Converting a Turbo PMAC Application to Power PMAC

This application note explains how to convert the setup for a Turbo PMAC application to an equivalent Power PMAC setup. The emphasis of this note is conversion with minimum changes, to be done in the shortest time. *This strategy does not take full advantage of many new Power PMAC capabilities that do not have direct Turbo PMAC equivalents.*

More detail on any aspect of the Power PMAC implementation can be found in the User's Manual and the Software Reference Manual.

This note focuses on four specific conversions, which differ from each other primarily in their mapping of software to hardware:

1. Turbo Clipper to Power Clipper
2. Turbo Brick to Power Brick
3. Turbo UMAC to Power UMAC or CK3M (µPower UMAC)
    a. Turbo UMAC with ACC-24E2*x*, ACC-51E, and ACC-84E, plus I/O boards, to Power UMAC with same*
    b. Turbo UMAC with ACC-24E2*x*, ACC-51E, and ACC-84E, plus I/O boards, to Power UMAC with ACC-24E3 and I/O boards
    c. Turbo UMAC with ACC-24E2*x*, ACC-51E, and ACC-84E, plus I/O boards, to CK3M (µPower UMAC) with CK3W axis and I/O boards
4. Turbo PMAC2 Ultralite to Power PMAC with MACRO (e.g. Etherlite)
    a. Turbo PMAC2 Ultralite to Power PMAC with ACC-5E
    b. Turbo PMAC2 Ultralite to Power PMAC with ACC-5E3 or ACC-5EP3

Many aspects of the conversion are common among all of these conversions. Others, particularly those dealing with the ASIC setup registers, are particular to a type of conversion.

This note will also be helpful for other conversions, including those from non-Turbo PMACs. The I-variable numbering scheme on non-Turbo PMACs is somewhat different, but of a similar concept.

---

* The conversion from Turbo Compact UMAC with ACC-24C2*x*, ACC-51C, and ACC-84C, plus I/O, to Power Compact UMAC with the same accessories is exactly parallel to this. Just substitute "C" in the accessory element names for "E".

Contents

Contents

Basic Communications

While the underlying details of communications are substantially different between Turbo PMAC and Power PMAC, the implications for actual use in an actual application are quite minor.

Hardware Communications Ports

Each Turbo PMAC has two hardware communications ports. The first is a serial port, either RS-232 or RS-422, depending on the particular version. This port has seldom been used in actual applications due to its low speed, but is fully capable of sending commands and program lines, and receiving responses.

The second is a "bus" port, presenting parallel data interfaces to the processor, with higher speed capabilities. The earliest Turbo PMAC configurations utilized true parallel buses – ISA, VME, PC/104, and PCI. Later configurations provided "wire buses" – USB and Ethernet – through a microcontroller that provided a parallel interface to the Turbo PMAC processor.

Power PMAC uses only Ethernet ports for its host communications. (A small serial interface is used only for factory configuration and occasional diagnostics.) A Power PMAC system may have 1, 2, or 4 hardware Ethernet ports, depending on the configuration and the options ordered.

In virtually all Power PMAC systems, only one hardware Ethernet port is used for communications with the "host" computer. Other ports, if present, are typically used for industrial networking (e.g. EtherCAT, Ethernet/IP, Modbus TCP) with other devices.

Because the Ethernet ports in Power PMAC interface directly to the main Power PMAC processor, the communications speed over Ethernet to the Power PMAC processor (whether at 100 Mbit/sec or 1 Gbit/sec) is faster than the communications speed over backplane buses like PCI to the Turbo PMAC processor. This means that Turbo PMAC applications can always be converted to Power PMAC without losing communications speed.

Development Software Communications

For Turbo PMAC systems, the PC software used to develop the application is called the "PMAC Executive". For Power PMAC systems, the equivalents software is called the "Integrated Development Environment" (IDE).

Both programs provide an initial window that allows the user to specify how to communicate to the PMAC controller. While the PMAC Executive window provides many communications options, the Power PMAC IDE window only provides Ethernet options. Usually all the user must specify is the IP address, leaving other options at their default values.

Once communications is established, Turbo PMAC users will find the basic capabilities of the Power PMAC IDE very similar to those of the older PMAC Executive. Underlying details of the communications interface are completely hidden from the user.

