in bit 8 of the 32-bit **SerialEncDataA** register. If the encoder provides more than 24 bits of absolute position data, the format element permits data from **SerialEncDataB** to be used as well. Note, however, that the data in **SerialEncDataA** must go all the way to bit 31 for this to work. In protocols such as Tamagawa and Panasonic, which provide only 17 bits of data in **SerialEncDataA** and more in **SerialEncDataB**, the full absolute position must be assembled in a user algorithm.

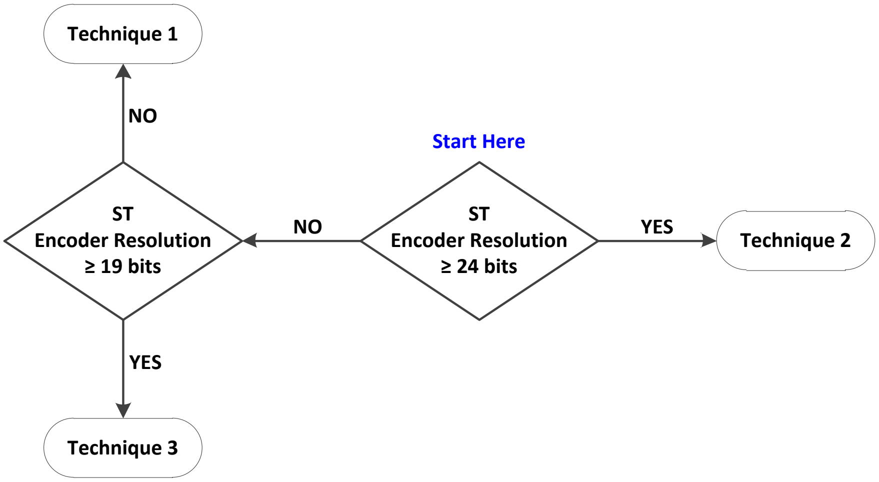
Using the ACC-84E with Turbo PMAC

In Turbo PMAC, the absolute serial encoder data is brought in as an unfiltered parallel Y-word into the Encoder Conversion Table (ECT) where it is processed for the PMAC to use for:

- On-going phase reference
- On-going position in the motor servo-loop
- Power-on absolute servo position
- Power-on phase reference

In general, encoder data is left-shifted 5 bits in the ECT to provide fractional data. This process can cause saturation of certain registers with higher resolution absolute serial encoders, thus for this type of encoders, it is recommended to process the data as unshifted. Moreover, special considerations need to be taken in setting up commutation (for commutated motors, e.g. brushless).

The following flowchart summarizes the recommended method to use, regardless of the Multi-turn (MT) data specification. It is only dependent on the Single-turn (ST) resolution (for rotary encoders) or protocol resolution (for linear scales).


Technique 1
NO
Start Here
ST Encoder Resolution ≥ 19 bits
NO
ST Encoder Resolution ≥ 24 bits
YES
Technique 2
YES
Technique 3

- Technique 1
  This technique places the Least Significant Bit (LSB) of the serial data in bit 5 of the result register providing the 5 bits of “non-existent” fraction.

- Technique 2
  This technique places the LSB of the serial data in bit 0 of the result register, creating no fractional bits. It requires a dedicated Encoder Conversion Table (ECT) entry for commutation.

- Technique 3
  This technique processes the data for position similarly to Technique 1, but it requires a dedicated ECT entry for commutation.

Note

Some applications may require deviating from the suggested setup methods (e.g. extremely high resolution and speed requirements). Contact Delta Tau for assistance with these special cases.

Setup Summary

Encoder Conversion Table Processing:

Process	Technique 1	Technique 2	Technique 3
ECT entry for Servo Loop Feedback	From serial register A, 5-bit shift	From serial register A, no shift	From serial register A, 5-bit shift
ECT entry for Commutation Feedback	N/A	From serial register A, 18 bits, no shift, Offset=ST-18	From serial register A, 18 bits, no shift, Offset=ST-18

Note

ST is the Single-turn resolution (in bits) for rotary encoders. Similarly, this would be the protocol resolution (in bits) for linear scales.

The position and velocity pointers are then assigned to the “ECT for position” result:

Parameter	Technique 1/2/3
Position (Ixx03)	@ ECT position result
Velocity (Ixx04)	@ ECT position result (typically, with single source feedback)

Commutation Source and Type (for commutated motors, e.g. brushless)

With technique 1, if the Single-turn + Multi-turn data bits fulfill 24 bits and are contiguous, then serial data register A can be used as the commutation source. Otherwise, the resulting register from the ECT for position is used for commutation (requires special settings for the commutation cycle size).
With techniques 2 and 3, the feedback source for commutation should come from its dedicated ECT.

Parameter	Technique 1		Technique 2/3
Commutation Source (Ixx83)	@ serial data register A	if ST+MT ≥ 24 bits	@ commutation ECT result
	@ ECT position result	if ST+MT < 24 bits	
Commutation Type (Ixx01)	= 3 (from Y register)	if ST+MT ≥ 24 bits	=1 (from X register)
	= 1 (from X register)	if ST+MT < 24 bits	

Note

Special considerations should be made if the Single-turn (ST) and Multi-turn (MT) data bits are NOT contiguous (in consecutive fields). Contact Delta Tau for assistance with these special cases.

Note

Multi-turn MT is equal to zero for encoders which do not possess Multi-turn data bits.

Resolution Scale Factor (SF)

Parameter	Encoder Type	Technique 1/3	Technique 2
Resolution Scale Factor SF	Rotary [counts/rev]	= 2^{ST}	= 2^{ST-5} = $2^{ST}/32$
	Linear [counts/user units]	= 1/RES	= 1/(32*RES)

Where ST: is the rotary encoder Single-turn resolution in bits
RES: is the linear scale resolution, in user units (e.g. mm)

Commutation Cycle Size

Parameter	Motor/Encoder	Technique 1	Technique 2/3
Ixx70	Rotary	= Number of pole pairs	
	Linear	= 1	
Ixx71	Rotary	= SF= 2^{ST} if Ixx01=3	= 2^{18} = 262144
		= 32 * SF= 32 * 2^{ST} if Ixx01=1	
	Linear	= ECL * SF= ECL/RES if Ixx01=3	= ECL * SF / 2^{Offset} = ECL/(RES*2^{Offset})
		= 32 * ECL * SF = 32 * (ECL/RES) if Ixx01=1	

Where ST: is the rotary encoder Single-turn resolution in bits
RES: is the linear scale resolution, in user units (e.g. mm)
ECL: is the electrical cycle length of the linear motor, same units as RES (e.g. mm)
Offset: is the ECT commutation Offset, it is (=ST-18 for rotary, or =RES-18 for linear)
SF: is the encoder resolution scale factor (calculated previously)

Position and Velocity Scale Factors, Position Error Limit

With technique 2, and technique 3 (with encoder resolutions greater than 20 bits), it is recommended to set the position and velocity scale factors to equal 1 and widen the position error limit. Otherwise, default values should be ok for all other cases. This alleviates register saturation(s), allows for higher commanded speed settings and easier PID (position-loop) tuning.

Parameter(s)	Technique 1	Technique 2	Technique 3
Ixx08, Ixx09	= 96	= 1	= 96 for ST < 20 = 1 for ST $\geq$ 20
Ixx67	Default	= 8388607	= Default for ST < 20 = 8388607 for ST $\geq$ 20

Absolute Power-On Position and Phasing

Process	Technique 1	Technique 2	Technique 3
Absolute Position Read	From serial register A, automatic settings	From serial register A, scaling required	From serial register A, automatic settings
Absolute Phasing	Automatic settings, depending on ST+MT	From ECT for Comm., automatic settings	From ECT for Comm., automatic settings

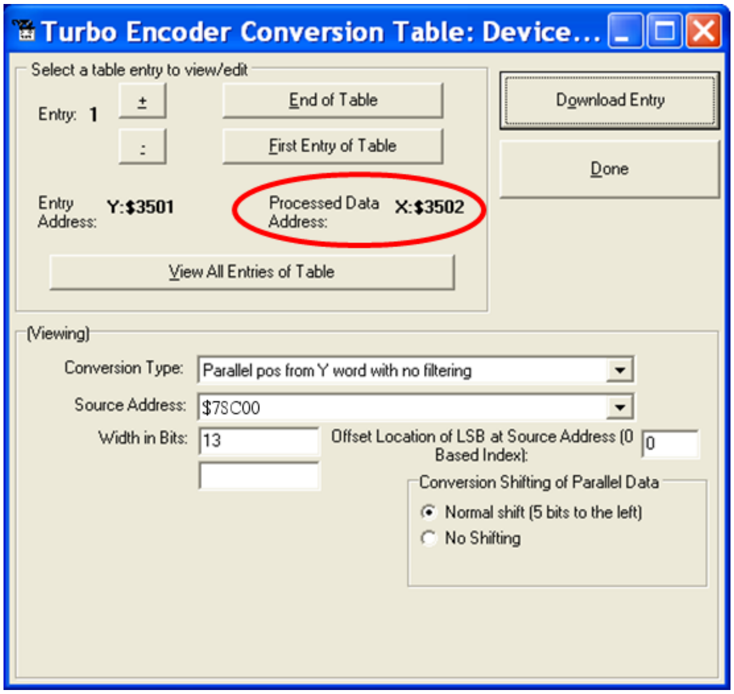
Technique 1 Example

Channel 1 is driving a 25-bit (13-bit Single-turn, 12-bit Multi-turn) rotary serial encoder, or a linear scale with similar protocol resolution (13 bits, 1 micron).

Encoder Conversion Table - for position (Technique 1)

- Conversion Type: Parallel position from Y word with no filtering
- Width in Bits: Single-turn/absolute resolution in bits (e.g. 13 bits)
- Offset Location of LSB: leave at zero
- Normal Shift (5 bits to the left)
- Source Address: serial data register A (see table below)
- Remember to click on Download Entry for the changes to take effect.

Turbo PAMC Base Address	Channel			
	1	2	3	4
$78C00	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
$79C00	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
$7AC00	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
$7BC00	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
$78D00	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
$79D00	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
$7AD00	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
$7BD00	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
$78E00	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
$79E00	Y:$79E00	Y:$79E04	Y:$79E08	Y:$79E0C
$7AE00	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
$7BE00	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C


Turbo Encoder Conversion Table: Device...
Select a table entry to view/edit
+
-
End of Table
First Entry of Table
Download Entry
Done
View All Entries of Table
(Viewing)
Conversion Shifting of Parallel Data

Entry: 1

Entry Address: Y:$3501

Processed Data Address: X:$3502

Conversion Type: Parallel pos from Y word with no filtering

Source Address: $78C00

Width in Bits: 13

Offset Location of LSB at Source Address (0 Based Index): 0

● Normal shift (5 bits to the left)

○ No Shifting

This is a 2-line ECT entry, its equivalent script code:

```
I8000=$278C00          ; Unfiltered parallel pos of location Y:$78C00
I8001=$00D000          ; Width and Offset. Processed result at $3502
```

Typically, the position and velocity pointers are set to the processed data address (e.g. $3502):

```
I100=1                 ; Mtr#1 Active. Remember to activate the channel to see feedback
I103=$3502             ; Mtr#1 position loop feedback address
I104=$3502             ; Mtr#1 velocity loop feedback address
```

Note

At this point, you should be able to move the motor/encoder shaft by hand and see ‘motor’ counts in the position window.

Counts per User Units (Technique 1)

With technique 1, the user should expect to see 2^{ST} counts per revolution for rotary encoders, and 1/Resolution counts per user unit for linear scales in the motor position window.

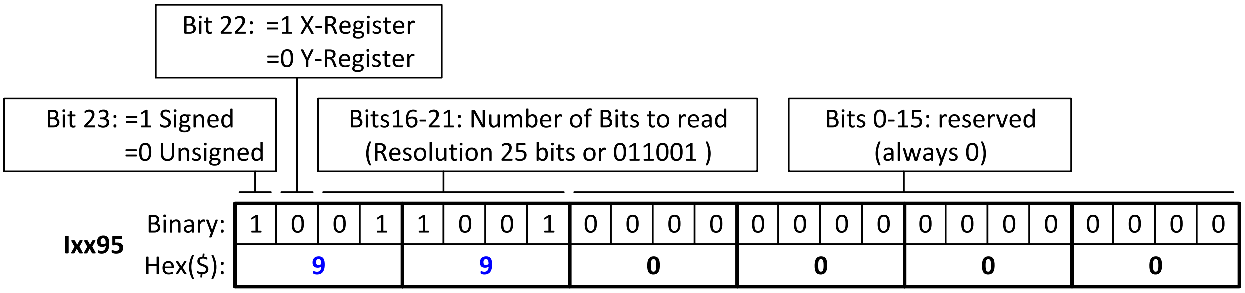
Examples: 25-bit rotary encoder (13-bit Single-turn): 2^{13}= 8,192 cts/rev
1-micron linear scale: 1/0.001= 1,000 cts/mm

Absolute Power-On Position Read (Technique 1)

With Technique 1, the absolute power-on read can be performed using PMAC's automatic settings (Ixx80, Ixx10 and Ixx95).

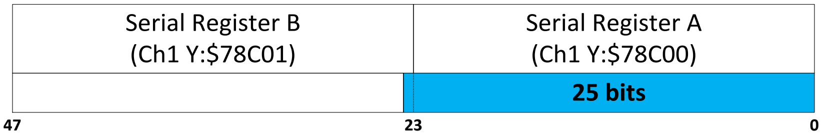
Example 1: Channel 1 driving a 25-bit (13-bit single turn, 12-bit multi-turn) rotary serial encoder:

```
I180=2                     ; Absolute power-on read enabled
I110=$78C00                ; Absolute power-on position address (ch1 serial data register A)
I195=$990000               ; Parallel Read, 25 bits, Signed, from Y-Register –User Input
```



Ixx95	Binary:	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Hex($):	9				9				0				0				0				0			

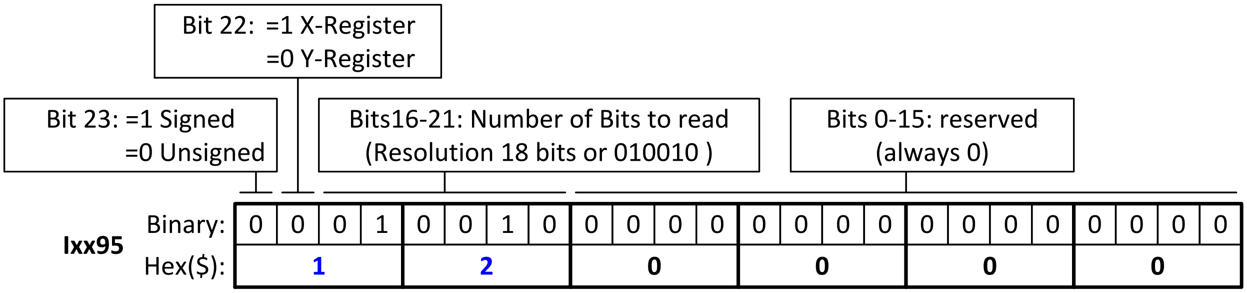
In this mode, PMAC reads and reports 25 bits from the consecutive serial data registers:



With the setting of Ixx80=2, the actual position is reported automatically on Power-up. Otherwise, a #1$* command is necessary to read and report the absolute position.

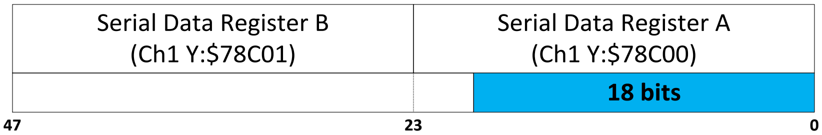
Example 2: Channel 1 driving an 18-bit (18-bit Single-turn, No Multi-turn) absolute rotary serial encoder, or a similar protocol resolution (18 bits) linear scale:

```
I180=2                     ; Absolute power-on read enabled
I110=$78C00                ; Absolute power-on position address (ch1 serial data register A)
I195=$120000               ; Parallel Read, 18 bits, Unsigned, from Y-Register –User Input
```



Ixx95	Binary:	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Hex($):	1				2				0				0				0				0			

In this mode, PMAC reads and reports 18 bits from the first serial data register:



With this setting of Ixx80=2, the actual position is reported automatically on Power-up. Otherwise, a #1$* command is necessary to read and report the absolute position.

Note

With absolute serial encoders (no multi-turn data), the power-on position format is set up for unsigned operation.

Note

The upper two fields in Ixx95 are the only relevant ones. Bits 0 through 15 are reserved and should always be set to 0.

Note

Some serial encoders use an external source for power. Make sure that this power is applied prior to performing an absolute read on power-up.

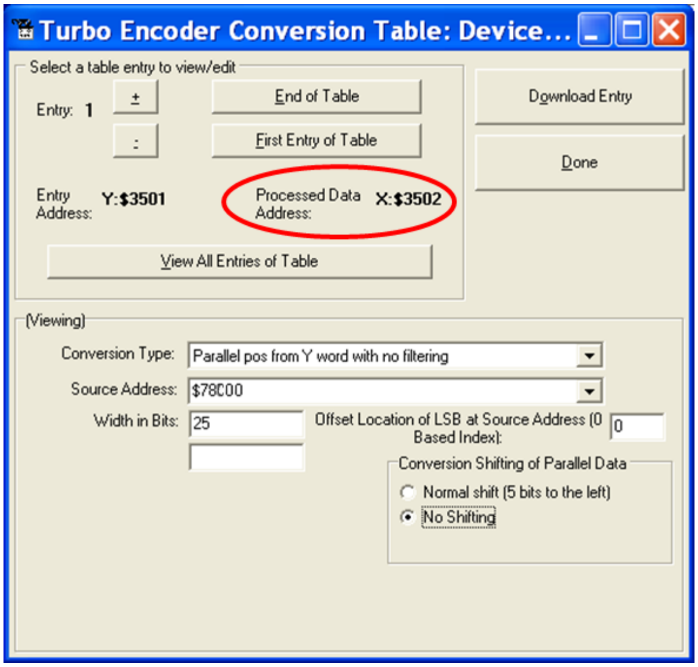
Technique 2 Example

Channel 1 is driving a 37-bit (25-bit Single-turn, 12-bit Multi-turn) rotary serial encoder, or a linear scale with similar protocol resolution (25 bits, 10 nanometer).

Encoder Conversion Table – for position (Technique 2)

- Conversion Type: Parallel position from Y word with no filtering
- Width in Bits: Single-turn/absolute resolution in bits (e.g. 25 bits)
- Offset Location of LSB: leave at zero
- No shifting
- Source Address: serial data register A (see table below)
- Remember to click on Download Entry for the changes to take effect.

Turbo PAMC Base Address	Channel			
	1	2	3	4
$78C00	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
$79C00	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
$7AC00	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
$7BC00	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
$78D00	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
$79D00	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
$7AD00	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
$7BD00	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
$78E00	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
$79E00	Y:$79E00	Y:$79E04	Y:$79E08	Y:$79E0C
$7AE00	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
$7BE00	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C


Turbo Encoder Conversion Table: Device...
Select a table entry to view/edit
+
-
End of Table
First Entry of Table
Download Entry
Done
View All Entries of Table
(Viewing)
Conversion Shifting of Parallel Data

Entry: 1

Entry Address: Y:$3501

Processed Data Address: X:$3502

Conversion Type: Parallel pos from Y word with no filtering

Source Address: $78C00

Width in Bits: 25

Offset Location of LSB at Source Address (0 Based Index): 0

○ Normal shift (5 bits to the left)

● No Shifting

This is a 2-line ECT entry, its equivalent script code:

```
I8000=$2F8C00          ; Unfiltered parallel pos of location Y:$78C00
I8001=$19000           ; Width and Offset. Processed result at $3502
```

Typically, the position and velocity pointers are set to the processed data address (e.g. $3502). Also, with technique 2, it is recommended to set the position and velocity scale factors to 1 and the position error limit to its maximum value:

```
I100=1                 ; Mtr#1 Active. Remember to activate the channel to see feedback
I103=$3502             ; Mtr#1 position loop feedback address
I104=$3502             ; Mtr#1 velocity loop feedback address
I108=1                 ; Mtr#1 position-loop scale factor
I109=1                 ; Mtr#1 velocity-loop scale factor
I167=8388607           ; Mtr#1 Position Error Limit
```

Note

At this point, you should be able to move the motor/encoder shaft by hand and see ‘motor’ counts in the position window

Counts Per User Units (Technique 2)

With technique 2, the user should expect to see $2^{ST-5}= 2^{ST}/32$ counts per revolution for rotary encoders, and 1/(32*Resolution) counts per user unit for linear scales in the motor position window.

Examples: 37-bit rotary encoder (25-bit Single-turn): $2^{25}/32$= 1,048,576 cts/rev
10-nanometer linear scale: 1/(32*0.000010)= 3,125 cts/mm

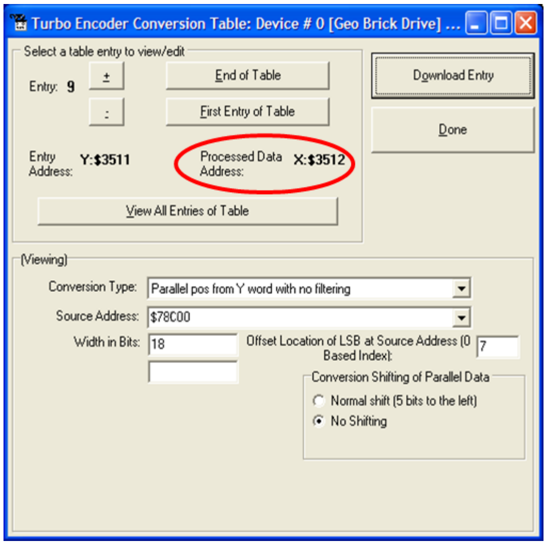
Encoder Conversion Table - for commutation (Technique 2)

Commutation with Turbo PMAC does not require high resolution data. With Technique 2, it is recommended to fix it at 18 bits. This will also eliminate quantization noise.

Note It is recommended to insert the commutation ECT entries after all of the position ECT entries have been configured.

Assuming that eight encoders have been configured for position, the first ECT for commutation for the first motor would be at entry number nine:

- Conversion Type: Parallel pos from Y word with no filtering
- Width in Bits: 18
- Offset Location of LSB: = Singleturn/protocol bits – 18 (e.g. 25-18=7)
- No shifting
- Source Address: serial data register A (same as position ECT for this motor)
- Remember to click on Download Entry for the changes to take effect.


Turbo Encoder Conversion Table: Device # 0 [Geo Brick Drive] ...
+
-
End of Table
First Entry of Table
Download Entry
Done
View All Entries of Table

Select a table entry to view/edit

Entry: 9

Entry Address: Y:$3511

Processed Data Address: X:$3512

(Viewing)

Conversion Type: Parallel pos from Y word with no filtering

Source Address: $78C00

Width in Bits: 18

Offset Location of LSB at Source Address (0 Based Index): 7

Conversion Shifting of Parallel Data

○ Normal shift (5 bits to the left)

● No Shifting

This is a 2-line ECT entry, its equivalent script code:

```
I8016=$2F8C00           ; Unfiltered parallel pos of location Y:$78C00 –User Input
I8017=$12007            ; Width and Offset. Processed result at X:$3512 –User Input
```

Note Record the processed data address (e.g. $3512). This is where the commutation position address Ixx83 will be pointing to. Also, this will be used in setting up the power-on phasing routine.

The commutation enable, and position address would then be:

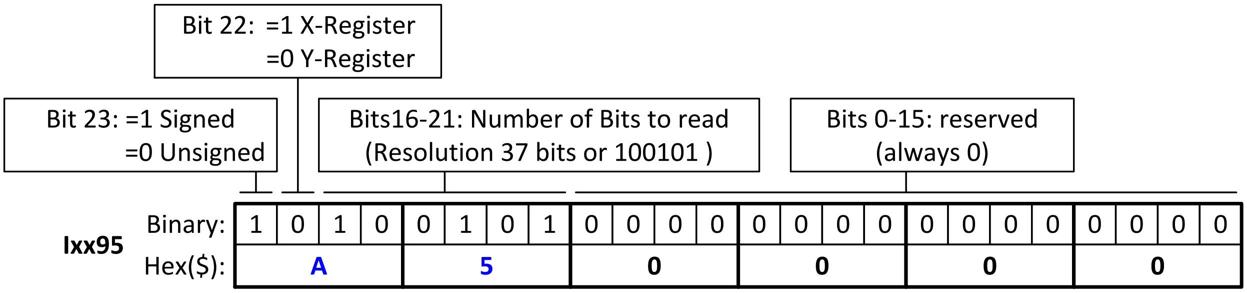
```
I101=1                  ; Mtr#1 Commutation enable, from X Register
I183=$3512              ; Mtr#1 Commutation Position Address –User Input
```

Absolute Power-On Position Read (Technique 2)

With technique 2, the absolute power-on position can be read directly from the serial data registers. But, proper scaling (5-bit right shift, in a PLC) is required to conform to the unshifted on-going position.