Saved Setup Data Structure Elements (I-Variables)

A key part of the configuration of an application is the saved configuration variables, called I-variables in Turbo PMAC, and saved setup data-structure elements in Power PMAC. A table of equivalencies between Turbo PMAC I-variable values and Power PMAC saved setup data structure elements can be found in the Power PMAC Software Reference Manual chapter *Power PMAC Turbo I-Variable Equivalents*. This note provides more detailed conversion instructions.

Global I-Variable Equivalents

The global I-variables in Turbo PMAC have numbers below 100. Many of their Power PMAC counterparts are elements in the **Sys** data structure, or in other non-motor, non-coordinate-system data structures.

Cycle Time Parameters

- Set **Sys.PhaseCycleExt** equal to **I7** for the same extension of the phase cycle.
- Set **Sys.RtIntPeriod** equal to **I8** for equivalent period for the real-time interrupt.
- Set **Sys.ServoPeriod** equal to (**I10** / 8,388,608) for the same numerical time value to be used in interpolation calculations.

PLC Program Control Parameters

In Turbo PMAC, if the saved value of **I5** is greater than 0, existing PLC programs are automatically enabled on power-on/reset.

- In Power PMAC, there is no direct equivalent of **I5**, and existing PLC programs are not automatically enabled on power-on/reset. To have some or all of the existing PLC programs enabled on power-on/reset, use an **`enable plc`** command in the project file `pp_startup.txt`.
- Set Power PMAC parameter **Sys.MaxRtPlc** to its default value of 0 so that PLC 0 will run in foreground (under the real-time interrupt) and PLCs 1 – 31 will run in background. This configuration is fixed in Turbo PMAC.

Multiplexed I/O Parameters

These parameters are only important if using a multiplexed I/O accessory such as an ACC-34*x*.

- **I27** to **MuxIo.InBit** and **MuxIo.OutBit**:
    - For the Power Clipper, set **MuxIo.InBit** to 0 if **I27** = 0, or to 7 if **I27** = 1. Set **MuxIo.OutBit** to 8.
    - For the Power UMAC with ACC-5E, set **MuxIo.InBit** to 8 if **I27** = 0, or to 15 if **I27** = 1. Set **MuxIo.OutBit** to 16.
    - For the Power UMAC with ACC-5E3, set **MuxIo.InBit** to 0 if **I27** = 0, or to 7 if **I27** = 1. Set **MuxIo.OutBit** to 8.

- **I29** to **MuxIo.pIn** and **MuxIo.pOut**:
    - For the Power Clipper, set **MuxIo.pIn** and **MuxIo.pOut** to **Clipper[0].GpioData[0].a**.
    - For the Power UMAC with ACC-5E, set **MuxIo.pIn** and **MuxIo.pOut** to **Acc5E[*i*].MuxData.a**.
    - For the Power UMAC with ACC-5E3, set **MuxIo.pIn** and **MuxIo.pOut** to **Acc5E3[*i*].GpioData[0].a**..

See the separate section below on multiplexed I/O for more details.

Compensation Table Parameters

- **I30** to **CompTable[*m*].Ctrl**:
    - Turbo PMAC compensation tables always roll over. To set up Power PMAC compensation tables to roll over, set bits 2 – 7 of **CompTable[*m*].Ctrl** to their default value of 0.
- **I51** to **Sys.CompEnable**:
    - If **I51** = 0, let **Sys.CompEnable** stay at its power-on default value of 0 to disable all tables.
    - If **I51** = 1, set **Sys.CompEnable** greater than the highest table number after each power-on/reset to enable all tables. (**Sys.CompEnable** is not a saved setup element.) Many users prefer to enable the tables after the motors have established their position references.

Communications Parameters

- **I9** to **echo** command value (for the same variable query response format)
    - If **I9** = 0, set **echo** command value = 7.
    - If **I9** = 1, set **echo** command value = 0.
    - If **I9** = 2, set **echo** command value = 15.
    - If **I9** = 3, set **echo** command value = 8.
- **I62** to **Sys.SendFileMode** (for similar termination of unsolicited messages)
    - If **I62** = 0, set each bit *n* (for active port *n*) of **Sys.SendFileMode** to 1 to add termination character to sent string.
    - If **I62** = 1, set each bit *n* (for active port *n*) of **Sys.SendFileMode** to 0 so no termination character is added to sent string.