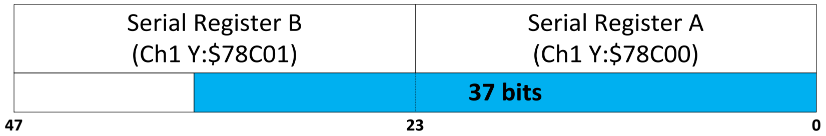
Example 1: Channel 1 driving a 37-bit (25-bit single turn, 12-bit multi-turn) rotary serial encoder:

```
I180=0                  ; Absolute power-on read disabled
I110=$78C00             ; Absolute power-on position address (ch1 serial data register A)
I195=$A50000            ; Parallel Read, 37 bits, Signed, from Y-Register -User Input
```



Bit 22: =1 X-Register
=0 Y-Register
Bit 23: =1 Signed
=0 Unsigned
Bits16-21: Number of Bits to read
(Resolution 37 bits or 100101)
Bits 0-15: reserved
(always 0)
Ixx95
Binary: 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Hex($): A 5 0 0 0 0

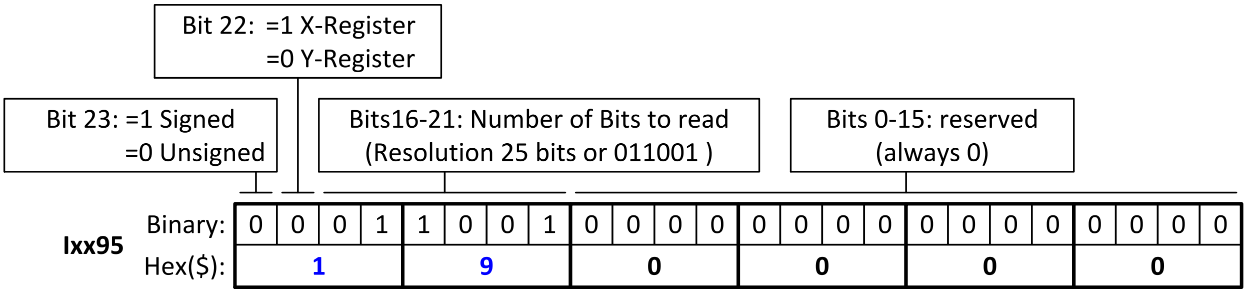
In this mode, PMAC reads 37 bits from the consecutive serial data registers:



With the setting of Ixx80=0, the actual position is not reported automatically on power-up. It will be reported after scaling (i.e. in PLC, below).

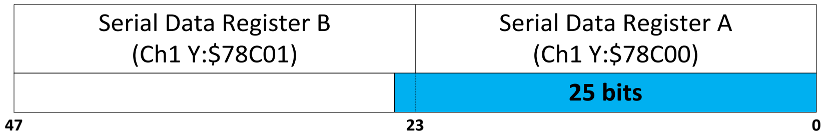
Example 2: Channel 1 driving a 25-bit (25-bit Single-turn, No Multi-turn) absolute rotary serial encoder, or a similar protocol resolution (25 bits) linear scale:

```
I180=0                  ; Absolute power-on read disabled
I110=$78B20             ; Absolute power-on position address (ch1 serial data register A)
I195=$190000            ; Parallel Read, 25 bits, Unsigned, from Y-Register -User Input
```



Bit 22: =1 X-Register
=0 Y-Register
Bit 23: =1 Signed
=0 Unsigned
Bits16-21: Number of Bits to read
(Resolution 25 bits or 011001)
Bits 0-15: reserved
(always 0)
Ixx95
Binary: 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Hex($): 1 9 0 0 0 0

In this mode, PMAC reads 25 bits from the first serial data register:



With the setting of Ixx80=0, the actual position is not reported automatically on power-up. It will be reported after scaling (i.e. in PLC, below).

Note

With absolute serial encoders (no multi-turn data), the power-on position format is set up for unsigned operation.

Note

The upper two fields in Ixx95 are the only relevant ones. Bits 0 through 15 are reserved and should always be set to 0.

Power-On Position scaling PLC example (for technique 2)

```
M162->D:$00008B                              ; #1 Actual position (Suggested M-Variable)

Open PLC 1 clear
I5111=100*8388608/I10 while(I5111>0) endw    ; 100 msec delay
CMD“#1K“                                     ; Make sure motor(s) killed
I5111=100*8388608/I10 while(I5111>0) endw    ; 100 msec delay
CMD“#1$*“                                    ; Read un-scaled absolute position
I5111=100*8388608/I10 while(I5111>0) endw    ; 100 msec delay
M162=M162/32                                 ; Scale absolute position (shift right 5 bits)
I5111=100*8388608/I10 while(I5111>0) endw    ; 100 msec delay
Dis PLC 1                                    ; Run once on power-up or reset
Close
```

Note

Some serial encoders use an external (not from the Brick) source for power. Make sure that this power is applied prior to performing an absolute read on power-up.

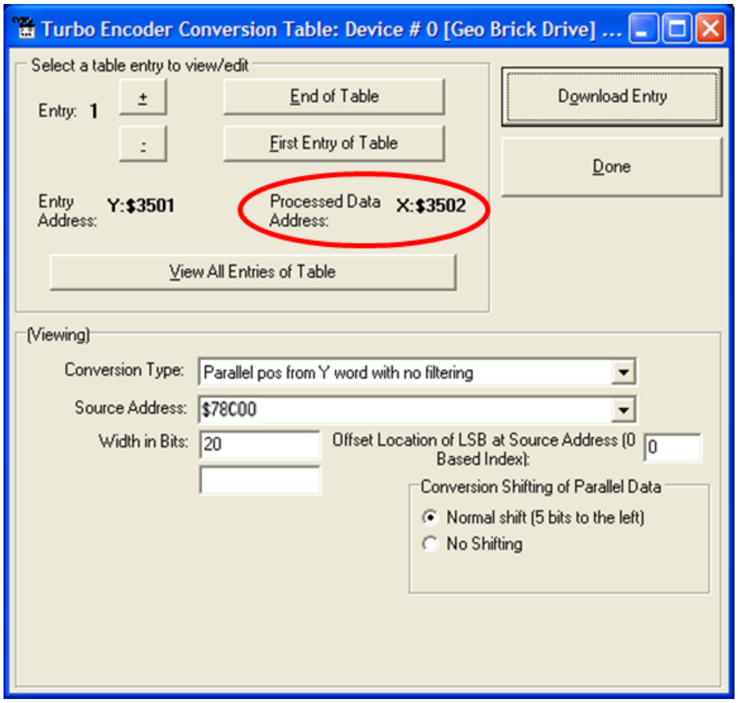
Technique 3 Example

Channel 1 is driving a 32-bit (20-bit Single-turn, 12-bit Multi-turn) rotary serial encoder, or a linear scale with similar protocol resolution (20 bits, 0.1 micron).

Encoder Conversion Table - for position (Technique 3)

- Conversion Type: Parallel position from Y word with no filtering
- Width in Bits: Single-turn/absolute resolution in bits (e.g. 20 bits)
- Offset Location of LSB: leave at zero
- Normal Shift (5 bits to the left)
- Source Address : serial data register A (see table below)
- Remember to click on Download Entry for the changes to take effect.

Turbo PAMC Base Address	Channel			
	1	2	3	4
$78C00	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
$79C00	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
$7AC00	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
$7BC00	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
$78D00	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
$79D00	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
$7AD00	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
$7BD00	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
$78E00	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
$79E00	Y:$79E00	Y:$79E04	Y:$79E08	Y:$79E0C
$7AE00	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
$7BE00	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C


Turbo Encoder Conversion Table: Device # 0 [Geo Brick Drive] ...
Select a table entry to view/edit
+
-
End of Table
First Entry of Table
Download Entry
Done
View All Entries of Table
(Viewing)
Conversion Shifting of Parallel Data

Entry: 1

Entry Address: Y:$3501

Processed Data Address: X:$3502

Conversion Type: Parallel pos from Y word with no filtering

Source Address: $78C00

Width in Bits: 20

Offset Location of LSB at Source Address (0 Based Index): 0

● Normal shift (5 bits to the left)

○ No Shifting

This is a 2-line ECT entry, its equivalent script code:

```
I8000=$278C00          ; Unfiltered parallel pos of location Y:$78C00
I8001=$014000          ; Width and Offset. Processed result at $3502
```

Typically, the position and velocity pointers are set to the processed data address (e.g. $3502). With Single-turn or linear resolutions less than 20 bits, the position/velocity scale factors, and position error limit can be left at default values. But with resolutions of 20 bits or greater, it is recommended to set the scale factors to 1 and the position error limit to its maximum value:

```
I100=1                 ; Mtr#1 Active. Remember to activate the channel to see feedback
I103=$3502             ; Mtr#1 position loop feedback address
I104=$3502             ; Mtr#1 velocity loop feedback address
I108=1                 ; Mtr#1 position-loop scale factor
I109=1                 ; Mtr#1 velocity-loop scale factor
I167=8388607           ; Mtr#1 Position Error Limit
```

Note

At this point, you should be able to move the motor/encoder shaft by hand and see ‘motor’ counts in the position window.

Counts Per User Units (Technique 3)

With technique 3, the user should expect to see 2^{ST} counts per revolution for rotary encoders, and 1/Resolution counts per user unit for linear scales in the motor position window.

Examples: 32-bit rotary encoder (20-bit Singleturn): 2^{20}= 1,048,576 cts/rev
0.1-micron linear scale: 1/0.0001= 10,000 cts/mm

Encoder Conversion Table - for commutation (Technique 3)

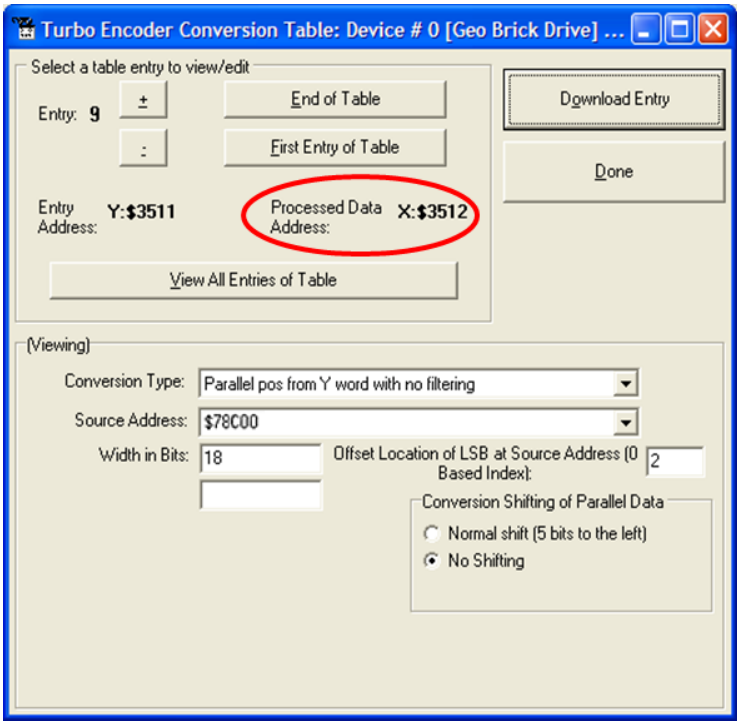
Commutation with Turbo PMAC does not require high resolution data. With Technique 3, it is recommended to fix it at 18 bits. This will also eliminate quantization noise.

Note

It is recommended to insert the commutation ECT entries after all of the position ECT entries have been configured.

Assuming that eight encoders have been configured for position, the first ECT for commutation for the first motor would be at entry number nine:

- Conversion Type: Parallel pos from Y word with no filtering
- Width in Bits: 18
- Offset Location of LSB = Singleturn/protocol bits – 18 (e.g. 20-18=2)
- No shifting
- Source Address: Serial data register A (same as position ECT for this motor)
- Remember to click on Download Entry for the changes to take effect.


Turbo Encoder Conversion Table: Device # 0 [Geo Brick Drive] ...
Select a table entry to view/edit
+
-
End of Table
First Entry of Table
Download Entry
Done
View All Entries of Table
(Viewing)
Conversion Shifting of Parallel Data

Entry: 9

Entry Address: Y:$3511

Processed Data Address: X:$3512

Conversion Type: Parallel pos from Y word with no filtering

Source Address: $78C00

Width in Bits: 18

Offset Location of LSB at Source Address (0 Based Index): 2

○ Normal shift (5 bits to the left)
● No Shifting

This is a 2-line ECT entry, its equivalent script code:

```
I8016=$2F8C00          ; Unfiltered parallel pos of location Y:$78C00 –User Input
I8017=$12002           ; Width and Offset. Processed result at X:$3512 –User Input
```

Note

Record the processed data address (e.g. $3512). This is where the commutation position address Ixx83 will be pointing to. Also, this will be used in setting up the power-on phasing routine.

The commutation enable, and position address would then be:

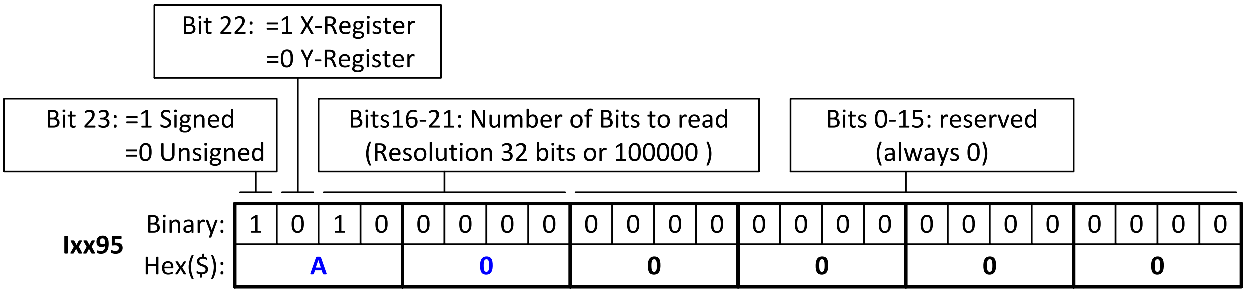
```
I101=1                 ; Mtr#1 Commutation enable, from X Register
I183=$3512             ; Mtr#1 Commutation Position Address –User Input
```

Absolute Power-On Position Read (Technique 3)

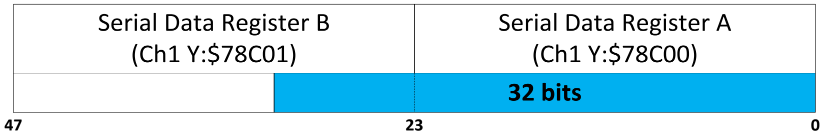
With Technique 3, the absolute power-on read can be performed using PMAC's automatic settings (Ixx80, Ixx10 and Ixx95).

Example 1: Channel 1 driving a 32-bit (20-bit single turn, 12-bit multi-turn) rotary serial encoder:

```
I180=2                 ; Absolute power-on read enabled
I110=$78C00            ; Absolute power-on position address (ch1 serial data register A)
I195=$A00000           ; Parallel Read, 32 bits, Signed, from Y-Register –User Input
```


Bit 22: =1 X-Register
=0 Y-Register
Bit 23: =1 Signed
=0 Unsigned
Bits16-21: Number of Bits to read
(Resolution 32 bits or 100000)
Bits 0-15: reserved
(always 0)
Ixx95
Binary: 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Hex($): A 0 0 0 0 0

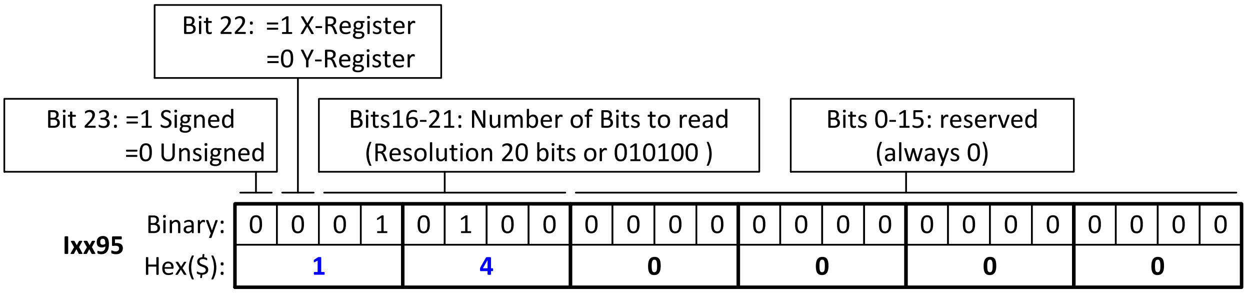
In this mode, PMAC reads and reports 32 bits from the consecutive serial data registers:



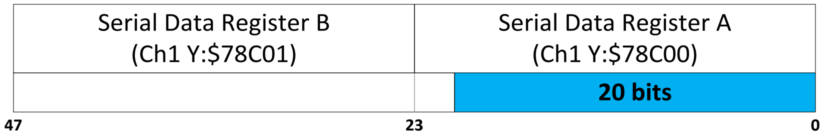
With the setting of Ixx80=2, the actual position is reported automatically on Power-up. Otherwise, a #1$* command is necessary to read and report the absolute position.

Example 2: Channel 1 driving a 20-bit (20-bit Single-turn, No Multi-turn) absolute rotary serial encoder, or a similar protocol resolution (20 bits) linear scale:

```
I180=2                 ; Absolute power-on read enabled
I110=$78C00            ; Absolute power-on position address (ch1 serial data register A)
I195=$140000           ; Parallel Read, 20 bits, Unsigned, from Y-Register –User Input
```


Bit 22: =1 X-Register
=0 Y-Register
Bit 23: =1 Signed
=0 Unsigned
Bits16-21: Number of Bits to read
(Resolution 20 bits or 010100)
Bits 0-15: reserved
(always 0)
Ixx95
Binary: 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Hex($): 1 4 0 0 0 0

In this mode, PMAC reads and reports 20 bits from the first serial data register:



With the setting of Ixx80=2, the actual position is reported automatically on Power-up. Otherwise, a #1$* command is necessary to read and report the absolute position.

Note

With absolute serial encoders (no multi-turn data), the power-on position format is set up for unsigned operation.

Note

The upper two fields in Ixx95 are the only relevant ones. Bits 0 through 15 are reserved and should always be set to 0.

Note

Some serial encoders use an external (not from the Brick) source for power. Make sure that this power is applied prior to performing an absolute read on power-up.

Ongoing Commutation Phase Position

For the commutation algorithm's ongoing phase position, Turbo PMAC reads the entire 24-bit register specified by Ixx83 every phase cycle. In order to be able to handle rollover of this data properly, the most significant bit (MSB) of this data must end up in bit 31 of the 32-bit result, shifted, in ECT, if necessary. With most protocols, no shifting is necessary, but some will require a net "left shift" to achieve this result.

No Data Shifting

To use serial encoder position from an ACC-84E FPGA-based interface for ongoing phase position where the data is properly left shifted or it is contiguous over 24-bit register.

Note

Position data from serial encoder protocols SSI, EnDat2.1/2.2, Sigma II/III/V, BiSS-B/C and Mitsubishi protocols usually fit into this method, unless the resolution per electrical cycle is greater than 16777215 $(2^{24} - 1)$ as discussed below, in which case data shifting method should be used.

- Ixx83 is set based upon the following table:

Turbo PAMC Base	Channel			
Address	1	2	3	4
$78C00	$78C00	$78C04	$78C08	$78C0C
$79C00	$79C00	$79C04	$79C08	$79C0C
$7AC00	$7AC00	$7AC04	$7AC08	$7AC0C
$7BC00	$7BC00	$7BC04	$7BC08	$7BC0C
$78D00	$78D00	$78D04	$78D08	$78D0C
$79D00	$79D00	$79D04	$79D08	$79D0C
$7AD00	$7AD00	$7AD04	$7AD08	$7AD0C
$7BD00	$7BD00	$7BD04	$7BD08	$7BD0C
$78E00	$78E00	$78E04	$78E08	$78E0C
$79E00	$79E00	$79E04	$79E08	$79E0C
$7AE00	$7AE00	$7AE04	$7AE08	$7AE0C
$7BE00	$7BE00	$7BE04	$7BE08	$7BE0C

The Ixx70 and Ixx71 determine the commutation cycle length for Turbo PMAC.

Note

It is important to notice that the Ixx71 has a range of 0 – 16777215 $(2^{24} - 1)$. In some cases where the encoder resolution is more than 24-bits per electrical cycle, user should use the shifted data method discussed in the next section rather than using the data directly as discussed here.

If number of counts (register LSB) per motor revolution is less than $(2^{24} - 1)$, Ixx70 and Ixx71 can be set as shown below:

- **Ixx01 = 3** *// Turbo PMAC commutation, commutation feedback from Y-register*
- **Ixx70 =** *(Number of pole pairs for motor revolution)*
- **Ixx71** = *(Register LSB per motor revolution)*

If number of counts (register LSB) per motor revolution is greater than $(2^{24} - 1)$ but the number of counts (register LSB) per electrical cycle is:

a. An Integer
b. Less than $(2^{24} - 1)$

Ixx70 and Ixx71 can be set as shown below:

- **Ixx01 = 3** *// Turbo PMAC commutation, commutation feedback from Y-register*
- **Ixx70 =** *1*
- **Ixx71** = *(Register LSBs per commutation cycle*)

Data Shifting

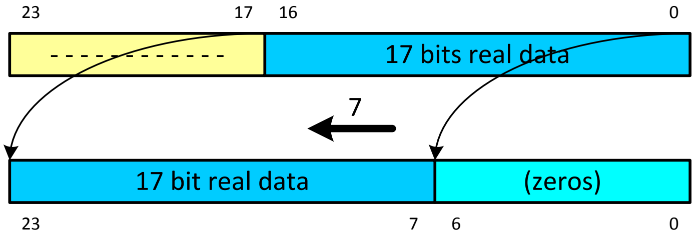
In order for Turbo PMAC CPU to be able to handle rollover of on-going phase position data properly, the most significant bit (MSB) of the position data must end up in bit 23 of the 24-bit result. In some cases the data needs to be shifted in order to move the MSB of position data (single turn position data in most cases) to bit 23 of the 24-resulting register.

Note

Position data from serial encoder protocols Tamagawa, Panasonic, Mitutoyo, Matsushita usually fit into this method, in addition to cases where the resolution per electrical cycle is greater than 16777215 $(2^{24} - 1)$ and data shifting method should be used.

Shifting-Up MSB of Real Data

In most cases (with Tamagawa, Panasonic, Mitutoyo, Matsushita protocols), the MSB of the real position data is not at bit 23 of the 24-bit register, which is required for Turbo PMAC, in order for it to properly handle the roll-over. In these cases, the data should be shifted in the Encoder Conversion Table (ECT) and the result of ECT can be used at commutation feedback:



ECT Data Shifting Example

In these cases a Y-memory Parallel Read method is used in Encoder Conversion Table. This entry is a two line entry where the first entry defines the source of the data and shift in resulting data and the second line determines the location of the real position data in the source register. The following table shows the typical values for this ECT entry:

1st line: Source register and result shift (no-shift):

ACC-84E Base Address	Channel			
	1	2	3	4
$78C00	$2F8C00	$2F8C04	$2F8C08	$2F8C0C
$79C00	$2F9C00	$2F9C04	$2F9C08	$2F9C0C
$7AC00	$2FAC00	$2FAC04	$2FAC08	$2FAC0C
$7BC00	$2FBC00	$2FBC04	$2FBC08	$2FBC0C
$78D00	$2F8D00	$2F8D04	$2F8D08	$2F8D0C
$79D00	$2F9D00	$2F9D04	$2F9D08	$2F9D0C
$7AD00	$2FAD00	$2FAD04	$2FAD08	$2FAD0C
$7BD00	$2FBD00	$2FBD04	$2FBD08	$2FBD0C
$78E00	$2F8E00	$2F8E04	$2F8E08	$2F8E0C
$79E00	$2F9E00	$2F9E04	$2F9E08	$2F9E0C
$7AE00	$2FAE00	$2FAE04	$2FAE08	$2FAE0C
$7BE00	$2FBE00	$2FBE04	$2FBE08	$2FBE0C

2nd line: Real position data location:
The following values are encoder/protocol dependent and should be selected accordingly. The following values are shown for most common encoder resolutions:

Feedback Resolution	2nd Line Setting
16-bit Single Rev, Starting at bit 0	$010000
17-bit Single Rev, Starting at bit 0	$011000
20-bit Single Rev, Starting at bit 0	$014000
17-bit Single Rev, Starting at bit 4	$011004
26-bit Single Rev, Starting at bit 0	$018002
32-bit Single Rev, Starting at bit 0	$018008
$7AD00	$2FAD00
$7BD00	$2FBD00
$78E00	$2F8E00
$79E00	$2F9E00
$7AE00	$2FAE00
$7BE00	$2FBE00

Power-On Commutation Phase Position

Because most serial encoders provide absolute position information, especially over one motor revolution, they are commonly used to provide the absolute rotor-angle position at power-up for the commutation algorithms. Doing this requires assigning proper values to several saved setup elements.

This section gives an overview of those settings; details can be found in the element descriptions in the Software Reference Manual, the *Setting Up Commutation* chapter of the User's Manual, and the Hardware Reference Manual for the interface. In addition the motor setup routines in the IDE software will walk you through this setup.

To use serial encoder position from an ACC-84E FPGA-based interface for absolute power-on phase position, the following saved setup elements must be specified:

- **Motor[*x*].pAbsPhasePos** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **Motor[*x*].AbsPhasePosFormat** = $*aabbccdd*   // Protocol-specific settings
- **Motor[*x*].AbsPhasePosSf** = 2048 / (*LSBs per commutation cycle*)
- **Motor[*x*].AbsPhasePosOffset** = (*Difference between sensor zero and commutation zero*)

For the format variable, the LSB of the encoder data is typically found in bit 8 of the 32-bit register, and only enough bits to cover a single commutation cycle need to be used. (However, it does not hurt to specify more bits than are required.) It is seldom required to use data from the next register.

Ongoing Servo Position

To use the serial encoder position for ongoing servo position, the data must first be processed in the encoder conversion table. This is done with a “Type 1” single-register-read conversion from **SerialEncDataA**. In order to be able to handle rollover of this data properly, the most significant bit (MSB) of this data must end up in bit 31 of the 32-bit result, shifted if necessary. With most protocols, no shifting is necessary, but some will require a net “left shift” to achieve this result.

To use serial encoder position from an ACC-84E FPGA-based interface for ongoing servo position, the following saved setup elements must be specified:

- **EncTable[*n*].Type** = 1
- **EncTable[*n*].pEnc** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **EncTable[*n*].index1** = (*32* **-** *# of bits*) // Shift left # of bits
- **EncTable[*n*].index2** = 8 // Shift right # of bits
- **EncTable[*n*].ScaleFactor** = 1/ ($2^{32 - \# \text{ of bits}}$) // For result in encoder LSBs
- **Motor[*x*].pEnc** = **EncTable[*n*].a** // Use table result for position-loop feedback
- **Motor[*x*].pEnc2** = **EncTable[*n*].a** // Use table result for velocity-loop feedback

Power-On Servo Position

Many serial encoders can provide absolute position over the entire range of travel of the motor. If so, Power PMAC can execute an absolute power-on read of the encoder to establish the reference position, eliminating the need for a homing search move.

This section gives an overview of those settings; details can be found in the element descriptions in the Software Reference Manual, the *Basic Motor Setup* chapter of the User's Manual, and the Hardware Reference Manual for the interface. In addition the motor setup routines in the IDE software will walk you through this setup.

To use serial encoder position from an ACC-84E FPGA-based interface for absolute power-on phase position, the following saved setup elements must be specified:

- **Motor[*x*].pAbsPos** = **Acc84E[*i*].Chan[*j*].SerialEncDataA.a**
- **Motor[*x*].AbsPosFormat** = $*aabbccdd*   // Protocol-specific settings
- **Motor[*x*].AbsPosSf** = (*Motor units per sensor LSB*)
- **Motor[*x*].AbsPosOffset** = (*Difference between sensor zero and motor zero*)

For the format variable, the LSB of the encoder data is typically found in bit 8 of the 32-bit **SerialEncDataA** register. If the encoder provides more than 24 bits of absolute position data, the format element permits data from **SerialEncDataB** to be used as well. Note, however, that the data in **SerialEncDataA** must go all the way to bit 31 for this to work. In protocols such as Tamagawa and Panasonic, which provide only 17 bits of data in **SerialEncDataA** and more in **SerialEncDataB**, the full absolute position must be assembled in a user algorithm.

Using the ACC-84E with MACRO

The ACC-84E can be used with MACRO8 and MACRO16 CPU in a MACRO UMAC rack to take advantage of serial encoders over a MACRO ring. The principal behind using the ACC-84E will be similar to what is available on Turbo PMAC2 UMAC CPU. However, there are some differences which will be discussed in this section.

Addressing and Register Addresses

ACC-84E will be address as an IO card on MACRO8 and MACRO16 CPUs. So the following addresses will be used:

Chip Select	3U MACRO CPU Adderss	Dip Switch SW1 Position			
		1	2	3	4
CS10	Y:$8800	Close	Close	Close	Close
	Y:$9800	Close	Close	Open	Close
	Y:$A800	Close	Close	Close	Open
	Y:$B800 ($FFE0*)	Close	Close	Open	Open
CS12	Y:$8840	Open	Close	Close	Close
	Y:$9840	Open	Close	Open	Close
	Y:$A840	Open	Close	Close	Open
	Y:$B840 ($FFE8*)	Open	Close	Open	Open
CS14	Y:$88C0	Close	Open	Close	Close
	Y:$98C0	Close	Open	Open	Close
	Y:$A8C0	Close	Open	Close	Open
	Y:$B8C0 ($FFF0*)	Close	Open	Open	Open

Based upon the base address selection, the address for each of the global registers and channel specific registers will change:

MACRO UMAC Base address	Global Setup Register Address	1st Channel Setup Register Address	2nd Channel Setup Register Address	3rd Channel Setup Register Address	4th Channel Setup Register Address
$8800	X:$880F	X:$8800	X:$8804	X:$8808	X:$880C
$9800	X:$980F	X:$9800	X:$9804	X:$9808	X:$980C
$A800	X:$A80F	X:$A800	X:$A804	X:$A808	X:$A80C
$B800	X:$B80F	X:$B800	X:$B804	X:$B808	X:$B80C
$8840	X:$884F	X:$8840	X:$8844	X:$8848	X:$884C
$9840	X:$894F	X:$8940	X:$8944	X:$8948	X:$894C
$A840	X:$A84F	X:$A840	X:$A844	X:$A848	X:$A84C
$B840	X:$B84F	X:$B840	X:$B844	X:$B848	X:$B84C
$88C0	X:$88CF	X:$88C0	X:$88C4	X:$88C8	X:$88CC
$98C0	X:$98CF	X:$98C0	X:$98C4	X:$98C8	X:$98CC
$A8C0	X:$A8CF	X:$A8C0	X:$A8C4	X:$A8C8	X:$A8CC
$B8C0	X:$B8CF	X:$B8C0	X:$B8C4	X:$B8C8	X:$B8CC

Each of the global and channel specific setup elements are set up exactly based upon the explanation in Turbo UMAC CPU section depending on the serial protocol in question.

Setting up the Global and Channel Registers on Power Up

As mentioned in the Turbo UMAC CPU section, for each of the protocol, certain setup words needs to be written to global and channel specific registers of ACC-84E. Since the ACC-84E is in the MACRO Rack and is not directly accessible to Turbo PMAC or Ultralite, the settings need to be sent through the MACRO node and MACRO communication. For this matter, use of MI198 and MI199 are recommended since this eliminates a need for defining MM variables on MACRO station. Here is an example of setting up an ACC-84E for BiSS-C protocol on a MACRO16 CPU. This PLC resides on Ultralite and, on power-up or execution of the PLC, sets up the global and channel specific registers for ACC-84E.

```
#define timer i5111
#define msec *8388607/i10while(i5111>0)endwhile

close
del gat
open plc 1 clear
cmd"clrf"
timer = 100 msec
cmd"msclrf15"
timer = 100 msec
cmd"ms0,mi198=$E8880F"        // pointing MI199 to Global Register at X:$880F
timer = 100 msec              // $E8 X Unsigned 24 bit, $78 Y Unsiged 24 bit
cmd"ms0,mi199=$63000B"        // Writing the value to Global Register
timer = 100 msec
cmd"ms0,mi198=$E88800"        // Pointing MI199 to Channel Specific register X:$8800
timer = 100 msec
cmd"ms0,mi199=$2114A0"        // Writing the value to Channel Specific register
timer = 100 msec
disable plc 1
close
```

Encoder Conversion Table Setup

In order to get the serial data passed through the MACRO interface, two different encoder conversion tables need to be set: MACRO ECT and Ultralite ECT.

The Encoder Conversion Table entry for MACRO CPU uses type $2 to read parallel data from Y-word where in most protocols position data, or at least lower bits of the data, is available. The second setting is to use the result of the ECT and send it over the MACRO to Ultralite.

```
ms0,mi120=$288800      // Method $2 parallel y-word read, Result at $10
                       // bit 19 set to 1 for no shifting of data
                       // $8800 channel base address
ms0,mi121=$FFFFFF      // Bits-Used Mask, All bits are used, Result at $11
ms0,mi101=$11          // node 0, sends result of 1st ECT entry as position
```

Once this data is available on the MACRO node, normal 24-bit Y-word parallel read will be used for reading of corresponding MACRO node, register 0 for on-going position of the axis. It is the user's choice whether or not to shift the data.

```
I8000=$2F8420          // Y-word parallel read from $78420, Node 0, Register 0, No-shift
I8001=$018000          // 24-bit wide read, starting at bit 0
```

Once the position data is available on the Ultralite side, position and velocity feedback pointers can be defined for the motors in question.

```
I103=$3502             // position feedback
I104=$3502             // velocity feedback
```

Absolute Power-On Phasing and Servo Power on Position

In order to read the absolute position over MACRO, the Power-Up Position Source Address (MS{anynode},MI111-MI118) variable needs to be set. This will allow the MACRO16 CPU to transfer the data whenever the MS{node},MI920 is queried. As for most of the protocols supported by ACC-84E, the position and status bits are read through the Channel Registers A and B, a Double-Y-Word parallel read should support most of the cases. The only question will be number of bits which need to be read, and that is dependent on serial encoder specifications.

```
ms0,mi111=$208800       //setup for power on position on node 0
                        //$1A is position data length $1A:26-bits $20:32-bits $12:18-bits
                        //8800 channel base address
```

Once MI111-MI118 are setup, reading MS{node},MI920 will return the position as a 48-bit value which can be assigned properly into motor actual position or can be used for establishing the phase reference of the motor. For examples, please refer to examples discussed in the Turbo UMAC setup section of this manual.

Commutating Over MACRO with High Resolution Encoders

It is possible to commutate motors with high resolution encoders over the MACRO ring. However, if the position resolution of encoder is more than 24-bits per revolution, certain measures should be taken since the automatic transfer of lower 24 bits of position data might not provide sufficient position information required for commutation and introduce considerable velocity limitations on motor control.

If the encoder on commutated motor has less than 24-bits of resolution per single revolution (at most 16,777,215 counts per electrical cycle), then the data available in register 0 of the node can be used for commutation purpose and the following setting are sufficient. Here is an example of commutation if the motor was on node 0.

```
I101=3          // motor is commutated, commutation data is on Y-memory location
I183=$78420     // Node 0, Register 0
I170=2          // 2-pole pair motor
I171=262144     // 18-bits of resolution per revolution of the motor
```

If the encoder has 24-bits or more resolution per electrical cycle, then the upper portion of the data becomes important for commutation purpose. Since the commutation algorithm in PMAC uses a 2048 state Sine table, only the upper 12-bits of data per electrical cycle will be important for PMAC. Since the ECT entry only reads the lower 24-bits of position data, in order to transfer the upper 24-bits of data (most significant bits), a separate node should be used. For this purpose, it is suggested that a MACRO PLC on the MACRO16 CPU be implemented. This PLC will copy the upper 24-bits of data from each of the channels and copies them to register 0 of corresponding IO nodes of SERVO nodes. This MACRO PLC should be downloaded to MACRO16 CPU through PERIN32PRO2 software and while in MACROASCII communication mode.

```
// Generic Definition

CLOSE

#define Chan1RegA      MM100
#define Chan1RegB      MM101
#define Chan2RegA      MM102
#define Chan2RegB      MM103
#define Chan3RegA      MM104
#define Chan3RegB      MM105
#define Chan4RegA      MM106
#define Chan4RegB      MM107

#define Node2Reg0      MM108
#define Node3Reg0      MM109
#define Node6Reg0      MM110
#define Node7Reg0      MM111

Chan1RegA->Y:$8800,0,24
```

```
Chan1RegB->Y:$8801,0,24
Chan2RegA->Y:$8804,0,24
Chan2RegB->Y:$8805,0,24
Chan3RegA->Y:$8808,0,24
Chan3RegB->Y:$8809,0,24
Chan4RegA->Y:$880C,0,24
Chan4RegB->Y:$880D,0,24

Node2Reg0->X:$C0A0,0,24
Node3Reg0->X:$C0A4,0,24
Node6Reg0->X:$C0A8,0,24
Node7Reg0->X:$C0AC,0,24
```

```
// 26-bit Encoder
OPEN MACPLCC
Node2Reg0=(Chan1RegB & $000003)*$100000 + Chan1RegA/$8
Node3Reg0=(Chan2RegB & $000003)*$100000 + Chan2RegA/$8
Node6Reg0=(Chan3RegB & $000003)*$100000 + Chan3RegA/$8
Node7Reg0=(Chan4RegB & $000003)*$100000 + Chan4RegA/$8
CLOSE
```

```
// 32-bit Encoder
OPEN MACPLCC
Node2Reg0=(Chan1RegB & $0000FF)*$8000 + Chan1RegA/$200
Node3Reg0=(Chan2RegB & $0000FF)*$8000 + Chan2RegA/$200
Node6Reg0=(Chan3RegB & $0000FF)*$8000 + Chan3RegA/$200
Node7Reg0=(Chan4RegB & $0000FF)*$8000 + Chan4RegA/$200
CLOSE
```

These MACRO PLCs will copy the upper 23-bits of position data from ACC-84E position registers into Register 0 of MACRO nodes 2, 3, 6 and 7. On the Ultralite side, the commutation parameters need to be set up accordingly:

```
I101=1          // motor is commutated, commutation data is on X-memory location
I183=$78420     // Node 2, Register 0, X-memory location
I170=2          // 2-pole pair motor, motor dependent
I171=8388608    // 23-bits of resolution per revolution of the motor
```

APPENDIX A: SETUP EXAMPLES

This section is divided into several sections explaining different serial protocols and their settings. All the examples given is based upon the first addressed channel on ACC-84E which is the factory default address. User should change these addresses in each example to match their address settings.

SSI Feedback Setup Example

The following example demonstrates how to setup a 32-bit binary SSI encoder for position control of a brushless motor on the first channel of a ACC-84E. Assume that the documentation for the encoder suggests 1MHz clock for the length of the cable that we have in the system:

Multi-Channel Setup Element

X:$78C0F																							
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M Divisor								N Divisor				Reserved		Trigger Clock	Trigger Edge	Trigger Delay				Protocol Code			
0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
6				3				0				1				0				2			

```
WX:$78B2F,$630102        ; 1MHz, Phase Clock, Falling Edge, no Delay
```

Single-Channel Setup Element

Channel #1 setup example, 32-bit SSI Binary Encoder

Channel 1:X:$78B20																							
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Parity Type		Trigger Mode	Trigger Enable	Gray Code to Binary	RxData Ready/ SENC	Reserved				Position Bits					
0	1	1	0	0	0	1	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
0				0				1				4				2				0			

```
WX:$78C00,$001420        ; 32-bit SSI encoder, Binary
I8000=$278B20            ; Unfiltered parallel position of location
                         ; Y:$78C00, normal 5-bit shifting
I8001=$18000             ; 24-bit processed result at $3502
I103=$3502               ; position loop feedback address
I104=$3502               ; velocity loop feedback address
```

As you may have noticed, the encoder conversion table only reads the lower 24 bits of data. This is acceptable since the data is incremental. Please note that since the data is being read by the encoder conversion table, as long as ECT has 2 reads in each 24-bit transition, it can handle the roll-over gracefully and motor position will be updated correctly.

Brushless Motor with SSI Feedback - Setup Notes

The following settings are only general guidelines for parameters which user needs to set for a generic brushless motor. These setting very well may be deferent on each system depending on the amplifier and motor selection.

```
I100= 1                 ; Motor activated
I101= 1                 ; Commutation Enable
I103= $3502             ; position feedback address
I104= $3502             ; velocity feedback address
I172= 1365              ; Commutation Phase Angle, amplifier dependent
I184= $FFF000           ; Current-Loop Feedback Mask Word, amplifier dependent
I166= 7636              ; PWM Scale Factor, normally 15% above PWM clock
I102= $78202            ; Command Output Register. First channel of first ACC-24E2
I182= $78206            ; Current Loop Feedback Address. First channel of first ACC-24E2
I183= $3502             ; Commutation Position Address, from resolver ECT result.
                ; Same as position address of motor if the same encode is used
I124= $20001            ; Flag control. Over-travel-limits are disabled
```

Check motor manufacturer specifications and refer to the Turbo SRM.

```
I170= 1                 ; Commutation Cycles per revolution (Number of pole pairs)
I171= 8192*32           ; Counts per revolution. Measured or provided by manufacturer
                        ; shifted 5 bits (*32) because commutation address from ECT
```

These are Safety parameters, I2T protection. Check motor manufacturer specifications and refer to the Turbo SRM.

```
I157= 8025              ; Motor#1 Continuous Current Limit
I158= 2167              ; Motor#1 Integrated Current Limit
I169= 24077             ; Motor#1 Output Command Limit
```

Please note that since the ECT table data is being used for commutation, it is better to have the Servo clock set to the same frequency as the Phase clock so the data is available for commutation routines.
If you have your Ixx03 and Ixx04 set up properly to point to the correct ECT entry, you should be able to observe position feedback in the position window when moving the motor by hand.
Using the PMACTuningPro2, you should be able to tune for the Current-Loop gains:

```
I161= 0.05              ; Motor#1 Current-Loop Integral gain
I162= 0.01              ; Motor#1 Current-Loop Forward-path proportional gain
I176= 0.5               ; Motor#1 Current-Loop Back-path proportional gain
```

Motor Phasing. We suggest using the stepper method for rough phasing:

```
I180= 6                 ; Motor#1 Power-up mode
I173= 1200              ; Motor#1 Phase finding output Value
I174= 60                ; Motor#1 Phase finding time
```

Issue a #1$ from the online command window to phase motor. Completion of phasing routine can be confirmed by checking the motor status window accessible through View menu in PEWIN32PRO2 software.

Open Loop Test:

1. Issue a `#1hmz` to zero the position counter in the position window.
2. Issue a `#1o1` from the online command window. This will send a 1% command output and should move the motor slightly.
3. Issue a `K` to kill motor. If motor has not moved increase the open loop command output by increments of one until you see counts change in the position window.
4. Repeat steps 1 thru 3 now issuing a negative open loop command `#1o-1`
5. Positive counts/movement should correspond to a positive open loop command, and negative movement should correspond to negative commands.
6. If step 5 is a true statement, then skip to PID tuning. Otherwise, the encoder counting direction doesn't match the commutation direction. In this case, we can either setup the SSI encoder to send the position in the opposite direction, or we can set the Ixx70 to the negative value of what we have setup at the moment.
7. Repeat Open Loop test (steps 1 thru 4) to make sure the commutation is correct.

8. PID tuning: Use PMACTuningPro2 Automatic or Interactive to find the best position-loop gains for your system.

Absolute phase and power-up/reset position

Absolute Servo Power-On Position Address and Format: Ixx10, Ixx95

To read an SSI encoder for absolute servo position, Ixx10 is set to the address of that channel's position register. Ixx95 is set according to the specification of the SSI encoder (how many bits, signed or unsigned value…etc). The motor offset variable Ixx26 contains the difference between the absolute position and the resulting motor position (if any).

```
I110= $78C00                  ; Absolute Servo power-on position address
I195= $A00000                 ; Signed, 32 bits
```

Absolute Phase Power-On Position Address and Format: Ixx81, Ixx91

To read an R/D converter for absolute phase position, Ixx81 is set to the address of that channel's position register. Ixx91 is set according to the specification of the SSI encoder. Please note that this is only possible if the number of counts in one electrical cycle is less than 2^{24}.

```
I181= $78C00                  ; Commutation position address
I191= $180000                 ; 24 bits
```

Motor Phase Offset: Ixx75

Ixx75 holds the distance between the zero position of an absolute encoder used for power-on phase position (specified by Ixx81 and Ixx91) and the zero position of Turbo PMAC's commutation cycle. The proper value for this parameter can be found following the procedure explained in Turbo User Manual.

EnDat 2.2 Feedback Setup Example:

The following example demonstrates how to setup a 37-bit binary EnDat 2.2 encoder for position control of a brushless motor on the first channel of an Acc-84E. Assume that the documentation for the encoder suggests 1MHz clock for the length of the cable that we have in the system:
Channel is reading a 37 bit EnDat2.2 Encoder. Note that the full 37-bit encoder data is used for absolute power-on position but the commutation/on-going position is limited to 24 bits by the Encoder Conversion Table (ECT).

X:$78C0F

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M Divisor								N Divisor				Reserved		Trigger Clock	Trigger Edge	Trigger Delay				Protocol Code			
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
0				0				2				0				0				3			

```
WX:$78C0F,$002003             ; Global Control register, 1 MHz Clock setting
```

Channel 1:X:$78C00

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		Command Code						Reserved		Trigger Mode	Trigger Enable	Reseved	RxData Ready/ SENC	Reserved				Position Bits					
0	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
0				7				1				4				2				5			

```
WX:$78C00,$071425             ; Ch1 Control register, 37-Bit EnDat Encoder
```

Encoder conversion table setup required for EnDat 2.2 encoder connected to the first channel on Acc-84E at base address set to $78C00 will be as follows:

```
I8000=$278C00                 ; Unfiltered parallel position of location
                      ; Y:$78C00, no shifting
I8001=$18000                  ; 24-bit processed result at $3502
I103=$3502                    ; position loop feedback address
I104=$3502                    ; velocity loop feedback address
```

Since the number of counts in EnDat 2.2 encoders usually are much higher than normal incremental encoders, the default settings for position and velocity feedback scale factors (a value of 96) can cause resolution restrictions on Servo gain settings. It is recommended that the scale factors be set to a smaller value.

```
I108=1         ; Motor1 position scale factor required not to saturate the Velocity
I109=1         ; Motor1 velocity-loop scale factor
```

Assigning values to the control registers should be performed upon power-up/reset in the initialization PLC.

```
Open plc 1 Clear
Disable plc 2..31
cmd"wx:$78C0F,$002003"   ; Global Control register, 1 MHz Clock setting
cmd"wx:$78C00,$071425"   ; Channel 1, read 37 bits
Disable plc 1
Enable plc 2..31
```

```
Close
```

Absolute phase and power-up/reset position

Knowing the difference between the absolute encoder position and the commutation cycle zero (stored in Ixx75 in PMAC), a phasing routine is no longer necessary on power-up/reset. The following procedure to find Ixx75 is done only once per channel while setting up the machine for the first time, assuming the mechanics and electronics are not to be changed and have not failed, been replaced or repaired:
Set Ixx79=500 and Ixx29=-500
Increase these values by increments of 100 until motor movement is observed when O0 is issued. range is 100 to Ixx69.
Issue a #nO0, wait for motor to stop moving
Set Ixx29=0, wait for motor to stop moving
Set Mxx71 to zero (see suggested M-variables)
Read position data directly from channel position register.
For the Ch1 37-bit EnDat2.2 example, we need to construct the 37-bit position data from 24 bits at Y:$78C00 [23:0] and 13 bits at Y:$78C01[12:0].
See plc example below.
Set Ixx75 to that value
Set Ixx79=0
Issue a #nK to kill the motor
Assuming that I175=3000 and knowing that we have 16777215 counts per electrical cycle, this PLC example shows how to construct the 37-bit position word

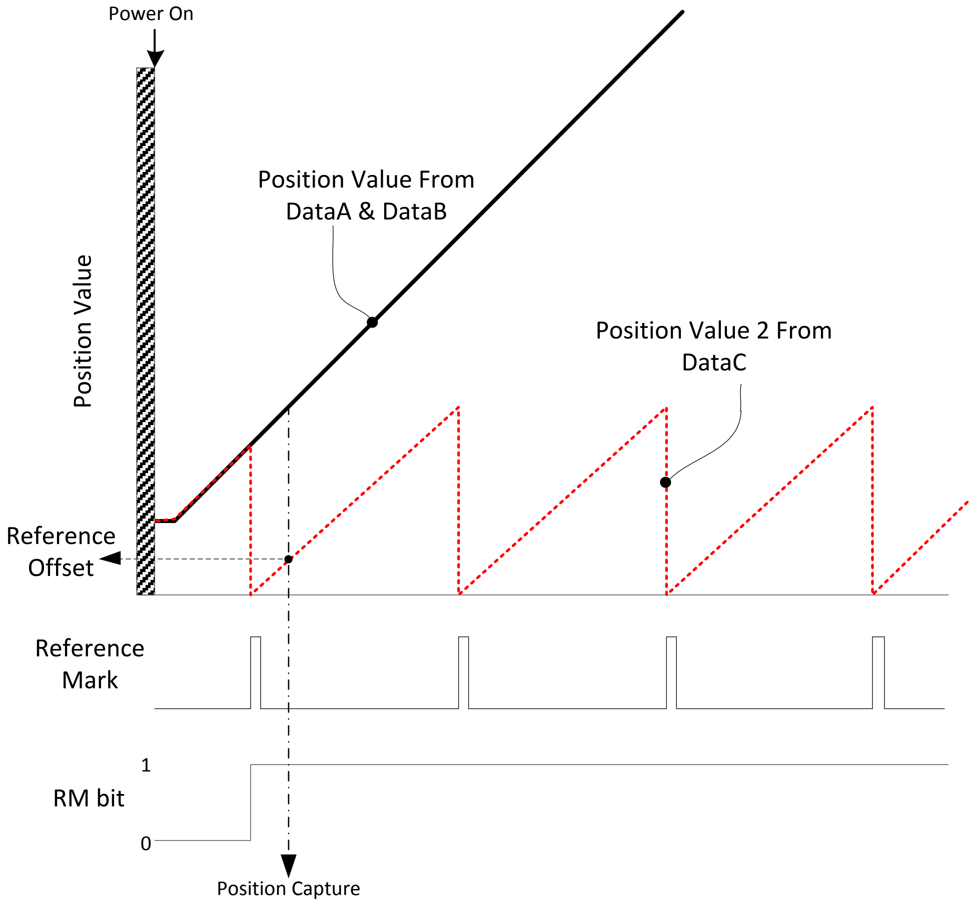
```
#define EnDat_pos_low             M1000
#define EnDat_pos_high            M1001
EnDat_pos_low->Y:$78C00,0,24,U    ; First 24 bits data, register A
EnDat_pos_high->Y:$78C01,0,16,U   ; Rest of data, lower13 bits register B
#define Phase_Offset              3000
#define msec *8388608/I10While(I5111>0)Endw

Open plc 2 clear
I5111=1000 msec                                  ; 1 sec delay
P1000= EnDat_pos_high&$1FFF                      ; Low 13 bits. Mask register B
If (P1000<$1000)                                 ; Positive encoder data?
    M162= (P1000*$1000000+ EnDat_pos_low)*I108   ; Actual position
Else                                             ; Negative encoder data?
    P1001= EnDat_pos_low ^$FFFFFF                ; XOR 24 bits
    P1002=P1000^$1FFF                            ; XOR 13 bits
    M162=-(P1002*$1000000+P1001+1)*I108          ; Actual position
    M148=0                                       ; Clear phasing search error
Endif
M171= (M162+ Phase_Offset )%(I171/I170)          ; Phase position
Disable plc 2
Close
```

EnDat 2.2 with Incremental Encoders:

In principal, incremental encoders transmit relative position values. After the encoder is powered up, the position value is 0 and a subdivided position value resulting from the interpolation of the current signal is transmitted.
Whereas relative position values can be transmitted immediately after switch-on, in order to receive absolute position values you must traverse a reference mark (RM), or two reference marks in sequence for encoders with distance-coded reference marks. This traversing is recorder in RM status bit.
The information containing:

- Reference run finished, i.e. absolute position value available and
- Position value 2

Can be requested with the "Encoder to send position values with additional information" command code. Then RM status bit in the transmission protocol indicates whether reference run has finished. If this is the case, position value 2 is available. Until this time, a relative position value is transmitted when position value 2 is requested.