MACRO Timing Parameters

These parameters only need to be set if using the MACRO ring.

- Set **Macro.IOTimeOut** equal to (**I78** * **I10** / 8,388,608) for the same time-out on MACRO background transfers. (Turbo PMAC permits separate input and output time-out values with **I78** and **I79**; Power PMAC has a common setting for both.)
- Set **Macro.TestPeriod** equal to **I80** for the same operation.
- Set **Macro.TestMaxErrors** equal to **I81** for the same operation.
- Set **Macro.TestReqdSynchs** equal to **I82** for the same operation.

ASIC I-Variable Equivalents

In some of the conversions, the same PMAC2-style Servo and/or MACRO ASICs are used, and there is simply a translation of the names of these setup elements. In other conversions, the PMAC-style ASICs are updated to the newer PMAC3-style ASICs, and the conversion are slightly more involved.

Turbo PMAC2 DSPGATE1 Servo IC I-Variables to Power PMAC DSPGATE1 Servo IC Elements

This conversion is necessary for Turbo UMAC ACC-24E2*x* to Power PMAC ACC-24E2*x*.

- Power PMAC ASIC index "***i***" equals (2 * ***m***) in Turbo PMAC I-variable number.
- Power PMAC ASIC channel index "***j***" equals (***n*** – 1) in Turbo PMAC I-variable number.

The data structure name shown here is the generic **Gate1[*i*]**. You can also use the hardware-specific alias name of **Acc24E2[*i*]**, **Acc24E2A[*i*]**, or **Acc24E2S[*i*]**.

IC Multi-Channel Parameters

- Set **Gate1[*i*].PwmPeriod** equal to **I7*m*00** for the same PWM and MaxPhase frequency.
- Set **Gate1[*i*].PhaseClockDiv** equal to **I7*m*01** for the same division from MaxPhase to Phase frequency.
- Set **Gate1[*i*].ServoClockDiv** equal to **I7*m*02** for the same division from Phase to Servo frequency.
- Set **Gate1[*i*].PhaseServoDir** equal to **I7*m*07** for the same direction of the phase and servo clocks. (These will typically be set automatically at power-on/reset.)
- Set **Gate1[*i*].HardwareClockCtrl** equal to **I7*m*03** for the same frequency settings of ASIC hardware clocks.
- Set **Gate1[*i*].PwmDeadTime** equal to **I7*m*04** for the same off time between top and bottom PWM on states.
- Set **Gate1[*i*].DacStrobe** equal to **I7*m*05** for the same strobe signal output for D/A converters.
- Set **Gate1[*i*].AdcStrobe** equal to **I7*m*06** for the same strobe signal output for A/D converters.

IC Single-Channel Parameters

- Set **Gate1[*i*].Chan[*j*].EncCtrl** equal to **I7*mn*0** for the same encoder decode method.
- Set **Gate1[*i*].Chan[*j*].Equ1Ena** equal to **I7*mn*1** for the same compare channel select.
- Set **Gate1[*i*].Chan[*j*].CaptCtrl** equal to **I7*mn*2** for the same capture signal select.
- Set **Gate1[*i*].Chan[*j*].CaptFlag** equal to **I7*mn*3** for the same capture flag select.

- Set **Gate1[*i*].Chan[*j*].GatedIndexSel** equal to **I7*mn*4** for the same use of full or gated index.
- Set **Gate1[*i*].Chan[*j*].IndexGateState** equal to **I7*mn*5** for the same gating for index.
- Set **Gate1[*i*].Chan[*j*].OutputMode** equal to **I7*mn*6** for the same command output type.
- Set **Gate1[*i*].Chan[*j*].OutputPol** equal to **I7*mn*7** for the same command output polarity.
- Set **Gate1[*i*].Chan[*j*].PfmDirPol** equal to **I7*mn*8** for the same PFM direction signal polarity.
- **I7*mn*9** to **Gate1[*i*].Chan[*j*].OneOverTEna**:
    - Set **Gate1[*i*].Chan[*j*].OneOverTEna** to 0 to disable hardware 1/T.
    - If **I7*mn*9** = 0 (hardware 1/T disabled), no further changes need to be made.
    - If **I7*mn*9** = 1 (hardware 1/T enabled), this mode of operation is not supported by Power PMAC software. The encoder feedback must be processed by **type** = 3 (software 1/T) in the Power PMAC Encoder Conversion Table.