The figure is for incremental rotary encoder with one reference mark, and position value 2 will be the absolute position relative to the reference mark in one revolution after reaching the first reference mark. Position value 2 is captured after reaching the first reference mark and written to the actual motor position register. By setting reference offset as current encoder position, reference mark position will become 0 count position.

Position value 2 can be obtained by sending proper command codes. The structure of position value 2 is as below.

Position Value 2, 48-bit		
Position Value 2 High Word 3	Position Value 2 Word 2	Position Value 2 Word 1
SerialEncoderDataC [15:0]	SerialEncoderDataC [15:0]	SerialEncoderDataC [15:0]
Command code $44	Command code $43	Command code $42

To obtain position value 2, three command codes need to be sent in sequence and data needs to be read in sequence. For each command, the data will be sent back in lower two bytes of Serial Encoder Data Register C as additional information 1 (byte1) and (byte2).

EnDat 2.2 Reference Mark Setup Example:

In this example, a Heidenhain ROD 486 encoder with 1024 lines is connected to a EIB 192 with 16384 subdivisions with EnDat 2.2. The feedback from EIB 192 to ACC84E is as a 24-bit encoder (1024*16384=2^24). ACC-84E is at base address $78C00. The goal is to establish 0 count position after reaching the first reference mark, and users need to either manually rotate the motor shaft or jog the motor after enabling the following PLC.

Encoder conversion table setting for Motor #1:

```
I8000=$278C00                ; Unfiltered parallel position of location
                             ; Y:$78C00, no shifting
I8001=$18000                 ; 24-bit processed result at $3502
I103=$3502                   ; Motor #1 position loop feedback address
I104=$3502                   ; Motor #1 velocity loop feedback address
```

PLC Program:

```
    //====== NOTES ABOUT THIS PLC EXAMPLE =====//
    // This PLC example utilizes:
    //         M6000 through M6008, M160, M165, M1060
    //         Suggested M-Variable M114, M162
    //         P2000 through P2009
    //         Coordinate system 16 Timer 2
    // Make sure that current and/or future configurations do not create conflicts with
    // these parameters.
    //=========================================//

    M6000..6010->*            ; Self-referenced M-Variables
    M6000..6010=0             ; Reset at download

    //====== GLOBAL CONTROL REGISTERS =========//
    #define    EnDatGlobalCtrl1_4     M6000  ; Channels 1-4 EnDat global control register
    EnDatGlobalCtrl1_4->X:$78C0F,0,24,U      ; Channels 1-4 EnDat global control register

    //====== CHANNEL CONTROL REGISTERS ========//
    #define    Ch1EnDatCtrl           M6001  ; Channel 1 EnDat control register
    Ch1EnDatCtrl->X:$78C00,0,24,U            ; Channel 1 EnDat control register

    //====== Define Data Registers ============//
    #define    SerialEncDataA         M6002  ; Ch#1 Position 1 Data A
    #define    SerialEncDataB         M6003  ; Ch#1 Position 1 Data B
    #define    SerialEncDataC         M6004  ; Ch#1 Position 2 Data C
    #define    SerialEncDataD         M6005  ; Ch#1 Position 2 Data D
    #define    SerialEncDataC_AddInfo M6006  ; Ch#1 Position 2 Data C Additional Info.
    #define    RM_bit                 M6007  ; Ch#1 Position 2 Data C RM bit
    #define    MRS_code               M6008  ; Ch#1 Position 2 Data C Ack. of MRS_code

    #define    Mtr1AmpEna             M114   ; Motor#1 Amp Enable bit
    #define    Mtr1ActPos             M162   ; Motor#1 Actual Position
    #define    Mtr1DesVel             M165   ; Motor#1 Desired Velocity
    #define    Mtr1ActVel             M166   ; Motor#1 Actual Velocity
    #define    Mtr1DesVel_unit        M160   ; Motor#1 Desired Velocity unit
    #define    Mtr1DesVel_fraction    M1060  ; Motor#1 Desired Velocity fraction

    #define    Pos1_Value             P2000  ; Position 1 Value
    #define    Pos2_Low               P2001  ; Position 2 Value Word 1
    #define    Pos2_Mid               P2002  ; Position 2 Value Word 2
    #define    Pos2_High              P2003  ; Position 2 Value Word 3
    #define    Pos2_Value             P2004  ; Position 2 Value
    #define    Index_Offset           P2005  ; Reference Mark Offset
    #define    SerialEncDataA_Capt    P2006  ; Position 1 Data A Capture Value
    #define    SerialEncDataB_Capt    P2007  ; Position 1 Data B Capture Value

    #define    InitialEnaStatus       P2008  ; Initial Motor Status: Enable/Disable
```

```
#define    FaultFlag             P2009   ; Data Receiving Timeout Flag

#define    Timer                 I6612   ; Use Coord#32 Timer 2
#define    msec                  *8388607/i10 While (I6612 > 0) EndWhile

//====== Address Assignment ======//
SerialEncDataA->Y:$78C00,0,24,U
SerialEncDataB->Y:$78C01,0,24,U
SerialEncDataC->Y:$78C02,0,24,U
SerialEncDataD->Y:$78C03,0,24,U
SerialEncDataC_AddInfo->Y:$78C02,0,16,U
RM_bit->Y:$78C02,22
MRS_code->Y:$78C02,16,4

Mtr1AmpEna->X:$078205,14                ; AENA1 output status
Mtr1ActPos->D:$00008B                   ; #1 Actual position (1/[Ixx08*32] cts)
Mtr1ActVel->X:$00009D,0,24,S            ; #1 Actual velocity (1/[Ixx09*32]cts/ms)
Mtr1DesVel_unit->X:$000086,0,24,S       ; #1 Desired cmd vel register, X-register
                                        ; units 3/[Ixx08*32] cts/msec at %100
Mtr1DesVel_fraction->Y:$000086,0,24,U   ; #1 Desired cmd vel register (Fractional)

//====== PLC Program Start ======//
Open PLC 3 Clear

FaultFlag=0                      ; Clear FaultFlag
EnDatGlobalCtrl1_4=$2003         ; 1MHz for Channel #1~4, cannot set higher
Ch1EnDatCtrl=$381418             ; 24-bit read, DataA[24], DataB[0], Position 1
Timer = 1 msec                   ; Wait for 1 msec
Ch1EnDatCtrl=$421418             ; Request Info in DataC, Pos 2 Word 1
Timer=1 msec                     ; Wait for 1 msec

While (RM_bit = 0)EndWhile       ; Capture RM_bit, wait for RM_bit to be 1

If (Mtr1AmpEna = 1)              ; Check initial motor status
  cmd"#1j/"                      ; If reaching RM by jogging, then jog stop
  InitialEnaStatus=1             ; Set Status as 1
Else
  cmd"#1k"                       ; If reaching RM by hand: kill; this is redundant
  InitialEnaStatus=0             ; Set Status as 0
EndIf

Mtr1DesVel = Mtr1DesVel_unit*3/(I108*32)+Mtr1DesVel_fraction/1677216   ; Desired Vel.

While (Mtr1DesVel > 10)          ; Wait for motor to settle
  Mtr1DesVel = Mtr1DesVel_unit*3/(I108*32)+Mtr1DesVel_fraction/1677216
                                 ; Check Velocity while waiting
EndWhile

//================== Position 2 Data Value read ==================//
I6612 = 10 *8388607/I10                         ; 10 msec delay for Data C
While (I6612 > 0 Or MRS_code != 2)EndWhile      ; Timeout Or when MRS code is 2
If (MRS_code = 2)                               ; When MRS code is 2,
  Pos2_Low=SerialEncDataC_AddInfo               ; Read Word 1 for Position 2
  SerialEncDataA_Capt=SerialEncDataA            ; Read DataA for Position 1
  SerialEncDataB_Capt=SerialEncDataB&$00FFFF    ; Read DataB for Position 1
Else
  FaultFlag=1                                   ; Timeout fault
EndIf

Ch1EnDatCtrl=$431418                            ; Request Info in DataC, Pos 2 Word 2
I6612 = 10 *8388607/I10                         ; 10 msec delay for Data C
While (I6612 > 0 Or MRS_code != 3)EndWhile      ; Timeout Or when MRS code is 3
If (MRS_code = 3 And FaultFlag = 0)             ; When MRS code is 3
  Pos2_Mid=SerialEncDataC_AddInfo               ; Read Word 2 for Position 2
Else
  FaultFlag=1                                   ; Timeout fault
EndIf

Ch1EnDatCtrl=$441418                            ; Request Info in DataC, Pos 2 Word 3
I6612 = 10 *8388607/I10                         ; 10 msec delay for Data C
While (I6612 > 0 Or MRS_code != 4)Endwhile      ; Timeout Or when MRS code is 4
```

```
    If (MRS_code = 4 And FaultFlag = 0)              ; When MRS code is 4
      Pos2_High=SerialEncDataC_AddInfo               ; Read Word 3 for Position 2
    Else
      FaultFlag=1                                    ; Timeout fault
    EndIf


    If (FaultFlag = 0)                               ; If reading is successful
      Pos2_Value = Pos2_Low + Pos2_Mid*$10000 + Pos2_High*$100000000
                                                     ; Construct actual Position 2 value
      Timer = 10 msec                                ; Wait 10 msec
      Ch1EnDatCtrl=$381418                           ; Change mode back to DataA reading

      If (InitialEnaStatus = 1)                      ; If initially enable
        cmd"#1k"                                     ; Kill motor 1
        While (Mtr1ActVel > 10) EndWhile             ; Wait for motor to settle
        Mtr1ActPos = Pos2_Value*(I108*32)            ; Write offset to Motor#1 Act. Pos.
        Timer = 10 msec                              ; 10 msec delay
        cmd"#1j/"                                    ; Closed-loop in position
      Else                                           ; If initially motor is killed
        While (Mtr1ActVel > 10) EndWhile             ; Wait for motor to settle
        Mtr1ActPos = Pos2_Value*(I108*32)            ; Write offset to Motor#1 Act. Pos.
      EndIf
    EndIf

    Disable PLC 3

    Close
```

Since the number of counts in EnDat 2.2 encoders usually are much higher than normal incremental encoders, the default settings for position and velocity feedback scale factors (a value of 96) can cause resolution restrictions on Servo gain settings. It is recommended that the scale factors be set to a smaller value.

```
I108=1         ; Motor1 position scale factor required not to saturate the Velocity
I109=1         ; Motor1 velocity-loop scale factor
```

Yaskawa Sigma II/III/V Feedback Setup Example

Channel Control Register Setup for Position Read

In this mode the Brick will update the data registers based upon the data received from the encoder based upon the trigger clock (servo or phase clock based upon the global register setting. Default is set to phase clock).

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command Code								Reserved		Trigger Mode	Trigger Enable	Reserved	RxDataReady / SENC_Mode	Rereserved	Reset Mode	Encoder Address							
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0				0				1				4				0				0			

The setup value can be written to the memory as a part of your start up PLC.

```
// Yaskawa Feedback Startup Example PLC:
Open PLC 1 clear
Disable PLC2..31
CMD”WX:$78C00,$1400"
CMD”WX:$78C04,$1400"
CMD”WX:$78C08,$1400"
CMD”WX:$78C0C,$1400"
Disable plc1
Close
```

Channel Specific Control Register Setup for Reset Mode

Yaskawa absolute encoders can generate fault flags which are latched and the only way to reset them is through this procedure. For a list of possible faults on Yaskawa absolute encoders and where to read them, please check the following section titled “Alarm Codes.”

To send a RESET command to the encoder, the channel control register needs to be modified a few times.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command Code								Reserved		Trigger Mode	Trigger Enable	Reserved	RxDataReady / SENC_Mode	Rereserved	Reset Mode	Encoder Address							

Reset Mode

0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	1
0				4				3				5				0				1			

NOP

0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0
0				0				3				5				0				1			

Position Read

0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0				0				1				4				0				0			

Essentially, the following commands are available for any of the channels:

```
// Yaskawa RESET Commands for channel 1 of ACC-84E with base address of $78C00
CMD”WX:$78C00,$043501"
CMD”WX:$78C00,$003501"
CMD”WX:$78C00,$001400"
```

This can be done through any of the PLCs. However, there is some handshaking required in order to make sure the RESET command is completed before the next command is sent down.
The following PLC shows an example on how to do a reset including the handshaking necessary with the encoder to ensure a proper reset.

```
// Yaskawa Absolute Encoder RESET Example PLC:

#define Chn1CtrlReg  M1011
Chn1CtrlReg->X:$78C00,0,24

#define Chn1Flags    M1021
Chn1Flags->Y:$78C01,0,24

#define Chn1Alarms   M1031
Chn1Alarms->Y:$78C02,8,8

Open PLC 10 clear
Chn1CtrlReg = $1400                 ;Make sure the channel is in Position Read Mode
I6612 = 100 * 8388607/i10           ; 100 msec timer
While(I6612>0)
Endwhile
If ((Chn1Alarms & $3)= 0)           ;If there is no Alarm, don't reset
        Disable PLC 10
Endif
p0=1
Chn1CtrlReg =$043501                ;Sending the RESET command on a single trigger
I6612 = 100 * 8388607/i10           ; 100 msec timer
While(I6612>0)
Endwhile
While ( (Chn1CtrlReg & $1000) = 1)  ;wait for the trigger to happen
Endwhile
While ((Chn1Flags & $100)=1)        ;Busy Signal on bit 8 of second data register
        if ((Chn1Flags & $800000)=1)  ; If timed out
                Chn1CtrlReg = $1400
                Disable PLC 10
        Endif
Endwhile
p0=2
Chn1CtrlReg = $003501               ;Sending the NOP command on a single trigger
I6612 = 100 * 8388607/i10           ; 100 msec timer
While(I6612>0)
Endwhile
While ( (Chn1CtrlReg & $1000) = 1)  ;wait for the trigger to happen
Endwhile
While ((Chn1Flags & $100)=1)        ;Busy Signal on bit 8 of second data register
        if ((Chn1Flags & $800000)=1)  ; If timed out
                Chn1CtrlReg = $1400
                Disable PLC 10
        Endif
Endwhile
p0=3
Chn1CtrlReg = $1400
Disable PLC 10
Close
```

Encoder Specific Settings

Yaskawa Sigma II & Sigma III protocol includes 5 feedback types with different resolutions and incremental / absolute modes. All of these feedbacks are supported by ACC-84E.

16-Bit Yaskawa Sigma II Absolute Encoder:

Y:Base+$1																Y:Base+$0																							
				11											0	23			20	19															4				
				M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0				
0				Multi-turn Position (16-bits)																Absolute Single Turn Data (16-bits)																undetermined			

Encoder Conversion Table Setup for on-going servo position and commutation angle:

Channel	ECT Line	Settings
1st Channel	1st Line	$200000 + Base Address + $0
	2nd Line	$020004
2nd Channel	1st Line	$200000 + Base Address + $4
	2nd Line	$020004
3rd Channel	1st Line	$200000 + Base Address + $8
	2nd Line	$020004
4th Channel	1st Line	$200000 + Base Address + $C
	2nd Line	$020004

Example:

16-bit absolute encoder on channel 1 of ACC-84E with a base address of $78C00:

```
I8000=$278C00
I8001=$020004
```

Absolute Position Reading:

In order to read the absolute position from the encoder and set the motor position accordingly, the data available in the **EncoderDataRegisterA** and **EncoderDataRegisterB** should be combined together. The following example demonstrates required calculations. This PLC needs to be executed once after system power-up/reset.

```
#define STD0_15         M1000
#define MTD0_3          M1001
#define MTD4_15         M1002
#define MTD0_15         M1003

STD0_15->Y:$78C00,4,16
MTD0_3->Y:$78C00,20,4
MTD4_15->Y:$78C01,0,12
MTD0_15->*

#define Mtr1ActPos      M162

open plc 28 clear
MTD0_15 = MTD4_15 * $10 + MTD0_3
If (MTD0_15>$7FFF)
        MTD0_15 = (MTD0_15^$FFFF + 1)*-1
        If (STD0_15 !=0)
                STD0_15 = (STD0_15^$FFFF + 1)*-1
        Endif
Endif
Mtr1ActPos = ((MTD0_15 * $10000)+ STD0_15) * I108 * 32
disable plc 28
close
```

Reading Absolute Phase Position

For commutation purpose, since the data doesn't start on bit 0 of the register, we have to use the output of the encoder conversion table for on-going phase position instead of the position register itself.

Since the output value of the ECT is already shifted left by 5-bits, the value in the Ixx71 would be equal to 65536 x 32 = 2097152. and Ixx70 would be equal to the number of pole pairs on the motor. Also, Ixx83 should be pointing to the correct ECT entry to read the ongoing position data.

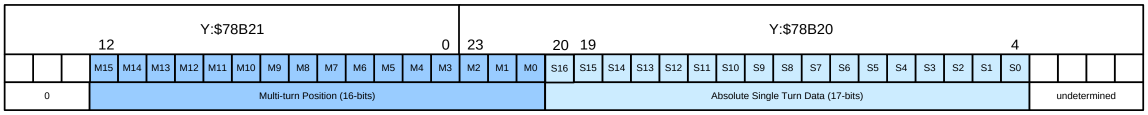
If the user wants to use the absolute feedback for power-on phasing with no motion, a similar approach would be used however the single turn data would be sufficient for phasing the motor.

Here is an example of how to determine the power-on phasing based on absolute data. The following procedure is only required once. After determining the phase reference value, a power-on PLC is sufficient to establish the phase reference and motor will be ready for commutation.

1. Tune the current loop on the motor (after setting correct values for Ixx00, Ixx01, Ixx24, Ixx82, Ixx84, Ixx57, Ixx58, Ixx69 and Ixx66, use the tuning software and tune the current loop i.e. Ixx61, Ixx62 and Ixx76).
2. Set a positive value (usually 10% of Ixx66) to Ixx79 and set Ixx29=0 and Issue an O0 command (open loop, zero output).
3. Read the single turn data (for the first channel, the data would be at Y:$78B20,4,16).
4. Set the Ixx79 back to its original value and issue a kill.
5. The following PLC will set up the phase reference.

```
#define Mtr1PhaseRef       P184
  Mtr1PhaseRef = 5461   ; This value should work for all 16-bit absolute encoders
(Yaskawa aligns their U phase with index)
#define Mtr1PhasePos       M171
  Mtr1PhasePos->X:$B4,24,S
#define Mtr1PhaseErr       M148
  Mtr1PhaseErr->Y:$C0,8
#define Mtr1CommSize       I171
  Mtr1CommSize = 65536
#define Mtr1CommCycles     I170
  Mtr1CommCycles = 3
#define Mtr1Commutate      I101
  Mtr1Commutate = 1              ;Pmac is commutating the motor, data is in the X
register
#define Mtr1CommFdbkAdr    I183
  Mtr1CommFdbkAdr = $3502       ;Address is the second entry in the encoder conversion
table
#define Mtr1STD0_15        M180
  Mtr1STD0_15->Y:$78C00,4,16

Open plc 29 clear
;Mtr1 Set Phase Position
Mtr1PhasePos = ( Mtr1STD0_15 % (Mtr1CommSize/Mtr1CommCycles) - Mtr1PhaseRef ) * 32 *
Mtr1CommCycles
Mtr1PhaseErr = 0;
disable plc 29
```

17-Bit Yaskawa Sigma II Absolute Encoder:


Y:$78B21
12
0
23
20
19
Y:$78B20
4
M15 M14 M13 M12 M11 M10 M9 M8 M7 M6 M5 M4 M3 M2 M1 M0
S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
0
Multi-turn Position (16-bits)
Absolute Single Turn Data (17-bits)
undetermined

Encoder Conversion Table Setup for on-going servo position and commutation:

Channel	ECT Line	Settings
1st Channel	1st Line	$200000 + Base Address + $0
	2nd Line	$021004
2nd Channel	1st Line	$200000 + Base Address + $4
	2nd Line	$021004
3rd Channel	1st Line	$200000 + Base Address + $8
	2nd Line	$021004
4th Channel	1st Line	$200000 + Base Address + $C
	2nd Line	$021004

Example:

17-bit absolute encoder on channel 1 of ACC-84E with a base address set to $78C00:

```
I8000=$278C00
I8001=$021004
```

Reading Absolute Position

In order to read the absolute position from the encoder and set the motor position accordingly, the data available in the **EncoderDataRegisterA** and **EncoderDataRegisterB** should be combined. The following example demonstrates required calculations. This PLC needs to be executed once after system power-up/reset.

```
#define FirstWord        M1000
#define SecondWord       M1001
#define STD0_16          M1002
#define MTD0_15          M1003

FirstWord->Y:$78C00,0,24
SecondWord->Y:$78C01,0,4
STD0_16->*
MTD0_15->*

#define Mtr1ActPos       M162

open plc 28 clear
MTD0_15 = (SecondWord & $1FFF) * $8 + int (Firstword / 2097152)
STD0_16 = int ((FirstWord & $1FFFF0) / 16)
If (MTD0_15>$7FFF)
       MTD0_15 = (MTD0_15^$FFFF + 1)*-1

       If (STD0_16 !=0)
              STD0_16 = (STD0_16^$1FFFF + 1)*-1
       Endif
Endif
Mtr1ActPos = ((MTD0_15 * $20000)+ STD0_16) * I108 * 32
disable plc 28
close
```

Reading Absolute Phase Position

For commutation purpose, since the data does not start on bit 0 of the register, we have to use the output of the encoder conversion table for on-going phase position instead of the position register itself. This means that the Servo Clock and the Phase Clock should be the same. Servo Cycle Extension Period

(Ixx60) can be used to lower the CPU load and not to face quantization errors on the PID loops if the high Servo rates cause problems.
Since the output value of the ECT is already shifted left by 5-bits, the value in the Ixx71 would be equal to 131072 x 32 = 2097152. and Ixx70 would be equal to the number of pole pairs on the motor. Also Ixx83 should be pointing to the correct ECT entry to read the ongoing position data.
If the user wants to use the absolute feedback for power-on phasing with no motion, a similar approach would be used. However, the single turn data would be sufficient for phasing the motor.
Here is an example of how to determine the power-on phasing based on absolute data. The following procedure is only required once. After determining the phase reference value, a power-on PLC would be sufficient to establish the phase reference and motor will be ready for commutation.

1. Tune the current loop on the motor (after setting correct values for Ixx00, Ixx01, Ixx24, Ixx82, Ixx84, Ixx57, Ixx58, Ixx69 and Ixx66, use the tuning software and tune the current loop i.e. Ixx61, Ixx62 and Ixx76).
2. Set a positive value (usually 10% of Ixx66) to Ixx79 and set Ixx29=0 and Issue an O0 command (open loop, zero output).
3. Read the Single turn data. (for the first channel, the data would be at Y:$78B20,0,24 but you have to apply the following function on it *int ((FirstWord & $1FFFF0) / 16))*.
4. Set the Ixx79 back to its original value and issue a kill.
5. The following PLC will set up the phase reference.

```
#define Mtr1PhaseRef       P184
  Mtr1PhaseRef = 5461   ; This value should work for all 16-bit absolute encoders
(Yaskawa aligns their U phase with index)
#define Mtr1PhasePos       M171
  Mtr1PhasePos->X:$B4,24,S
#define Mtr1PhaseErr       M148
  Mtr1PhaseErr->Y:$C0,8
#define Mtr1CommSize       I171
  Mtr1CommSize = 131072
#define Mtr1CommCycles     I170
  Mtr1CommCycles = 3
#define Mtr1Commutate      I101
  Mtr1Commutate = 1              ;Pmac is commutating the motor, data is in the X
register
#define Mtr1CommFdbkAdr    I183
  Mtr1CommFdbkAdr = $3502       ;Address is the second entry in the encoder conversion
table
#define Mtr1STD0_15        M180
  Mtr1STD0_15->Y:$78C00,0,24

Open plc 29 clear
;Mtr1 Set Phase Position
Mtr1PhasePos = ( int((Mtr1STD0_15&$1FFFF0)/$F) % (Mtr1CommSize/Mtr1CommCycles) -
Mtr1PhaseRef ) * 32 * Mtr1CommCycles
Mtr1PhaseErr = 0;
disable plc 29
```

20-Bit Yaskawa Sigma III Absolute Encoder:

Y:$78B21																Y:$78B20																							
15															0	23															4								
M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	S19	S18	S17	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0				
Multi-turn Position (16-bits)																Absolute Single Turn Data (17-bits)																				undetermined			

Encoder Conversion Table Setup for on-going servo position and commutation:

Channel	ECT Line	Settings
1st Channel	1st Line	$200000 + Base Address + $0
	2nd Line	$024004
2nd Channel	1st Line	$200000 + Base Address + $4
	2nd Line	$024004
3rd Channel	1st Line	$200000 + Base Address + $8
	2nd Line	$024004
4th Channel	1st Line	$200000 + Base Address + $C
	2nd Line	$024004

Example:
20-bit absolute encoder on channel 1 of ACC-84E with a base address set to $78C00:

```
I8000=$278C00
I8001=$024004
```

Reading Absolute Position

In order to read the absolute position from the encoder and set the motor position accordingly, the data available in the **EncoderDataRegisterA** and **EncoderDataRegisterB** should be combined. The following example demonstrates required calculations. This PLC needs to be executed once after system power-up/reset.

```
#define FirstWord        M1000
#define SecondWord       M1001
#define STD0_19          M1002
#define MTD0_15          M1003

FirstWord->Y:$78C00,0,24
SecondWord->Y:$78C01,0,4
STD0_19->*
MTD0_15->*

#define Mtr1ActPos       M162

open plc 28 clear
MTD0_15 = (SecondWord & $FFFF)
STD0_19 = int ((FirstWord & $FFFFF0) / 16)
If (MTD0_15>$7FFF)
       MTD0_15 = (MTD0_15^$FFFF + 1)*-1

       If (STD0_19 !=0)
              STD0_19 = (STD0_19^$FFFFF + 1)*-1
       Endif
Endif
Mtr1ActPos = ((MTD0_15 * $100000)+ STD0_19) * I108 * 32
disable plc 28
close
```

Reading Absolute Phase Position

For commutation purpose, since the data does not start on bit 0 of the register, we have to use the output of the encoder conversion table for on-going phase position instead of the position register itself. This means that the Servo Clock and the Phase Clock should be the same. Servo Cycle Extension Period (Ixx60) can be used to lower the CPU load and not to face quantization errors on the PID loops if the high Servo rates cause problems.

Since the output value of the ECT is already shifted left by 5-bits, the value in the Ixx71 would be equal to 2^20 x 32 = 33554432, but the maximum valid value for the Ixx71 is 16777216 which is half the value we require. In this case we would use a ratio between Ixx71 and Ixx70. As an example, assume a 20-bit encoder is mounted on a Yaskawa motor which has 4 pole pairs, in this case we would set Ixx70 = 1 and Ixx71 = 33554432/4 = 8388608 .

If the user wants to use the absolute feedback for power-on phasing with no motion, a similar approach would be used however the single turn data would be sufficient for phasing the motor.

Here is an example of how to determine the power-on phasing based on absolute data. The following procedure is only required once. After determining the phase reference value, a power on PLC would be sufficient to establish the phase reference and motor will be ready for commutation.

1. Tune the current loop on the motor (after setting correct values for Ixx00, Ixx01, Ixx24, Ixx82, Ixx84, Ixx57, Ixx58, Ixx69 and Ixx66, use the tuning software and tune the current loop i.e. Ixx61, Ixx62 and Ixx76).
2. Set a positive value (usually 10% of Ixx66) to Ixx79 and set Ixx29=0 and Issue an O0 command (open loop, zero output).
3. Read the Single turn data. (for the first channel, the data would be at Y:$78B20,0,24 but you have to divide the number by 16 to shift the data 4-bits to right.
4. Set the Ixx79 back to its original value and issue a kill.
5. The following PLC will set up the phase reference.

```
#define FirstWord        M1000
#define STD0_19          M1002

FirstWord->Y:$78C00,0,24
STD0_19->*

#define Mtr1PhasePos      M171     ; Suggested M-Variable definition
#define Mtr1PhaseSrchErr  M148     ; Suggested M-Variable definition
#define MeasPhaseRef      30000    ; Measured Single Turn Value based on the test above

open plc 29 clear
STD0_19 = int ((FirstWord & $FFFFF0) / 16);
if (STD0_19 !< MeasPhaseRef)
   Mtr1PhasePos = ( STD0_19 - MeasPhaseRef ) * 32 ;
Else
   Mtr1PhasePos = ( 1048576 - MeasPhaseRef + STD0_19 ) * 32;
EndIf
Mtr1PhaseSrchErr = 0;
disable plc 28
close
```

17-Bit Yaskawa Sigma II Incremental Encoder:

Y:$78B21

					10										0
					S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6
Undetermined					Incremental Compensation (11-bits)										

Y:$78B20

	22																6			3	2	1	0
	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0			U	V	W	Z
N	Incremental Position in Single Turn (17-bits)																	Undet.		Hall Signals			Z

Encoder Conversion Table Setup for on-going servo position and commutation:

Channel	ECT Line	Settings
1st Channel	1st Line	$200000 + Base Address + $0
	2nd Line	$011006
2nd Channel	1st Line	$200000 + Base Address + $4
	2nd Line	$011006
3rd Channel	1st Line	$200000 + Base Address + $8
	2nd Line	$011006
4th Channel	1st Line	$200000 + Base Address + $C
	2nd Line	$011006

Example:

17-bit incremental encoder on channel 1 of ACC-84E with base address set to $78C00:

```
I8000=$278C00
I8001=$011006
```

Homing of incremental encoder based on its index

This section explains how to use the encoder internal flags and data for homing to internal index pulse which happens once per revolution. If the user simply wants to home to an external home flag or limit flag, this can be achieved by using I7mn2 and I7mn3 settings and doing a software capture based upon Ixx97=1 (software capture is required since the gate array doesn't see the encoder counts directly a hardware capture is not possible). A combination of external flag and incremental index pulse is also possible.

In order to use the internal index pulse of the encoder and its flags, the following steps should be followed:

1. Bit 14 of the alarm indicates whether the index has been detected since the last power-up or not.
2. The motor should be jogged until the bit 14 of the alarm codes becomes low.
3. Once this bit is low, the encoder will place the "incremental compensation" value in the lower 11 bits of the second word.
4. By subtracting the "incremental compensation" from the "incremental position," the true position from the index can be calculated.

The following code is an example on how to do the homing based upon the steps above. It is strongly recommended that home search moves be conducted at a slow speed.

```
#define FirstWord           M1000
#define SecondWord          M1001
#define OriginNotPassed     M1002

FirstWord->Y:$78C00,0,24
SecondWord->Y:$78C01,0,24
OriginNotPassed->Y:$78C02,14

#define Mtr1ActPos      M162    ; Suggested M-Variable Definition, Mtr 1 Actual Position

open plc 29 clear
if (OriginNotPassed = 1)
   cmd "#1j+"                  ; start moving toward the positive direction
   while (OriginNotPassed = 1)  ; until the index is detected
   endwhile
   cmd “#1k”
endif
while (Secondword & $8FF = 0)   ; there is a 2msec delay before inc. comp. is updated
endwhile
Mtr1ActPos = (int ((FirstWord & $8FFFC0) / $40) - (SecondWord & $8FF) * $40 ) * i108 * 32
disable plc 29
close
```

Power-on phase referencing using Hall sensors

The Hall sensor data comes back on bits 1, 2 and 3 of the first word. This data can be used in order to establish an estimated phase reference for the motor on power-up. However, hall phasing will have ±30° error, which can result in loss of up to 14 percent of the torque output, but usually this is good enough for moving the motor until a more accurate reference is established (homing) and phase position data is updated accordingly.

Here is an example of how to determine the power-on phasing based on hall data. The following procedure is only required once. After determining the phase reference value, a power-on PLC would be sufficient to establish the phase reference and motor will be ready for commutation. Once the motor is moving and a better position reference can be established (usually homing sequence), the phase position can be fine-tuned.

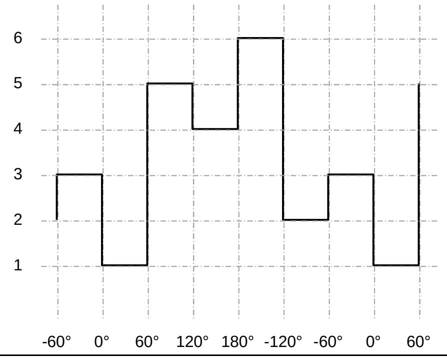
1. Tune the current loop on the motor (after setting correct values for Ixx00, Ixx01, Ixx24, Ixx82, Ixx84, Ixx57, Ixx58, Ixx69 and Ixx66, use the tuning software and tune the current loop i.e. Ixx61, Ixx62 and Ixx76).
2. Set the Ixx70 and Ixx71 based upon the type of encoder you have (for a 17-bit incremental encoder, Ixx71 = 131072 and Ixx70 = [number of pole pairs] and Ixx83 would be pointing to the correct Encoder Conversion Table Entry).
3. Set a positive value (usually 10 percent of Ixx66) to Ixx79 and set Ixx29=0 and Issue an O0 command (open loop, zero output).
4. Read the hall sensor data. This data can be read for the first encoder at Y:$78B20,0,4. We are interested only in bits 1 through 3 so the value should be shifted right by one bit or simply divide it by 2. This will be a number between 1 to 6 (a value of 0 or 7 is not valid)
5. Set Mxx71=0.
6. Set the Ixx79 back to its original value and issue a kill.
7. The following PLC will set up the phase reference.

Note

The Phase###Pos definition will change based on the number that is read during step 4 of the setup procedure explained earlier.

Step 4 Value	Definitions
1	#define Phase30Deg 1 #define Phase90Deg 5 #define Phase150Deg 4 #define Phase210Deg 6 #define Phase270Deg 2 #define Phase330Deg 3
2	#define Phase30Deg 2 #define Phase90Deg 3 #define Phase150Deg 1 #define Phase210Deg 5 #define Phase270Deg 4 #define Phase330Deg 6
3	#define Phase30Deg 3 #define Phase90Deg 1 #define Phase150Deg 5 #define Phase210Deg 4 #define Phase270Deg 6 #define Phase330Deg 2

Step 4 Value	Definitions
4	#define Phase30Deg 4 #define Phase90Deg 6 #define Phase150Deg 2 #define Phase210Deg 3 #define Phase270Deg 1 #define Phase330Deg 5
5	#define Phase30Deg 5 #define Phase90Deg 4 #define Phase150Deg 6 #define Phase210Deg 2 #define Phase270Deg 3 #define Phase330Deg 1
6	#define Phase30Deg 6 #define Phase90Deg 2 #define Phase150Deg 3 #define Phase210Deg 1 #define Phase270Deg 5 #define Phase330Deg 4



```
#define FirstWord        M1000
#define Halls            M1002

FirstWord->Y:$78C00,0,24
Halls->*

#define Mtr1PhasePos      M171     ; Suggested M-Variable definition
#define Mtr1PhaseSrchErr  M148     ; Suggested M-Variable definition

#define Phase30Deg        1
#define Phase90Deg        5
#define Phase150Deg       4
#define Phase210Deg       6
#define Phase270Deg       2
#define Phase330Deg       3

open plc 29 clear
Halls = int ((FirstWord & $E) / 2);
If (Halls = Phase30Deg)
   Mtr1PhasePos = I171 * 30 / 360;
Endif
If (Halls = Phase90Deg)
   Mtr1PhasePos = I171 * 90 / 360;
Endif
If (Halls = Phase150Deg)
   Mtr1PhasePos = I171 * 150 / 360;
Endif
If (Halls = Phase210Deg)
   Mtr1PhasePos = I171 * 210 / 360;
Endif
If (Halls = Phase270Deg)
   Mtr1PhasePos = I171 * 270 / 360;
Endif
```

```
If (Halls = Phase330Deg)
   Mtr1PhasePos = I171 * 330 / 360;
Endif
Mtr1PhaseSrchErr = 0;
disable plc 28
close
```

13-Bit Yaskawa Sigma II Incremental Encoder:

Y:$78B21

					10										0
					S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2
Undetermined					Incremental Compensation (11-bits)										

Y:$78B20

	22												10							3	2	1	0
	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0							U	V	W	Z
N	Incremental Position in Single Turn (13-bits)													Undetermined						Hall Signals			Z

Encoder Conversion Table Setup for on-going servo position and commutation:

Channel	ECT Line	Settings
1st Channel	1st Line	$200000 + Base Address + $0
	2nd Line	$00D00A
2nd Channel	1st Line	$200000 + Base Address + $4
	2nd Line	$00D00A
3rd Channel	1st Line	$200000 + Base Address + $8
	2nd Line	$00D00A
4th Channel	1st Line	$200000 + Base Address + $C
	2nd Line	$00D00A

Example:

13-bit incremental encoder on channel 1 of ACC-84E with base address set to $78C00:

```
I8000=$278C00
I8001=$00D00A
```

Homing of incremental encoder based on its index

This section explains how to use the encoder internal flags and data for homing to internal index pulse which happens once per revolution. If the user simply wants to home to an external home flag or limit flag, this can be achieved by using I7mn2 and I7mn3 settings and doing a software capture based upon Ixx97=1 (software capture is required since the gate array doesn’t see the encoder counts directly a hardware capture is not possible). Also, a combination of external flag and incremental index pulse is possible.

In order to use the internal index pulse of the encoder and its flags, the following steps should be followed:

1. Bit 14 of the alarm indicates whether the index has been detected since the last power-up or not.
2. The motor should be jogged until the bit 14 of the alarm codes becomes low.
3. Once this bit is low, the encoder will place the “incremental compensation” value in the lower 11 bits of the second word.
4. By subtracting the “incremental compensation” from the “incremental position” the true position from the index can be calculated.

The following code is an example of how to do the homing based upon the steps above. It is strongly recommended that home search moves be conducted at a slow speed.

```
#define FirstWord          M1000
#define SecondWord         M1001
#define OriginNotPassed    M1002

FirstWord->Y:$78C00,0,24
SecondWord->Y:$78C01,0,24
OriginNotPassed->Y:$78C02,14

#define Mtr1ActPos      M162    ; Suggested M-Variable Definition, Mtr 1 Actual Position

open plc 29 clear
if (OriginNotPassed = 1)
   cmd “#1j+“                   ; start moving toward the positive direction
   while (OriginNotPassed = 1)  ; until the index is detected
   endwhile
   cmd “#1k”
endif
while (Secondword & $8FF = 0)   ; there is a 2msec delay before inc. comp. is updated
endwhile
Mtr1ActPos = (int ((FirstWord & $8FFC00) / $400) – (SecondWord & $8FF) * $4 ) * i108 * 32
disable plc 29
close
```

Power-on phase referencing using Hall sensors

The Hall sensor data comes back on bits 1, 2 and 3 of the first word. This data can be used in order to establish an estimated phase reference for the motor on power-up. However, hall phasing will have ±30° error, which can result in loss of up to 14 percent of the torque output, but usually this is good enough for moving the motor until a more accurate reference is established (homing) and phase position data is updated accordingly.

Here is an example of how to determine the power-on phasing based on hall data. The following procedure is only required once. After determining the phase reference value, a power-on PLC would be sufficient to establish the phase reference and motor will be ready for commutation. Once the motor is moving and a better position reference can be established (usually homing sequence), the phase position can be fine tuned.

1. Tune the current loop on the motor (after setting correct values for Ixx00, Ixx01, Ixx24, Ixx82, Ixx84, Ixx57, Ixx58, Ixx69 and Ixx66, use the tuning software and tune the current loop i.e. Ixx61, Ixx62 and Ixx76).
2. Set the Ixx70 and Ixx71 based upon the type of encode you have. (for a 13-bit incremental encoder, Ixx71 = 8192 and Ixx70 = [number of pole pairs] and Ixx83 would be pointing to the correct Encoder Conversion Table Entry).
3. Set a positive value (usually 10 percent of Ixx66) to Ixx79 and set Ixx29=0 and Issue an O0 command (open loop, zero output).
4. Read the hall sensor data. This data can be read for the first encoder at Y:$78B20,0,4. We are interested only in bits 1 through 3 so the value should be shifted right by one bit or simply divide it by 2. This will be a number between 1 to 6 (a value of 0 or 7 is not valid).
5. Set Mxx71=0.
6. Set the Ixx79 back to its original value and issue a kill.
7. The following PLC will set up the phase reference.

Note

The Phase###Pos definition will change based on the number that is read during step 4 of the setup procedure explained earlier.

Step 4 Value	Definitions

1	`#define Phase30Deg 1`
	`#define Phase90Deg 5`
	`#define Phase150Deg 4`
	`#define Phase210Deg 6`
	`#define Phase270Deg 2`
	`#define Phase330Deg 3`

2	`#define Phase30Deg 2`
	`#define Phase90Deg 3`
	`#define Phase150Deg 1`
	`#define Phase210Deg 5`
	`#define Phase270Deg 4`
	`#define Phase330Deg 6`

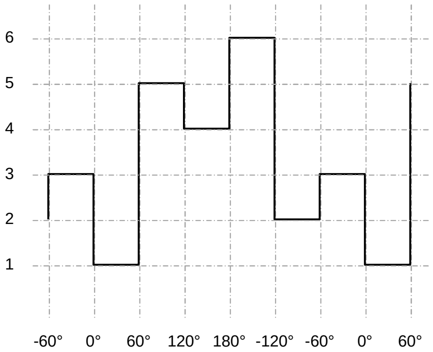
3	`#define Phase30Deg 3`
	`#define Phase90Deg 1`
	`#define Phase150Deg 5`
	`#define Phase210Deg 4`
	`#define Phase270Deg 6`
	`#define Phase330Deg 2`

Step 4 Value	Definitions

4	`#define Phase30Deg 4`
	`#define Phase90Deg 6`
	`#define Phase150Deg 2`
	`#define Phase210Deg 3`
	`#define Phase270Deg 1`
	`#define Phase330Deg 5`

5	`#define Phase30Deg 5`
	`#define Phase90Deg 4`
	`#define Phase150Deg 6`
	`#define Phase210Deg 2`
	`#define Phase270Deg 3`
	`#define Phase330Deg 1`

6	`#define Phase30Deg 6`
	`#define Phase90Deg 2`
	`#define Phase150Deg 3`
	`#define Phase210Deg 1`
	`#define Phase270Deg 5`
	`#define Phase330Deg 4`



```
#define FirstWord       M1000
#define Halls           M1002

FirstWord->Y:$78C00,0,24
Halls->*

#define Mtr1PhasePos      M171     ; Suggested M-Variable definition
#define Mtr1PhaseSrchErr  M148     ; Suggested M-Variable definition

#define Phase30Deg        1
#define Phase90Deg        5
#define Phase150Deg       4
#define Phase210Deg       6
#define Phase270Deg       2
#define Phase330Deg       3

open plc 29 clear
Halls = int ((FirstWord & $E) / 2);
If (Halls = Phase30Deg)
   Mtr1PhasePos = I171 * 30 / 360;
Endif
If (Halls = Phase90Deg)
   Mtr1PhasePos = I171 * 90 / 360;
Endif
If (Halls = Phase150Deg)
   Mtr1PhasePos = I171 * 150 / 360;
Endif
If (Halls = Phase210Deg)
   Mtr1PhasePos = I171 * 210 / 360;
Endif
If (Halls = Phase270Deg)
   Mtr1PhasePos = I171 * 270 / 360;
Endif
If (Halls = Phase330Deg)
   Mtr1PhasePos = I171 * 330 / 360;
Endif
Mtr1PhaseSrchErr = 0;
disable plc 28
close
```

BiSS-C Feedback Setup Example:

The following example demonstrates how to set up a 26-bit Resolute BiSS-C encoder for position control of a brushless motor on the first channel of an ACC-84E. Assume that the documentation for the encoder suggests 2MHz clock for the length of the cable that we have in the system:
Channel is reading a 26-bit BiSS-C Encoder. Note that the full 26-bit encoder data is used for absolute power-on position but the commutation/on-going position is limited to 24 bits by the Encoder Conversion Table (ECT).

X:$78C0F

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M Divisor								N Divisor				Reserved		Trigger Clock	Trigger Edge	Trigger Delay				Protocol Code			
0	1	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1
6				3				0				0				0				B			

```
WX:$78C0F,$63000B              ; Global Control register, 2 MHz Clock setting
```

Channel 1:X:$78B20

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRC_Mask								Reserved		Trigger Mode	Trigger Enable	Reseved	RxData Ready/ SENC	Reseved	StatusBits			Position Bits					
0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	1	0	1	0
2				1				1				4				9				A			

```
WX:$78C00,$21149A              ; Ch1 Control register, 37-Bit EnDat Encoder
```

Assigning values to the control registers should be performed upon power-up/reset in the initialization PLC.

```
Open plc 1 Clear
Disable plc 2..31
cmd"wx:$78C0F,$63000B"    ; Global Control register, 1 MHz Clock setting
cmd"wx:$78C00,$21149A"    ; Channel 1, read 26 bits
Disable plc 1
Enable plc 2..31
Close
```

Encoder conversion table setup required for BiSS-C encoder connected to the first channel on ACC-84E at base address set to $78C00 will be as follows:

```
I8000=$2F8B20                  ; Unfiltered parallel position of location
                        ; Y:$78B20, no shifting
I8001=$18000                   ; 24-bit processed result at $3502
I8002=$2F8B20                  ; Unfiltered parallel position of location
                        ; Y:$78B20, no shifting
I8003=$17003                   ; 23-bit read starting at bit 2,processed result at $3504
                               ; for commutation
I103=$3502                     ; position loop feedback address
I104=$3502                     ; velocity loop feedback address
```

Usually the number of counts in BiSS-C encoders are much higher than normal incremental encoders, the default settings for position and velocity feedback scale factors (a value of 96) can cause resolution restrictions on Servo gain settings. It is recommended that the scale factors be set to a smaller value.

```
I108=1          ; Motor1 position scale factor required not to saturate the Velocity
I109=1          ; Motor1 velocity-loop scale factor
```

Also, notice that the entries in ECT are not shifting the data. This means LSB of encoder data is 1/32 of a count as shown in position window. This can be crucial for preventing velocity limitations due to overflowing velocity registers in PMAC. The maximum velocity acceptable by PMAC is ($3\times(2^{18}-1)$) or 786431 counts per millisecond. Notice that this can be achieved very easily with a high resolution encoder. For example, a 26-bit encoder, if the data is shifted so LSB represents a count for PMAC, then only a maximum velocity of 700 RPM can be achieved. However, if the LSB of position data is used as 1/32 of a count, the maximum speed increases to 22,400 RPM.

$$\frac{3.\left(2^{18}-1\right)\text{ counts per msec}}{2^{26}\text{ counts per rev}} = 0.011718705\text{ rev per msec} = 703.12\text{ RPM}$$

Commutation with High Resolution Encoders (more than 23 bits per revolution)

The commutation in PMAC is based upon settings of Ixx70 and Ixx71 which define the number of pole pairs per revolution and the number of counts per revolution. Although the range for Ixx70 is not an issue with actual motors, the range of Ixx71 (24 bit value) can be a limitation when used with high resolution encoders.

The maximum value which can be assigned to Ix71 is a value of 16777215 or ($2^{24}-1$) meaning if the encoder generates more than 16777215 counts per revolution, we would have a problem setting the Ix71. In order to overcome this problem, a second entry in encoder conversion table can be utilized. In this entry, instead of reading the LSB of the position data, the upper 23 bits of data will be read. For example, in a 26 bit encoder, the second encoder conversion table entry would be set as follows:

```
I8002=$2F8B20                ; Unfiltered parallel position of location
                     ; Y:$78B20, no shifting
I8003=$17003                 ; 23-bit read starting at bit 3; Processed result at $3504
                             ; for commutation
I183=$3504                   ; on-going phase position
```

These settings will cause the ECT to read the upper 23 bits of position information starting at bit 3 (23+3=26 bits). Although the encoder generates 2^{26} counts per revolution, the output of ECT for this entry will only pass the upper 23 bits of data for use in commutation of the motor. The following table shows a few suggestions depending on the position bits of different encoders.

Encoder Resolution	2nd line
26 bit	$17003
32 bit	$17009

So the commutation parameters will be set as follows:

```
I171=8388608                 ; 23 bit data position per revolution
I170=2                       ; 2 pole pair motor
```

Absolute Power-On Servo Position

Since the BiSS-C protocol can provide absolute position, home search moves become redundant. Although the ECT entry will read the on-going position, it is only looking at lower 24 bits of position data and if the encoder position has more than 24-bits resolution, the higher bits are being neglected. In this case a power-on sequence should read all the bits and assigns them to actual position of the motor. There are two approaches for performing this task depending on ECT setup and whether the data is shifted or not.

Shifted Position Data

For shifted data, the approach is very simple and it uses the internal Ixx10 and Ixx95 settings of the Turbo PMAC in order to read all the position bits and assign them to actual position. The required values for Ixx10 depend on the base address of the card and channel number as shown in the table below:

Base Address	Ixx10 For 1st channel	Ixx10 For 2nd channel	Ixx10 For 3rd channel	Ixx10 For 4th channel
$78C00	$78C00	$78C04	$78C08	$78C0C
$79C00	$79C00	$79C04	$79C08	$79C0C
$7AC00	$7AC00	$7AC04	$7AC08	$7AC0C
$7BC00	$7BC00	$7BC04	$7BC08	$7BC0C
$78D00	$78D00	$78D04	$78D08	$78D0C
$79D00	$79D00	$79D04	$79D08	$79D0C
$7AD00	$7AD00	$7AD04	$7AD08	$7AD0C
$7BD00	$7BD00	$7BD04	$7BD08	$7BD0C
$78E00	$78E00	$78E04	$78E08	$78E0C
$79E00	$79E00	$79E04	$79E08	$79E0C
$7AE00	$7AE00	$7AE04	$7AE08	$7AE0C
$7BE00	$7BE00	$7BE04	$7BE08	$7BE0C

Setting of Ixx10 causes the PMAC to read the data in the address location as servo position upon execution of **$*** or **$**** command or upon power-on/reset if bit 2 of Ixx80 is set to 1. However, Ixx95 needs to be setup in order to identify how to read the position data from register defined by Ixx10. Since the data is in parallel format and in Y-memory, bits 16 to 21 of Ixx95 defines the length of the data. For example:

Position Bits	Ixx95 Setting
18	$120000
26	$1A000
32	$200000

No-Shift Position Data

If the position data is not shifted in ECT (which is usually done in order to prevent any velocity limitations), the LSB of position data reported by encoder equals to 1/32 of a count motor position. PMAC's built-in power-on servo position registers (Ixx10 and Ixx95) cannot be used in this case since these registers expect the LSB to have a value of 1 count and instead a PLC should read the encoder registers and write the correct position data to actual position of the motor.

Here is an example on how to read the position data from ACC-84E registers and assign them to motor actual position in a PLC.

```
CLOSE
DEL GAT

#define Chn1RegA       M2000
#define Chn1RegB       M2001
```

```
Chn1RegA->Y:$78C00,0,24            ; 1st 24-bits of position data
Chn1RegB->Y:$78C01,0,16            ; overflow of the bits
#define Mtr1ActPos     M162        ; Suggested M-variable definition

OPEN PLC 10 CLEAR
Mtr1ActPos = (Chn1RegB * $1000000 + Chn1RegA) * I108
DISABLE PLC 10
CLOSE
```

Absolute Power-On/Reset Phase Position

By knowing the difference between the absolute encoder position and the commutation cycle zero (stored in Ixx75 in PMAC), a phase search routine is no longer necessary on power-up/reset. In order to have a power-on/reset phasing based upon the absolute encoder, Ixx81, Ixx91 and Ixx75 needs to be set.
Motor power-on phase position address (Ixx81) should point to the same address used for motor commutation position address (Ixx83) which is processed data from ECT with lower resolution.
Motor power-on phase position format (Ixx91) setting depends on your settings for commutation specific ECT entry.

Encoder Resolution	ECT 2nd line	Ixx91
18 bits	$12000	$520000
26 bits	$17003	$570000
32 bits	$17009	$570000

The following procedure explains that finding Ixx75 is done only once per channel while setting up the machine for the first time, assuming the mechanics and electronics are not to be changed and have not failed/been replaced or repaired:

1. Set Ixx79=500 and Ixx29=-500 (The sign of value assigned to Ixx79 should match the sign of Ixx70 and sign of value for Ixx29 is always opposite to Ixx79).
2. Increase these values by increments of 100 until motor is locked in to a position when O0 is issued. Acceptable range for Ixx79 and Ixx29 is 0 to Ixx57 (continuous current limit).
3. Issue a #nO0, wait for motor to stop moving.
4. Set Ixx29=0, wait for motor to stop moving.
5. Set Mxx71 to zero (see suggested M-variables).
6. Read position data from ECT X-word where Ixx81 and Ixx83 are pointing (use RX command. For example: RX:$3504).
7. Set Ixx75 to the negative of the value read in step 6 multiplied by Ixx70 modulo Ix71

$$\text{Ixx75}=\left(-\text{ Position Read While at Zero Phase } \times \text{ Ixx70}\right)\%\text{ Ixx71}$$

8. Set Ixx79=0.
9. Issue a #nK to kill the motor.

The following examples are for typical encoder resolutions available on BiSS-C protocol:

```
// Renishaw Resolute Rotary Encoder - 18 Bit

wx:$78C0F,$63000B
wx:$78C00,$211492

I8000=$278C00
I8001=$012000
I8002=$2F8C00
I8003=$012000
I8004=$0

I103=$3502
I104=$3502
I183=$3504

i108=1
```

```
i109=1

I171=262144
I170=2         // motor specific

I180=0
I181=$3504
I191=$520000

I110=$78C00
I195=$120000

// Other I-variables which needs to be set before motor can be used
// in order of setup
// I100, I101, I102, I124, I125, I166, I182, I84, I172, I7mn6
// Tune current loop I161, I162, I176
// I175
// Tune servo loop I130, I131, I132, I133, I134, I135
```

```
// Renishaw Resolute Rotary Encoder – 26 Bit

wx:$78C0F,$63000B
wx:$78C00,$21149A

I8000=$2F8C00
I8001=$018000
I8002=$2F8C00
I8003=$017003
I8004=$0

I103=$3502
I104=$3502
I183=$3504

i108=1
i109=1

I171=8388608
I170=2         // motor specific

I180=0
I181=$3504
I191=$570000

CLOSE
DEL GAT

#define Chn1RegA       M2000
#define Chn1RegB       M2001
Chn1RegA->Y:$78C00,0,24              ; 1st 24-bits of position data
Chn1RegB->Y:$78C01,0,16              ; overflow of the bits
#define Mtr1ActPos     M162          ; Suggested M-variable definition

OPEN PLC 10 CLEAR
Mtr1ActPos = (Chn1RegB * $1000000 + Chn1RegA) * I108
DISABLE PLC 10
CLOSE

// Other I-variables which needs to be set before motor can be used
// in order of setup
// I100, I101, I102, I124, I125, I166, I182, I84, I172, I7mn6
// Tune current loop I161, I162, I176
// I175
// Tune servo loop I130, I131, I132, I133, I134, I135
```

```
// Renishaw Resolute Rotary Encoder – 32 Bit

wx:$78C0F,$63000B
wx:$78C00,$211420
```

```
I8000=$2F8C00
I8001=$018000
I8002=$2F8C00
I8003=$017009
I8004=$0

I103=$3502
I104=$3502
I183=$3504

i108=1
i109=1

I171=8388608
I170=2         // motor specific

I180=0
I181=$3504
I191=$570000

CLOSE
DEL GAT

#define Chn1RegA      M2000
#define Chn1RegB      M2001
Chn1RegA->Y:$78C00,0,24               ; 1st 24-bits of position data
Chn1RegB->Y:$78C01,0,16               ; overflow of the bits
#define Mtr1ActPos    M162            ; Suggested M-variable definition

OPEN PLC 10 CLEAR
Mtr1ActPos = (Chn1RegB * $1000000 + Chn1RegA) * I108
DISABLE PLC 10
CLOSE


// Other I-variables which needs to be set before motor can be used
// in order of setup
// I100, I101, I102, I124, I125, I166, I182, I84, I172, I7mn6
// Tune current loop I161, I162, I176
// I175
// Tune servo loop I130, I131, I132, I133, I134, I135
```

APPENDIX B: SERIAL LINK (XY2-100) PROTOCOL SUPPORT

The XY2-100 Serial Link (also known as Serial Link 1 and XYZ-100) is a synchronous TIA/EIA-422-B differential digital interface for communication of three 16-bit position words and a single 16-bit status word for two- and three-axis servo applications.

Delta Tau introduced support for this protocol in 1st quarter of 2015 on its ACC-84x FPGA Based Serial Encoder Interface platform. The implementation of XY2-100 is based upon XY2-100 Serial Link 1 Specification by General Scanning (GSI) and expanded to support 18 and 20 bit data formats.

This protocol is available on the following implementations of ACC-84x Series:

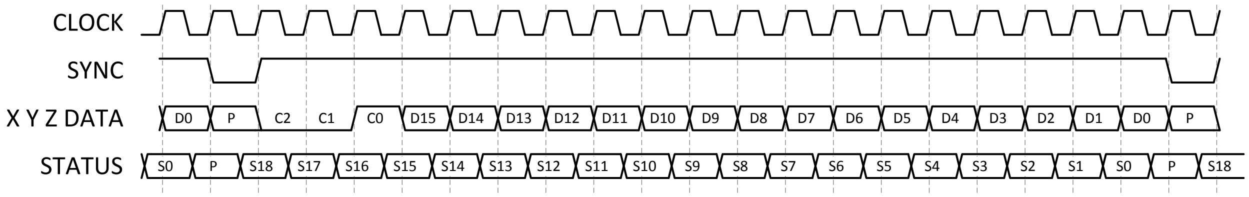
- ACC-84E (UMAC, both Turbo and Power PMAC)
- ACC-84C (Compact UMAC or CPCI, both Turbo and Power PMAC)
- ACC-84S (Turbo PMAC2A/PC104, Turbo Clipper and Power Clipper)
- ACC-84B (Power Brick)
- Auxiliary Input for Turbo Brick

Signal Description

Note

The information provided in this section is only for reference and to provide a better understanding of the implementation. The XY2-100 option of ACC-84x automatically generates proper signals and their timings defined by XY2-100 protocol.

The XY2-100 Serial Link (or XYZ-100) uses 5 (6 in case of XYZ-100) signals for communication between the trajectory generation engine (ACC-84x and PMAC) and the scanhead/galvanometer servo drive.



XY2-100 Timing Diagram

Clock

The Clock is transmitted by the position data generator (ACC-84x), 20 cycles per frame. Its nominal frequency is 2MHz. With 2MHz clock and 20 cycles per frame, the position data frame is updated every 10µs.

For most galvanometers, the standard internal controller update rate is 20µs, for smaller galvanometers the update rate is 10µs, for larger inertias a 40µs update rate may be needed.

ACC-84x implementation of XY2-100 provides capability for changing the clock frequency, which allows different update rates to scanheads/galvanometers.

Sync

The frame Sync is a single logical "0" pulse, once per frame, transmitted by the position data generator one clock cycle prior to the first bit of the frame.

X, Y & Z Data

In standard XY2-100 protocol, the X, Y, & Z Data are three 20-bit serial data streams consisting of a 3-bit control code, one 16-bit position word (unsigned, MSB first), and a parity bit (even parity).

ACC-84x implementation of XY2-100 provides capability of transmitting standard 16-bit position data as described by XY2-100 protocol. In addition, it can transmit 18-bit (compatible with 18-bit Serial Link 2) and 20-bit (compatible with Canon GM-1000) position data packets. In addition, the variant of the parity bit (odd or even) can be selected.

Status

The Status data is a 20-bit serial data stream consisting of a 3-bit control code, one 16-bit status word, and a parity bit (even parity) which is generated by the scanhead/galvanometer servo drive and read by ACC-84x.

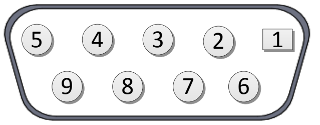
Connections

There are two groups of connections for ACC-84x, depending on the form factor: DE-9 pin connectors used on ACC-84E and ACC-84S and DA-15 connectors used in Brick family of products.

Unlike serial encoder interface implementations on ACC-84x where the connectors all have the same pin-outs and supports 4 channels of same encoder protocol, the XY2-100 combines all channels of the ACC-84x for interfacing with a galvanometer or scanhead. First 3 channel connects are used for interfacing to XY2-100 device. A differential PWM output with programmable period and duty cycle is provided on the last channel. This output can be used for control of the laser intensity.

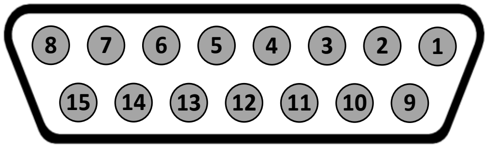
DE-9 Connector Pin Out

The DE-9 connector pin out is used on ACC-84E and ACC-84S.

D-Sub DE9 Female Mating: D-Sub DE9 Male				
Channel Pin #	Channel 1	Channel 2	Channel 3	Channel 4
1	STATUS–	SYNC–	CLOCK–	PWM–
2	CHX–	CHY–	CHZ–	N.C.
3	N.C.	N.C.	N.C.	N.C.
4	GND	GND	GND	GND
5	GND	GND	GND	GND
6	STATUS+	SYNC+	CLOCK +	PWM+
7	CHX+	CHY+	CHZ+	N.C.
8	N.C.	N.C.	N.C.	N.C.
9	+5VDC	+5VDC	+5VDC	+5VDC

DA-15 Connector Pin Out

The DA-15 connector pin out is used on Brick Family of products.

<table>
<tr><td colspan="2">D-Sub DA15 Female
Mating: D-Sub DA15 Male</td><td colspan="3"> </td></tr>
<tr><td>Channel
Pin #</td><td>Channel 1</td><td>Channel 2</td><td>Channel 3</td><td>Channel 4</td></tr>
<tr><td>1</td><td>A+/SIN+</td><td>A+/SIN+</td><td>A+/SIN+</td><td>A+/SIN+</td></tr>
<tr><td>2</td><td>B+/COS+</td><td>B+/COS+</td><td>B+/COS+</td><td>B+/COS+</td></tr>
<tr><td>3</td><td>INDEX+</td><td>INDEX+</td><td>INDEX+</td><td>INDEX+</td></tr>
<tr><td>4</td><td>+5VDC</td><td>+5VDC</td><td>+5VDC</td><td>+5VDC</td></tr>
<tr><td>5</td><td>CHX–</td><td>CHY–</td><td>CHZ–</td><td>N.C.</td></tr>
<tr><td>6</td><td>STATUS–</td><td>SYNC–</td><td>CLOCK–</td><td>PWM–</td></tr>
<tr><td>7</td><td>2.5V REF</td><td>2.5V REF</td><td>2.5V REF</td><td>2.5V REF</td></tr>
<tr><td>8</td><td>PTC</td><td>PTC</td><td>PTC</td><td>PTC</td></tr>
<tr><td>9</td><td>A–/SIN–</td><td>A–/SIN–</td><td>A–/SIN–</td><td>A–/SIN–</td></tr>
<tr><td>10</td><td>B–/COS–</td><td>B–/COS–</td><td>B–/COS–</td><td>B–/COS–</td></tr>
<tr><td>11</td><td>INDEX–</td><td>INDEX–</td><td>INDEX–</td><td>INDEX–</td></tr>
<tr><td>12</td><td>GND</td><td>GND</td><td>GND</td><td>GND</td></tr>
<tr><td>13</td><td>STATUS+</td><td>SYNC+</td><td>CLOCK +</td><td>PWM+</td></tr>
<tr><td>14</td><td>CHX+</td><td>CHY+</td><td>CHZ+</td><td>N.C.</td></tr>
<tr><td>15</td><td>RES_EXT</td><td>RES_EXT</td><td>RES_EXT</td><td>RES_EXT</td></tr>
</table>

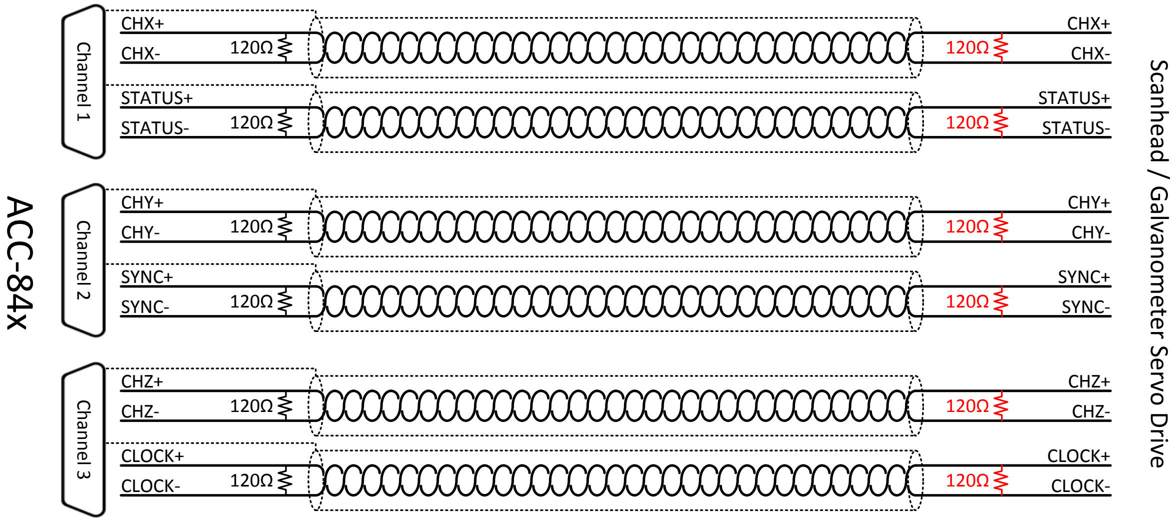
Note

Pins/signals indicated in light gray are available on the same X1-X8 connectors on the Brick, but they are not used in conjunction with XY2-100 protocol. Regardless of the state of XY2-100 (enable or disabled) these encoder input pins have their original functionality and can be setup/used as explained in the hardware reference manual for Brick product in question.

Signal Termination

All form factors of ACC-84x, incorporates differential line transceivers suitable for high speed bidirectional data communication. It is designed for balanced data transmission and complies with both RS-485 and RS-422 EIA Standards. The transmission line of choice for RS-485 communications is a twisted pair. Twisted pair cable tends to cancel common-mode noise and also causes cancellation of the magnetic fields generated by the current flowing through each wire, thereby reducing the effective inductance of the pair. As with any transmission line, it is important that reflections are minimized. This can be achieved by terminating the extreme ends of the line using resistors equal to the characteristic impedance of the line.

In general, termination schematic between ACC-84x and Scan-head/galvanometer servo drive as shown in the diagram below should be implemented. This document recommends use of double shielded twisted pair cable for XY2-100 link.



Note

Scanhead/Galvanometer servo drive manufacturers often incorporate termination resistors on their input. Please consult with the manufacturer data to identify if implementation of external termination resistor at the drop off point is necessary.

Note

All termination resistors at the ACC-84x is necessary for transmission of XY2-100 protocol and compliance with RS-485 termination requirements.

Setup Elements

Note

This document uses ACC-84E[*i*] in all mentions are Power PMAC registers. In case of other form factors, users should use ACC84C[*i*], ACC84B[*i*] and ACC84S[*i*] instead.

Multi-Channel Setup Element

The multi-channel setup element **Acc84E[*i*].SerialEncCtrl** (saved element in Power PMAC only and non-saved in Turbo PMAC, must be setup in power/initialization PLC) specifies several aspects of the XY2-100 configuration: trigger enable, the interpolation clock selection and the clock frequency.
The different components of this 24-bit full-word element cannot be accessed as independent elements, so it is necessary to assemble the full-word value from the values of the individual components. It is easiest to treat the value as a hexadecimal value, so the individual components can be seen independently.

Power PMAC Global Control Register	**Turbo PMAC Global Control Register**	**Switch Position (SW1)**			
		1	**2**	**3**	**4**
ACC84E[0].SerialEncCtrl	X:$78C0F	Close	Close	Close	Close
ACC84E[4].SerialEncCtrl	X:$79C0F	Close	Close	Open	Close
ACC84E[8].SerialEncCtrl	X:$7AC0F	Close	Close	Close	Open
ACC84E[12].SerialEncCtrl	X:$7BC0F	Close	Close	Open	Open
ACC84E[1].SerialEncCtrl	X:$78D0F	Open	Close	Close	Close
ACC84E[5].SerialEncCtrl	X:$79D0F	Open	Close	Open	Close
ACC84E[9].SerialEncCtrl	X:$7AD0F	Open	Close	Close	Open
ACC84E[13].SerialEncCtrl	X:$7BD0F	Open	Close	Open	Open
ACC84E[2].SerialEncCtrl	X:$78E0F	Close	Open	Close	Close
ACC84E[6].SerialEncCtrl	X:$79E0F	Close	Open	Open	Close
ACC84E[10].SerialEncCtrl	X:$7AE0F	Close	Open	Close	Open
ACC84E[14].SerialEncCtrl	X:$7BE0F	Close	Open	Open	Open

Acc84E[*i*].SerialEncCtrl is the full-word element that comprises the multi-channel setup for serial encoder interfaces for the ACC-84E. It is comprised of the following components (which cannot be accessed as independent elements):

Component	Turbo PMAC/ Power PMAC Script Bits	Hex Digit #	C Bits	Functionality
ClockMDiv	23 – 16	1 – 2	31 – 24	Clock linear division factor
ClockNDiv	15 – 12	3	23 – 20	Clock exponent division factor
TxEnable	11	4	19	Enables transfer of XY2-100 data
Parity	10	4	18	Selection of parity bit variant
ClockSel	09	4	17	Interpolation clock source select
-	08	4	16	*Reserved*
ModeSel	07 – 06	5	15 – 14	Transmitted position data resolution select
-	05 – 04	5	13 – 12	*Reserved*
-	03 – 00	6	11 – 08	*Reserved*

The components *ClockMDiv* and *ClockNDiv* control how the XY2-100 clock frequency is generated from the IC's fixed 100 MHz clock frequency. The equation for this clock frequency f_{XY2} is:

$$f_{XY2}(MHz) = \frac{100}{(M+1)*2^N}$$

where *M* is short for *ClockMDiv*. This 8-bit component can take a value from 0 to 255. *N* is short for *ClockNDiv*. This 4-bit component can take a value from 0 to 15, so the resulting 2^N divisor can take a value from 1 to 32,768. Table below includes the most common settings for *M* and *N* dividers.

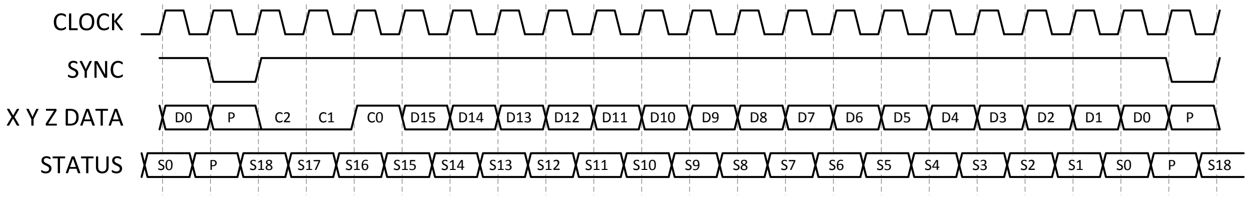
SerialClockMDiv	*SerialClockNDiv*	Clock Freq (MHz)	Update Period (μsec)	Position Frame Update Freq (kHz)
24 ($18)	1 ($1)	2.0 MHz	10	100
24 ($18)	2 ($2)	1.0 MHz	20	50
24 ($18)	3 ($3)	0.5 MHz	40	25

The component *TxEnable* controls whether the XY2-100 position data is being transferred to the scanhead/galvanometer. Setting this bit to 1 enables the driver circuitry for the XY2-100 Clock, Sync and Data lines. This bit must be set to 1 to command any scanheads/galvanometers. If there is an alternate use for the same signal pins, this bit must be set to 0 so the drivers do not conflict with the alternate use.

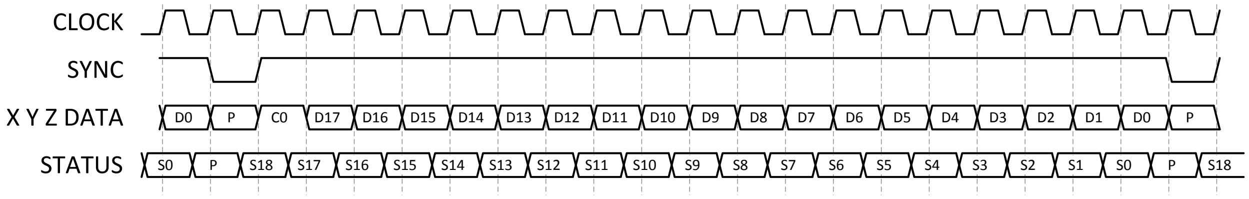
The component *ClockSel* controls which Power PMAC clock signal is used for capturing the position data generated by Power PMAC and perform linear interpolation between the received commands. If motor trajectory is calculated in servo loop, standard for Power PMAC and only choice in Turbo PMAC, then the *ClockSel* should be set to 0 to select Servo Loop. In Power PMAC, if bit 3 (value 8) of **Motor[*x*].PhaseCtrl** is set to 1, this motor will close its position/velocity servo loop on the phase interrupt. This permits some Power PMAC motors, such as those driving "fast-tool servos" or galvanometers, to close their loops at a substantially higher frequency than other motors in the system. In this case *ClockSel* should be set to 1 to allow interpolation between position updates at Phase rate.

The component *ModeSel* specifies the X, Y and Z position packet format and position resolution. The *ModeSel* allows users to select between 16-bit, 18-bit and 20-bit resolution. The following timing diagrams show the data format of each *ModeSel* setting:

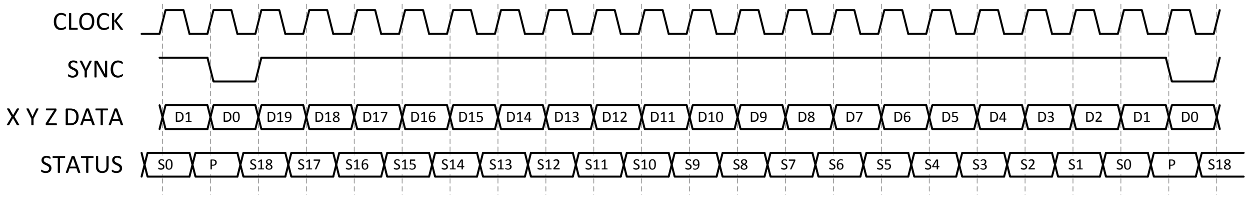
ModeSel = 00
16-bit data format (XY2-100 Standard, Serial Link)


CLOCK
SYNC
X Y Z DATA
D0 P C2 C1 C0 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 P
STATUS
S0 P S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0 P S18

ModeSel = 01
18-bit data format (Serial Link 2)


CLOCK
SYNC
X Y Z DATA
D0 P C0 D17 D16 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 P
STATUS
S0 P S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0 P S18

ModeSel = 10
20-bit data format


CLOCK
SYNC
X Y Z DATA
D1 D0 D19 D18 D17 D16 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
STATUS
S0 P S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0 P S18

ModeSel = 11 (*Reserved for future*)

The following list shows typical settings of **Acc84E[*i*].SerialEncCtrl** for a 2MHz transfer clock.

SerialClockMDiv:	= $18	// Serial clock frequency = bit transmission frequency
SerialClockNDiv:	= 1	
TxEnable:	= 1	// Enable transmission of XY2-100 position data
Parity:	= 1	// Odd parity selection
ClockSel:	= 0	// Servo clock selection for interpolation
ModeSel:	= $0	// XY2-100 Serial Link Standard (16-bit) position data

For example, for a 2 MHz bit transmission rate, ***SerialClockMDiv*** = 24 ($18) and ***SerialClockNDiv*** = 1 ($1) **Acc84E[*i*].SerialEncCtrl** is set to $181C00 for interpolation based upon Servo clock and odd parity calculation.

Hex Digit ($)	1				8				1				C				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*TE*	*P*	*CS*	--	*ModeSel*									

Channel Specific Command Register

Each channel of the FPGA has a 24-bit saved setup element **Acc84E[*i*].Chan[*j*].SerialEncCmd** (saved element in Power PMAC only and non-saved in Turbo PMAC).

Power PMAC Channel Control Register	**Turbo PAMC Base Address**	**Channel**			
		1 (j=0)	**2 (j=1)**	**3 (j=2)**	**4 (j=3)**
ACC84E[0].Chan[j].SerialEncCmd	$78C00	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
ACC84E[4].Chan[j].SerialEncCmd	$79C00	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
ACC84E[8].Chan[j].SerialEncCmd	$7AC00	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
ACC84E[12].Chan[j].SerialEncCmd	$7BC00	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
ACC84E[1].Chan[j].SerialEncCmd	$78D00	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
ACC84E[5].Chan[j].SerialEncCmd	$79D00	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
ACC84E[9].Chan[j].SerialEncCmd	$7AD00	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
ACC84E[13].Chan[j].SerialEncCmd	$7BD00	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
ACC84E[2].Chan[j].SerialEncCmd	$78E00	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
ACC84E[6].Chan[j].SerialEncCmd	$79E00	Y:$79E00	Y:$79E04	Y:$79E08	Y:$79E0C
ACC84E[10].Chan[j].SerialEncCmd	$7AE00	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
ACC84E[14].Chan[j].SerialEncCmd	$7BE00	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C

Position Command Registers

The FPGA always reads full 24-bit registers **Acc84E[*i*].Chan[*j*].SerialEncCmd**, for j equal to 0, 1 and 2, as signed 24-bit position command for axis X, Y and Z on the rising edge of selected interpolation clock, set by *ClockSel* in **Acc84E[*i*].SerialEncCtrl**. After reading the update commanded position, firmware applies a linear interpolation between commanded positions at the frame rate of the XY2-100 update rate set by components *ClockMDiv* and *ClockNDiv* of **Acc84E[*i*].SerialEncCtrl**.

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	-	-
Component:	*Commanded Position*																									

After interpolation at frame rate of the XY2-100, depending on the *ModeSel* component of **Acc84E[*i*].SerialEncCtrl**, upper 16-bit, 18-bit or 20-bits of the commanded position data is transmitted to the scanhead/galvanometer servo drive.

Note

Unlike PMAC convention, in which the position is usually calculated as a signed integer (signed floating point in case of Power PMAC), XY2-100 protocol defines it as an unsigned integer. In order to simplify this conversion, Delta Tau's implementation of XY2-100 on ACC-84x products, deals with this conversion of signed position commands to unsigned values as defined by XY2-100 standard.

ModeSel = 00

16-bit data format (XY2-100 Standard, Serial Link)

Hex Digit ($)																									-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	S0	S1	S2	S3	S4	S5	S6	S7	-	-
Component:	*16-bit Commanded Position*																*8-bit sub-count*									

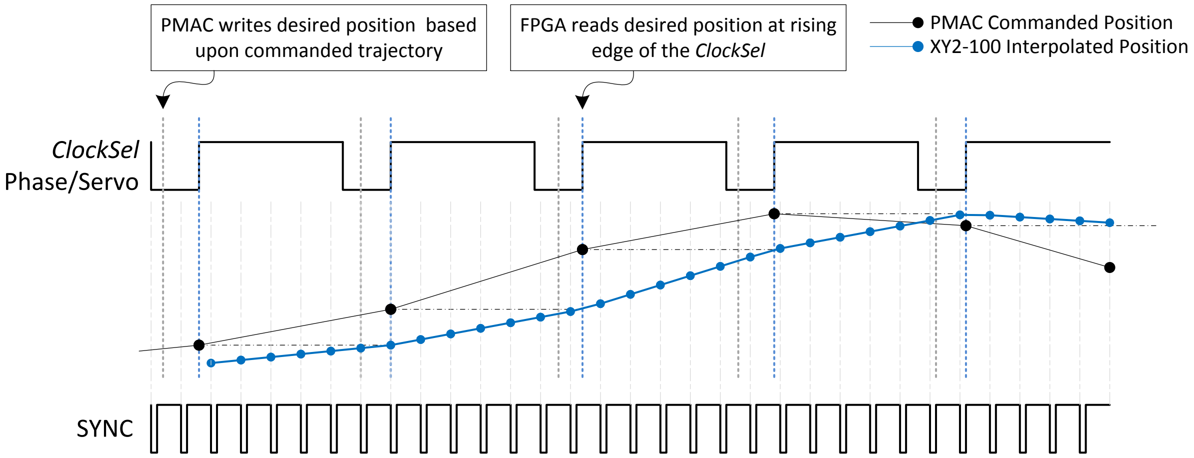
ModeSel = 01
18-bit data format (Serial Link 2)

Hex Digit ($)	0				0				5				C				1				9				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	S0	S1	S2	S3	S4	S5	-	-
Component:	*18-bit Commanded Position*																		*6-bit sub-count*							

ModeSel = 10
20-bit data format

Hex Digit ($)	0				0				5				C				1				9				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	S0	S1	S2	S3	-	-
Component:	*20-bit Commanded Position*																				*4-bit sub-count*					

ModeSel = 11 (*Reserved for future*)


PMAC writes desired position based upon commanded trajectory
FPGA reads desired position at rising edge of the *ClockSel*
PMAC Commanded Position
XY2-100 Interpolated Position
ClockSel
Phase/Servo
SYNC

Separation of frequency domains between Phase/Servo clock and XY2-100 and linear Interpolation of commanded position

PWM Command Register

The FPGA always reads full 24-bit registers **Acc84E[*i*].Chan[*j*].SerialEncCmd**, for the last channel with j=3, and it is used for adjusting the PWM output frequency and duty cycle. This PWM output is not a part of XY2-100 protocol, but it is an added feature to ACC-84x with XY2-100 option. This output is intended for use with Laser sources in order to control laser's power. It is comprised of the following components:

Component	Turbo PMAC/ Power PMAC Script Bits	Hex Digit #	C Bits	Functionality
DutyCycle	23 – 12	1- 3	31 – 20	Positive duty cycle of output
PwmPeriod	11 – 00	3-6	19 – 08	PWM period of the output

The component *PwmPeriod* controls the period duration of the PWM output and it is inversely proportional to the frequency of the signal. The equation for this PWM output frequency f_{PWM} is:

$$f_{PWM}(kHz) = \frac{10^5}{16 \times P}$$

where *P* is short for *PwmPeriod*. This 12-bit component can take a value from 1 to 4095 and generates PWM frequencies ranging from 1526 Hz to 6.25 MHz.

Here are some examples for *PwmPeriod* settings:

PwmPeriod	PWM Freq (kHz)
3125 ($C35)	2
1250 ($4E2)	5
625 ($271)	10
312 ($138)	20
125 ($07D)	50

The component *DutyCycle* controls the positive duty cycle of the PWM+ output signal available on 4th channel of the ACC-84x and it is independent of the PWM frequency set by *PwmPeriod* component. The equation for this PWM duty cycle is:

$$\text{Positive Duty Cycle}\,(\%) = \frac{D}{4096} \times 100$$

where *D* is short for *DutyCycle*. This 12-bit component can take a value from 0 to 4095 and generates duty cycles ranging from 0% to 99.97%.

The following list shows typical settings of **Acc84E[*i*].Chan[3].SerialEncCmd** for a 5kHz PWM output at 25% duty cycle.

DutyCycle:	= $400	// $400=1024 setting for 25% duty cycle command (1024/4096)
PwmPeiod:	= $4E2	// $4E2(1250) setting for PwmPeriod generates a 5kHz PWM freq.

Hex Digit ($)	4				0				0				4				E				2				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	-	-
Component:	*Duty Cycle*												*PwmPeriod*													

Status Data Structures

Status elements of ACC-84E are read only elements where the received data and status flags are written by FPGA at every frame cycle.

Single-Channel Status Elements

Some aspects of the XY2-100 protocol, such as interpolated position data and scanhead/servo drive provided status bits can be read individually for each channel. Each channel of the FPGA has four 24-bit status elements:

- **Acc84E[*i*].Chan[*j*].SerialEncDataA**
- **Acc84E[*i*].Chan[*j*].SerialEncDataB**
- **Acc84E[*i*].Chan[*j*].SerialEncDataC**
- **Acc84E[*i*].Chan[*j*].SerialEncDataD**

In XY2-100 protocol, only **Acc84E[*i*].Chan[*j*].SerialEncDataA** register is used and all other status registers are set to 0.

POWER	**TURBO**	**Power PMAC ACC-84E Data Register**	**Channel**			
			1 (j=0)	**2 (j=1)**	**3 (j=2)**	**4 (j=3)**
ACC84E[0]	$78C00	Chan[j].SerialEncDataA	Y:$78C00	Y:$78C04	Y:$78C08	Y:$78C0C
		Chan[j].SerialEncDataB	Y:$78C01	Y:$78C05	Y:$78C09	Y:$78C0D
		Chan[j].SerialEncDataC	Y:$78C02	Y:$78C06	Y:$78C0A	Y:$78C0E
		Chan[j].SerialEncDataD	Y:$78C03	Y:$78C07	Y:$78C0B	Y:$78C0F
ACC84E[4]	$79C00	Chan[j].SerialEncDataA	Y:$79C00	Y:$79C04	Y:$79C08	Y:$79C0C
		Chan[j].SerialEncDataB	Y:$79C01	Y:$79C05	Y:$79C09	Y:$79C0D
		Chan[j].SerialEncDataC	Y:$79C02	Y:$79C06	Y:$79C0A	Y:$79C0E
		Chan[j].SerialEncDataD	Y:$79C03	Y:$79C07	Y:$79C0B	Y:$79C0F
ACC84E[8]	$7AC00	Chan[j].SerialEncDataA	Y:$7AC00	Y:$7AC04	Y:$7AC08	Y:$7AC0C
		Chan[j].SerialEncDataB	Y:$7AC01	Y:$7AC05	Y:$7AC09	Y:$7AC0D
		Chan[j].SerialEncDataC	Y:$7AC02	Y:$7AC06	Y:$7AC0A	Y:$7AC0E
		Chan[j].SerialEncDataD	Y:$7AC03	Y:$7AC07	Y:$7AC0B	Y:$7AC0F
ACC84E[12]	$7BC00	Chan[j].SerialEncDataA	Y:$7BC00	Y:$7BC04	Y:$7BC08	Y:$7BC0C
		Chan[j].SerialEncDataB	Y:$7BC01	Y:$7BC05	Y:$7BC09	Y:$7BC0D
		Chan[j].SerialEncDataC	Y:$7BC02	Y:$7BC06	Y:$7BC0A	Y:$7BC0E
		Chan[j].SerialEncDataD	Y:$7BC03	Y:$7BC07	Y:$7BC0B	Y:$7BC0F
ACC84E[1]	$78D00	Chan[j].SerialEncDataA	Y:$78D00	Y:$78D04	Y:$78D08	Y:$78D0C
		Chan[j].SerialEncDataB	Y:$78D01	Y:$78D05	Y:$78D09	Y:$78D0D
		Chan[j].SerialEncDataC	Y:$78D02	Y:$78D06	Y:$78D0A	Y:$78D0E
		Chan[j].SerialEncDataD	Y:$78D03	Y:$78D07	Y:$78D0B	Y:$78D0F
ACC84E[5]	$79D00	Chan[j].SerialEncDataA	Y:$79D00	Y:$79D04	Y:$79D08	Y:$79D0C
		Chan[j].SerialEncDataB	Y:$79D01	Y:$79D05	Y:$79D09	Y:$79D0D
		Chan[j].SerialEncDataC	Y:$79D02	Y:$79D06	Y:$79D0A	Y:$79D0E
		Chan[j].SerialEncDataD	Y:$79D03	Y:$79D07	Y:$79D0B	Y:$79D0F
ACC84E[9]	$7AD00	Chan[j].SerialEncDataA	Y:$7AD00	Y:$7AD04	Y:$7AD08	Y:$7AD0C
		Chan[j].SerialEncDataB	Y:$7AD01	Y:$7AD05	Y:$7AD09	Y:$7AD0D
		Chan[j].SerialEncDataC	Y:$7AD02	Y:$7AD06	Y:$7AD0A	Y:$7AD0E
		Chan[j].SerialEncDataD	Y:$7AD03	Y:$7AD07	Y:$7AD0B	Y:$7AD0F
ACC84E[13]	$7BD00	Chan[j].SerialEncDataA	Y:$7BD00	Y:$7BD04	Y:$7BD08	Y:$7BD0C
		Chan[j].SerialEncDataB	Y:$7BD01	Y:$7BD05	Y:$7BD09	Y:$7BD0D
		Chan[j].SerialEncDataC	Y:$7BD02	Y:$7BD06	Y:$7BD0A	Y:$7BD0E
		Chan[j].SerialEncDataD	Y:$7BD03	Y:$7BD07	Y:$7BD0B	Y:$7BD0F
ACC84E[2]	$78E00	Chan[j].SerialEncDataA	Y:$78E00	Y:$78E04	Y:$78E08	Y:$78E0C
		Chan[j].SerialEncDataB	Y:$78E01	Y:$78E05	Y:$78E08	Y:$78E0D
		Chan[j].SerialEncDataC	Y:$78E02	Y:$78E06	Y:$78E09	Y:$78E0E
		Chan[j].SerialEncDataD	Y:$78E03	Y:$78E07	Y:$78E0A	Y:$78E0F
ACC84E[6]	$79E00	Chan[j].SerialEncDataA	Y:$79E00	Y:$79E04	Y:$79E0B	Y:$79E0C
		Chan[j].SerialEncDataB	Y:$79E01	Y:$79E05	Y:$79E0C	Y:$79E0D
		Chan[j].SerialEncDataC	Y:$79E02	Y:$79E06	Y:$79E0D	Y:$79E0E
		Chan[j].SerialEncDataD	Y:$79E03	Y:$79E07	Y:$79E0E	Y:$79E0F
ACC84E[10]	$7AE00	Chan[j].SerialEncDataA	Y:$7AE00	Y:$7AE04	Y:$7AE08	Y:$7AE0C
		Chan[j].SerialEncDataB	Y:$7AE01	Y:$7AE05	Y:$7AE09	Y:$7AE0D
		Chan[j].SerialEncDataC	Y:$7AE02	Y:$7AE06	Y:$7AE0A	Y:$7AE0E
		Chan[j].SerialEncDataD	Y:$7AE03	Y:$7AE07	Y:$7AE0B	Y:$7AE0F
ACC84E[14]	$7BE00	Chan[j].SerialEncDataA	Y:$7BE00	Y:$7BE04	Y:$7BE08	Y:$7BE0C
		Chan[j].SerialEncDataB	Y:$7BE01	Y:$7BE05	Y:$7BE09	Y:$7BE0D
		Chan[j].SerialEncDataC	Y:$7BE02	Y:$7BE06	Y:$7BE0A	Y:$7BE0E
		Chan[j].SerialEncDataD	Y:$7BE03	Y:$7BE07	Y:$7BE0B	Y:$7BE0F

The FPGA always writes full 24-bit registers **Acc84E[*i*].Chan[*j*].SerialEncDataA**, for j equal to 0, 1 and 2, as signed 24-bit interpolated position command for axis X, Y and Z on the falling edge of interpolation clock (SYNC). This data is purely for use as simulated feedback for PMAC motors as the position loop is closed in scanhead/galvanometer drive and only access to status bits about the following error is available under XY2-100 protocol.

The interpolated position is always written as a 24-bit signed integer. Depending on the *ModeSel* setting, user should scale the position values reported in this register to represent 16-bit, 18-bit or 20-bit whole count position data. Although the XY2-100 only receives the upper 16-bit, 18-bit or 20-bits of position from the 24-bit interpolated values, the sub-counts are essential for proper interpolation and avoiding round off errors.

Acc84E[*i*].Chan[0].SerialEncDataA returns interpolated X Data

Hex Digit ($)																								-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	-	-
Component:	*Interpolated Commanded X Position*																								

Acc84E[*i*].Chan[1].SerialEncDataA returns interpolated Y Data

Hex Digit ($)																								-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	-	-
Component:	*Interpolated Commanded Y Position*																								

Acc84E[*i*].Chan[2].SerialEncDataA returns interpolated Z Data

Hex Digit ($)																								-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	-	-
Component:	*Interpolated Commanded Z Position*																								

The 24-bit registers **Acc84E[*i*].Chan[3].SerialEncDataA** is update every frame cycle and it includes the full status word returned by scanhead/galvanometer servo drive.

> ***Note***
>
> The status bit definition varies between equipment vendors and it is strongly recommended for the user to consult with appropriate documentation on the target hardware for further information on these status bits.

The status bit definition based upon XY2-100 Serial Link 1 Specification by General Scanning (GSI) is defined as below.

Hex Digit ($)																					0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	S18	S17	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0	P	-	-	-	-	-	-
Component:	*Returned Status Bits*																								

Bit	2-Axis Status	3-Axis Status

S18	0	0
S17	1	0
S16	1	1
S15	Power Status	X Error Status (X Servo Ready)
S14	Temperature Status	X Temperature Status
S13	In-field	X Tracking Error
S12	X Position Acknowledge	0
S11	Y Position Acknowledge	Y Error Status (Y Servo Ready)
S10	1	Y Temperature Status
S9	0	Y Tracking Error
S8	1	0
S7	Power Status	Z Error Status (Z Servo Ready)
S6	Temperature Status	Z Temperature Status
S5	In-field	Z Tracking Error
S4	X Position Acknowledge	0
S3	Y Position Acknowledge	Serial Link X Parity Error
S2	1	Serial Link Y Parity Error
S1	0	Serial Link Z Parity Error
S0	1	Serial Link Clock Error
P	x (no parity)	even parity

Power PMAC Setup Example

A few separate setup elements should be modified from their default values for using ACC-84E with XY2-100 interface protocol as a communication channel between Power PMAC and scanhead. The following sections outline an example setup where a 2 axis scanhead, driven by a galvo-servo drive is connected to ACC-84E and motors 1 & 2 of PMAC are setup to control these galvanometers.

ACC-84E Setup Element Example

The servo drive used in this example expects a standard 16-bit position command with a 2MHz serial clock frequency and odd parity calculation..

SerialClockMDiv:	= $18	// Serial clock frequency = bit transmission frequency
SerialClockNDiv:	= 1	
TxEnable:	= 1	// Enable transmission of XY2-100 position data
Parity:	= 1	// Odd parity selection (0: Even, 1: Odd)
ClockSel:	= 0	// Servo clock selection for interpolation
ModeSel:	= $0	// XY2-100 Serial Link Standard (16-bit) position data

For a 2 MHz bit transmission rate, ***SerialClockMDiv*** = 24 ($18) and ***SerialClockNDiv*** = 1 ($1) **Acc84E[*i*].SerialEncCtrl** is set to $181C00 for interpolation based upon Servo clock and odd parity calculation.

Hex Digit ($)	1				8				1				C				0				0				-	-
Script Bit #	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
Bit Value	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	-	-
Component:	*SerialClockMDiv*								*SerialClockNDiv*				*TE*	*P*	*CS*	--	*ModeSel*									

```
Acc84E[0].SerialEncCtrl = $181C00
```

Encoder Conversion Table Example

Standard XY2-100 protocol does not provide any real feedback from the galvanometers to PMAC since the position loop is closed in the servo drive or scanhead and not in the controller. In ACC-84E, XY2-100 implementation, a simulated feedback is provided which provides a pseudo-feedback for use in Power PMAC ECT. This feedback represents the interpolated position command which is sent to XY2-100 at 1/20 rate of serial clock frequency set by **Acc84E[*i*].SerialEncCtrl**.

```
EncTable[1].type = 1
EncTable[1].pEnc = Acc84E[0].Chan[0].SerialEncDataA.a
EncTable[1].pEnc1 = Sys.pushm
EncTable[1].index1 = 8
EncTable[1].index2 = 8
EncTable[1].index3 = 0
EncTable[1].index4 = 0
EncTable[1].index5 = 0
EncTable[1].index6 = 0
EncTable[1].ScaleFactor = 1

EncTable[2].type = 1
EncTable[2].pEnc = Acc84E[0].Chan[1].SerialEncDataA.a
EncTable[2].pEnc1 = Sys.pushm
EncTable[2].index1 = 8
EncTable[2].index2 = 8
EncTable[2].index3 = 0
```

```
EncTable[2].index4 = 0
EncTable[2].index5 = 0
EncTable[2].index6 = 0
EncTable[2].ScaleFactor = 1
```

Power PMAC Motor Setup Example

Motor setting in Power PMAC is much simpler in Power PMAC because of built-in position control servo algorithm. The motor settings required for commanding XY2-100 axis can be separated into couple of groups. First set of parameters are common regardless of the resolution of XY2-100 drive used.

```
Motor[1].Ctrl=Sys.PosCtrl
Motor[1].pDac = Acc84E[0].Chan[0].SerialEncCmd.a
Motor[1].pEnc = EncTable[1].a
Motor[1].pEnc2 = EncTable[1].a
Motor[1].pAmpEnable = 0
Motor[1].pAmpFault = 0
Motor[1].pLimits = 0
Motor[1].ServoCtrl = 1
Motor[1].pAbsPos = Acc84E[0].Chan[0].SerialEncDataA.a
Motor[1].AbsPosFormat = $01001808
Motor[1].PowerOnMode=4
Motor[1].FatalFeLimit=0
Motor[1].WarnFeLimit=0

Motor[2].Ctrl=Sys.PosCtrl
Motor[2].pDac = Acc84E[0].Chan[1].SerialEncCmd.a
Motor[2].pEnc = EncTable[2].a
Motor[2].pEnc2 = EncTable[2].a
Motor[2].pAmpEnable = 0
Motor[2].pAmpFault = 0
Motor[2].pLimits = 0
Motor[2].ServoCtrl = 1
Motor[2].pAbsPos = Acc84E[0].Chan[1].SerialEncDataA.a
Motor[2].AbsPosFormat = $01001808
Motor[2].PowerOnMode=4
Motor[2].FatalFeLimit=0
Motor[2].WarnFeLimit=0
```

The following settings are dependent on the selected data length mode:

ModeSel = 00
16-bit data format (XY2-100 Standard, Serial Link)

```
Motor[1].PosSf = 1/exp2(16)
Motor[1].Pos2Sf = Motor[1].PosSf
Motor[1].AbsPosSf = 1/exp2(8)
Motor[1].MaxPos = exp2(15)-1
Motor[1].MinPos = -exp2(15)
Motor[1].MaxDac = exp2(15)

Motor[2].PosSf = 1/exp2(16)
Motor[2].Pos2Sf = Motor[2].PosSf
Motor[2].AbsPosSf = 1/exp2(8)
Motor[2].MaxPos = exp2(15)-1
Motor[2].MinPos = -exp2(15)
Motor[2].MaxDac = exp2(15)
```

ModeSel = 01
18-bit data format (Serial Link 2)

```
Motor[1].PosSf = 1/exp2(14)
Motor[1].Pos2Sf = Motor[1].PosSf
Motor[1].AbsPosSf = 1/exp2(6)
Motor[1].MaxPos = exp2(17)-1
Motor[1].MinPos = -exp2(17)
Motor[1].MaxDac = exp2(17)

Motor[2].PosSf = 1/exp2(14)
Motor[2].Pos2Sf = Motor[1].PosSf
Motor[2].AbsPosSf = 1/exp2(6)
Motor[2].MaxPos = exp2(17)-1
Motor[2].MinPos = -exp2(17)
Motor[2].MaxDac = exp2(17)
```

ModeSel = 10
20-bit data format

```
Motor[1].PosSf = 1/exp2(12)
Motor[1].Pos2Sf = Motor[1].PosSf
Motor[1].AbsPosSf = 1/exp2(4)
Motor[1].MaxPos = exp2(19)-1
Motor[1].MinPos = -exp2(19)
Motor[1].MaxDac = exp2(19)

Motor[2].PosSf = 1/exp2(12)
Motor[2].Pos2Sf = Motor[1].PosSf
Motor[2].AbsPosSf = 1/exp2(4)
Motor[2].MaxPos = exp2(19)-1
Motor[2].MinPos = -exp2(19)
Motor[2].MaxDac = exp2(19)
```

Motor Speed/Acceleration Limitations

Power PMAC's default acceleration and speed setting are set at conservative values. In comparison, galvanometers and scanheads can complete a full stroke step move in a couple of microseconds. The following settings are suggested for preventing PMAC from limiting the speeds and accelerations achievable by the galvos. However, it is suggested that the proper calculated values based upon the specifications provided by manufactures of the galvos are implemented instead.

ModeSel = 00 16-bit data format (XY2-100 Standard, Serial Link)

```
Motor[1].MaxSpeed = exp2(15)
Motor[1].InvAMax = 1/exp2(15)
Motor[1].InvDMax = 1/exp2(15)
Motor[1].InvJMax = 1/exp2(15)

Motor[2].MaxSpeed = exp2(15)
Motor[2].InvAMax = 1/exp2(15)
Motor[2].InvDMax = 1/exp2(15)
Motor[2].InvJMax = 1/exp2(15)
```

Initializing Motor Position

Although simulated feedback provides interpolated position data for PMAC, there is a caveat in general with using simulated position feedbacks and that is a potential positive feedback when the motor is in killed state.
This positive feedback, or virtual runaway at constant speed, is cause by two independent processes where the result of one, is fed back to the second one and vice versa. The first process is that PMAC firmware copies the actual position of the motor into its desired position when the motor is killed. This is necessary in order to prevent any position jumps when the motor is put to closed loop mode. The actual position of the motor is read from the ECT which is processed at the beginning of the servo cycle. The commanded position is written to output, **ACC84E[*i*].Chan[*j*].SerialEncCmd** in this case, at the end of servo calculations but will take effect only on the rising edge of the servo clock which can be as little as 50% delayed until the next ECT execution. If the initial motor position value and commanded value are the same, there is no problem. But if there is a difference, however small, it will cause a positive feedback or virtual runaway at constant speed.

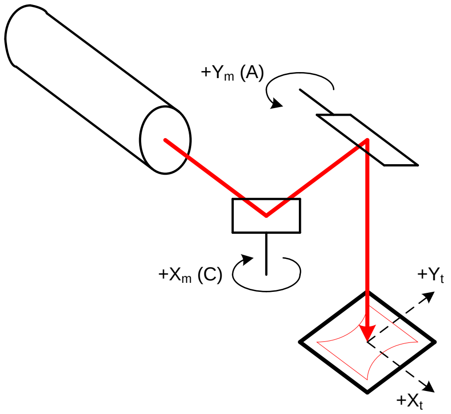
In order to prevent this from happening two counter measures should be implemented:

1. User should setup the **Motor[*x*].PowerOnMode** register to a value of 4, forcing the read of the simulated feedback upon power-on/reset.
2. Implement the following code during development of the code where the **Motor[*x*].PowerOnMode** is not active until next power up/reset cycle.
   This code breaks the positive feedback for a moment which is sufficient for allowing the feedback and commanded values for the motor to be equal.

```
#1..3KILL
EncTable[1].ScaleFactor=0
EncTable[2].ScaleFactor=0
EncTable[3].ScaleFactor=0
Motor[1].HomePos=0
Motor[2].HomePos=0
Motor[3].HomePos=0
Motor[1].Pos=0
Motor[2].Pos=0
Motor[3].Pos=0

// brief delay required here

EncTable[1].ScaleFactor=1
EncTable[2].ScaleFactor=1
EncTable[3].ScaleFactor=1
```

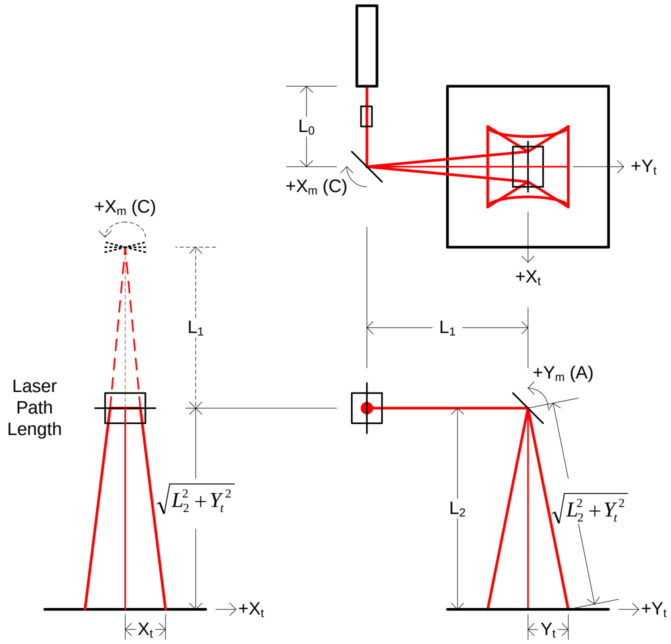
Non-linearity of Scanheads

The pin-cushion effect caused by galvanometers, can be corrected using kinematic routines. A typical configuration of a laser mirror system is shown in isometric view below:



Basic Laser Mirror Arrangement

The forward and inverse-kinematic equations for a basic system of this type are shown in the following diagram with orthogonal views of the system:



$$X_t = \left(L_1 + \sqrt{L_2^2 + Y_t^2}\right)\tan\,(2C)$$

$$C = \frac{1}{2}\tan^{-1}\left(\frac{X_t}{L_1 + \sqrt{L_2^2 + Y_t^2}}\right)$$

$$Y_t = L_2\tan(2A)$$

$$A = \frac{1}{2}\tan^{-1}\left(\frac{Y_t}{L_2}\right)$$

Basic Laser Mirror Kinematics

All of the actuators, both for the workpiece holder and for the laser control, are defined as inverse-kinematic axes in the same coordinate system.

Here is a simple implementation of forward and inverse kinematic routines for a given galvanometer:

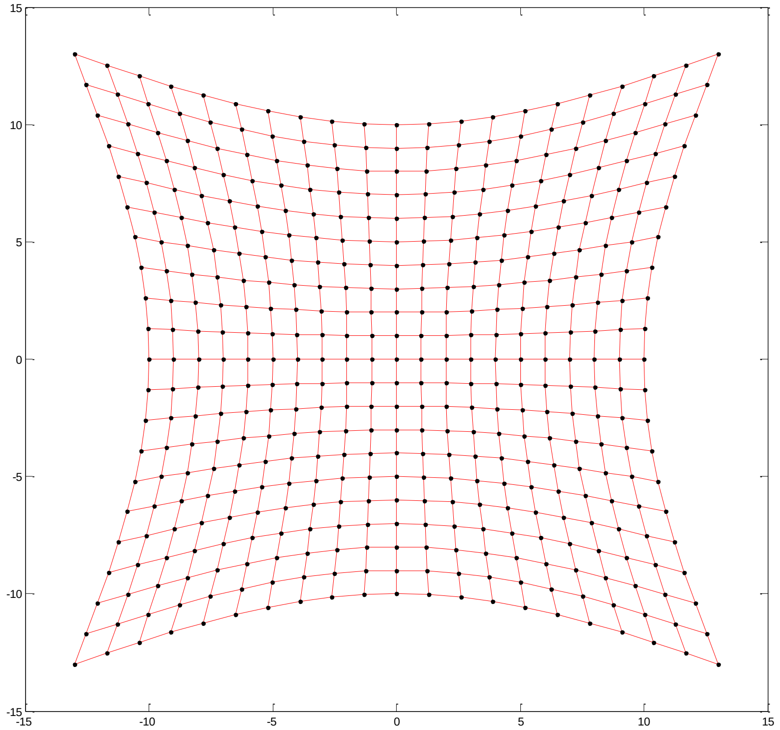
```
global pi = 3.14159265358979323846264338832795;                      // constant pi
global DegtoRad = 0.01745329251994329576923690768489;                // Degrees to Radians
global RadtoDeg = 57.295779513082320876798154814105;                 // Radians to Degrees
global Len1 = 0.374 * 25.4;       // distance between spot center on mirrors at neutral position (mm)
global Len2 = 251;                // distance from Y-mirror to XY-stage surface (mm)
global GalvoSF   = 10.27/32767;   // ratio of XY2-100 command ±32767 to Galvonameter deflection

open forward (1)
        local GalvoXAngD, GalvoYAngD, Ytemp;
        if(KinVelEna==0) KinAxisUsed=$C0;
        GalvoXAngD = KinPosMotor1 * GalvoSF;
        GalvoYAngD = KinPosMotor2 * (-GalvoSF);
        Ytemp = Len2 * tan(GalvoYAngD * DegtoRad);
        KinPosAxisY = Ytemp;
        KinPosAxisX = ( Len1+sqrt(pow(Len2,2)+pow(Ytemp,2))) * tan(GalvoXAngD* DegtoRad);
close

open inverse (1)
        KinPosMotor2 = atan(KinPosAxisY/Len2)* RadtoDeg/(-GalvoSF);
        KinPosMotor1 = atan(KinPosAxisX/(Len1+sqrt(pow(Len2,2)+pow(KinPosAxisY,2))))*RadtoDeg/GalvoSF;
close
```

Corrections for Scanner/Optics Non-linearity

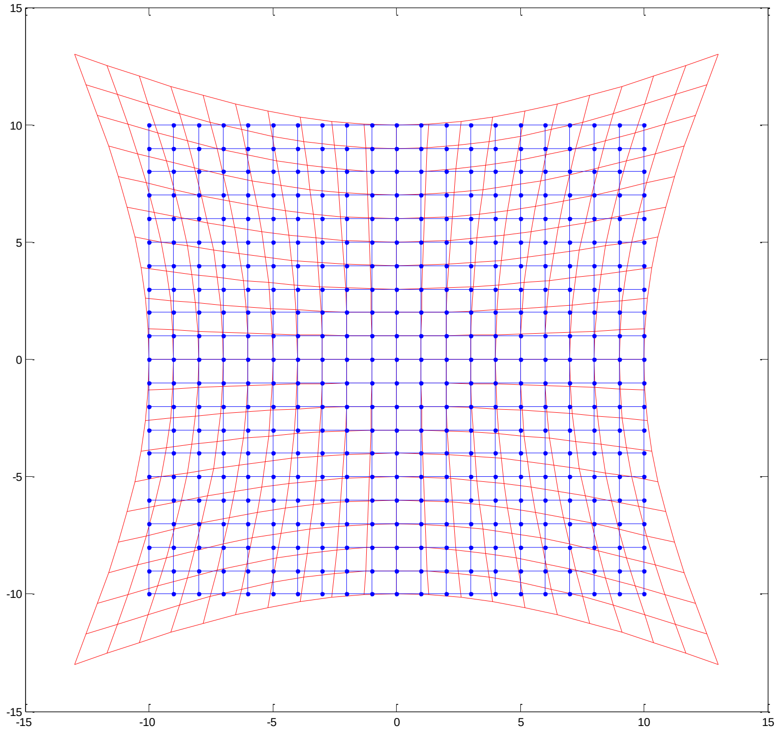
The inherent non-linearity of the optics and scanners can be compensated for using PMAC's built-in 2D compensation tables. This method requires physical marking and measurement of patterns (mostly uniformly spaced matrix patterns) in order to calculate the correction tables required for each of the galvos based upon commanded position of them. This method is especially useful if any optics such as Flat-field, F-theta or Telecenteric lenses are involved and their non-linearities also need to be compensated.

The following steps should be followed in generating the compensation table:

1. Select a grid size which covers the full command range for the galvanometer (for example a grid which covers ±32767 if a 16-bit command is used). For this example, a 3D array called "CmdTable" is constructed with indexes *i*,*j* and *k* for each of the dimensions. The X and Y galvo command position for node row *i* and column *j* are stored in elements *k*=1 and *k*=2 respectively.
2. Program the PMAC to generate the grid points defined above on the work piece, by marking or etching.
3. Identify the resulting locations on the workpiece (an exaggerated version of resulted matrix is shown below in red with resulting nodes corresponding to CmdTable entries shown in black dots).
   The X and Yposition of each node should to be measured in engineering units, and stored. In this example the measured locations are stored in a 3D array called "RawTable". This array has the same dimensions as the "CmdTable":



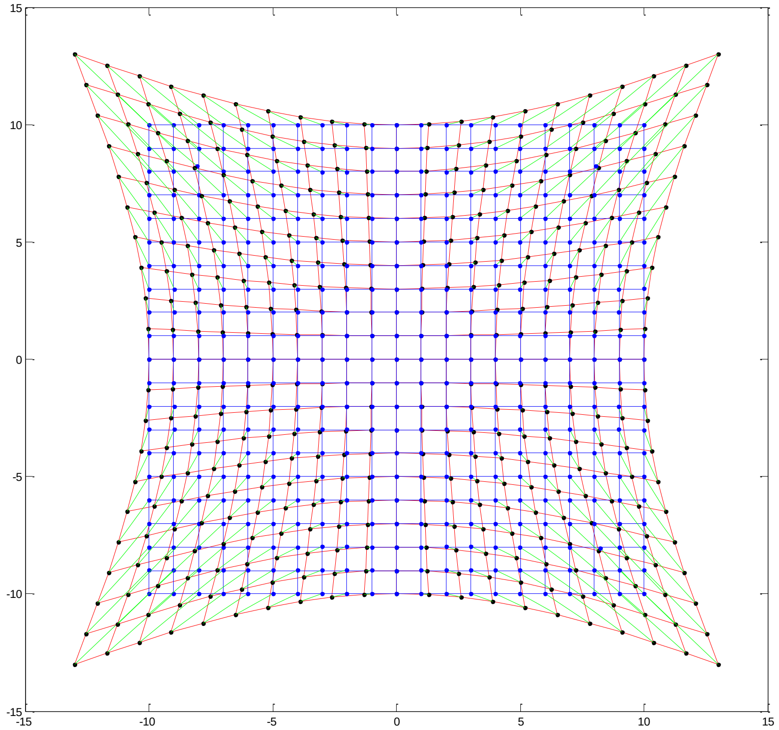
4. Identify the maximum usable workspace with a rectangular outline. Notice that the compensation table cannot make corrections for rotation, so the selected rectangle direction should match the direction of distorted XY field. There are multiple methods for optimizing the best usable area which are beyond the intended scope of this document. A simple approach in selection of this area is shown in this example.



5. Generate a table with same dimensions as the “CmdTable” and “RawTable”, called “BestFitTable” in which the XY coordinates of nodes for the best fit grid are stored. In the picture above node are shown in blue dots.

   The next steps are automated in the code, but explained for reference.

6. Identify mesh element of “RawTable” where each “BestFitTable” node is located.
7. Calculate the interpolated value of “BestFitTable” node, based upon its “RawTable” surrounding nodes.
8. Transfer the interpolation from “RawTable” to “CmdTable” corresponding nodes/mesh item and store it in “Corrected” array.
9. The difference between “CmdTable” and “Corrected” table are the entries for the two 2D compensation tables.



The following example code is written in MATLAB script, but it can be converted to any language with array support.

```
%Main Procedure

Res = 32767;
HalfLen = 10;
MeshSize = 21;

hold off;
plot(RawTable(:,:,1),RawTable(:,:,2),'red');
hold on;
plot(RawTable(:,:,1)',RawTable(:,:,2)','red');

xmin = max(RawTable(1:1,:,1));
xmax = min(RawTable(MeshSize:MeshSize,:,1));
ymin = max(RawTable(:,1:1,2));
ymax = min(RawTable(:,MeshSize:MeshSize,2));

x_temp = linspace(xmin,xmax,MeshSize);
y_temp = linspace(ymin,ymax,MeshSize);

BestFitTable = zeros(MeshSize,MeshSize,2);
for i = 1:MeshSize
  BestFitTable(:,i,1)=x_temp';
  BestFitTable(i,:,2)=y_temp;
end

plot(BestFitTable(:,:,1),BestFitTable(:,:,2),'blue');
plot(BestFitTable(:,:,1)',BestFitTable(:,:,2)','blue');

meshdist = zeros(MeshSize-1,MeshSize-1);
meshid = zeros(MeshSize,MeshSize,2);
for i= 1:MeshSize
 for j=1:MeshSize
  for m = 1:MeshSize-1
   for n = 1:MeshSize-1
    meshdist(m,n) = sqrt((BestFitTable(i,j,1)-RawTable(m,n,1))^2+(BestFitTable(i,j,2)-RawTable(m,n,2))^2) + ...
```

```
           sqrt((BestFitTable(i,j,1)-RawTable(m+1,n,1))^2+(BestFitTable(i,j,2)-RawTable(m+1,n,2))^2) + ...
           sqrt((BestFitTable(i,j,1)-RawTable(m+1,n+1,1))^2+(BestFitTable(i,j,2)-RawTable(m+1,n+1,2))^2) + ...
           sqrt((BestFitTable(i,j,1)-RawTable(m,n+1,1))^2+(BestFitTable(i,j,2)-RawTable(m,n+1,2))^2) ;
   end
  end
  if (BestFitTable(i,j,1)==RawTable(m,n,1) && BestFitTable(i,j,2)==RawTable(m,n,2))
   meshid(i,j,1) = m;
   meshid(i,j,2) = n;
  else
   [MinLen,MinInd]=min(meshdist(:));
   [meshid(i,j,1),meshid(i,j,2)] = ind2sub(size(meshdist),MinInd);
  end
 end
end

Corrected = zeros(MeshSize,MeshSize,2);
for i= 1:MeshSize
    for j=1:MeshSize
       [Corrected(i,j,1),Corrected(i,j,2)] = PlanarInterpolation( ...
            CmdTable(meshid(i,j,1):meshid(i,j,1)+1,meshid(i,j,2):meshid(i,j,2)+1,:), ...
            RawTable(meshid(i,j,1):meshid(i,j,1)+1,meshid(i,j,2):meshid(i,j,2)+1,:),  ...
            BestFitTable(i,j,1),  BestFitTable(i,j,2) );
    end
end
```

The following functions were used in this procedure.

```
function [ Xint, Yint ] = PlanarInterpolation( CmdPos, ActPos , Xinput, Yinput )

x1 = ActPos(1,1,1);
y1 = ActPos(1,1,2);
x2 = ActPos(1,2,1);
y2 = ActPos(1,2,2);
x3 = ActPos(2,1,1);
y3 = ActPos(2,1,2);
x4 = ActPos(2,2,1);
y4 = ActPos(2,2,2);

X1 = CmdPos(1,1,1);
Y1 = CmdPos(1,1,2);
X2 = CmdPos(1,2,1);
Y2 = CmdPos(1,2,2);
X3 = CmdPos(2,1,1);
Y3 = CmdPos(2,1,2);

m12=(y2-y1)/(x2-x1);
m13=(y3-y1)/(x3-x1);
m24=(y4-y2)/(x4-x2);
m34=(y4-y3)/(x4-x3);

if (m12==m34)
    m1=m12;
else
    [xe1,ye1] = LineIntersection(x1,y1,x2,y2,x3,y3,x4,y4);
    m1=(ye1-Yinput)/(xe1-Xinput);
end

if (m13==m24)
    m2=m13;
else
    [xe2,ye2] = LineIntersection(x1,y1,x3,y3,x2,y2,x4,y4);
    m2=(ye2-Yinput)/(xe2-Xinput);
end

if (isinf(m1))
    x13 = Xinput;
    y13 = m13*(x13-x1)+y1;
else
    if (isinf(m13))
        x13 = x1;
        y13 = m1 * (x13 - Xinput) + Yinput;
    else
    x13 = (m1*Xinput - m13*x1 +y1 - Yinput)/(m1-m13);
    y13 = m13*(x13-x1)+y1;
    end
end
```

```
if (isinf(m2))
    x12 = Xinput;
    y12 = m12*(x12-x1)+y1;
else
    if (isinf(m12))
        x12 = x1;
        y12 = m2 * (x12 - Xinput) + Yinput;
    else
    x12 = (m2*Xinput - m12*x1 +y1 - Yinput)/(m2-m12);
    y12 = m12*(x12-x1)+y1;
    end
end

ratio1 = sqrt((x12-x1)^2+(y12-y1)^2)/sqrt((x2-x1)^2+(y2-y1)^2);
ratio2 = sqrt((x13-x1)^2+(y13-y1)^2)/sqrt((x3-x1)^2+(y3-y1)^2);

Xint= X1 + ratio2 * (X3-X1);
Yint= Y1 + ratio1 * (Y2-Y1);
end

function [ X,Y ] = LineIntersection( x1,y1,x2,y2,x3,y3,x4,y4 )
X = ((x1*y2-y1*x2)*(x3-x4)-(x1-x2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4));
Y = ((x1*y2-y1*x2)*(y3-y4)-(y1-y2)*(x3*y4-y3*x4))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4));
end
```