Turbo PMAC2 DSPGATE1 Servo IC I-Variables to Power PMAC DSPGATE3 Elements

This conversion is necessary for the following updates:

Turbo Clipper to Power Clipper

- Power PMAC ASIC index "***i***" equals ***m*** in Turbo PMAC I-variable number.
- Power PMAC ASIC channel index "***j***" equals (***n*** – 1) in Turbo PMAC I-variable number.

The data structure name shown here is the generic **Gate3[*i*]**. You can also use the hardware-specific alias name of **Clipper[*i*]**.

Turbo Brick to Power Brick

- Power PMAC ASIC index "***i***" equals ***m*** in Turbo PMAC I-variable number.
- Power PMAC ASIC channel index "***j***" equals (***n*** – 1) in Turbo PMAC I-variable number.

The data structure name shown here is the generic **Gate3[*i*]**. You can also use the hardware-specific alias name of **PowerBrick[*i*]**.

Turbo UMAC ACC-24E2*x* to Power UMAC ACC-24E3 or CK3E with CK3W-AX

- Power PMAC ASIC index "***i***" equals (***m*** – 2) in Turbo PMAC I-variable number.
- Power PMAC ASIC channel index "***j***" equals (***n*** – 1) in Turbo PMAC I-variable number.

The data structure name shown here is the generic **Gate3[*i*]**. You can also use the hardware-specific alias name of **Acc24E3[*i*]**.

IC Multi-Channel Parameters

- Set **Gate3[*i*].PhaseFreq** equal to [117,964,800 / (2 * **I7*m*00** + 3) / (**I7*m*01** + 1)] for the same Phase clock frequency.

- Set **Gate3[*i*].PhaseClockDiv** and **Gate3[*i*].PhaseClockMult** to 0 for the same operation.

- Set **Gate3[*i*].ServoClockDiv** equal to **I7*m*02** for the same division from Phase to Servo frequency.

- Set **Gate3[*i*].PhaseServoDir** equal to **I7*m*07** for the same direction of the phase and servo clocks. (These will typically be set automatically at power-on/reset.)

- Set **Gate3[*i*].EncClockDiv** equal to [(**I7*m*03** & 7) – 1] for the closest (slightly higher) encoder sampling frequency.

- Set **Gate3[*i*].PfmClockDiv** equal to [(**I7*m*03** & 56) / 8 – 1] for the closest (slightly higher) pulse-frequency modulation update frequency.

- Set **Gate3[*i*].DacClockDiv** equal to [(**I7*m*03** & 448) / 64 – 1] for the closest (slightly higher) D/A converter update frequency.

- Set **Gate3[*i*].AdcEncClockDiv** equal to [(**I7*m*03** & 3584) / 512 – 1] for the closest encoder (slightly higher) A/D converter update frequency.

- Set **Gate3[*i*].AdcAmpClockDiv** equal to [(**I7*m*03** & 3584) / 512 – 1] for the closest (slightly higher) amplifier A/D converter update frequency.

- Set **Gate3[*i*].DacStrobe** equal to [(**I7*m*05** | 8388608) * 256] for the same DAC strobe signal. Note that the conversion may involve the use of different D/A converters that require different DAC strobe signals. Check the Power PMAC manuals for your particular configuration.

- Set **Gate3[*i*].AdcEncStrobe**, **AdcEncDelay**, **AdcEncUtoS**, and **AdcEncHeaderBits** to the settings specified in the Hardware Reference Manual for your Power PMAC configuration. (These settings are only important if you are using the ASIC to interface to sinusoidal encoders or resolvers, or for Power Clipper, to the on-board optional ADCs.)

- Set **Gate3[*i*].AdcAmpStrobe**, **AdcAmpDelay**, **AdcAmpUtoS**, and **AdcAmpHeaderBits** to the settings specified in the Hardware Reference Manual for your Power PMAC configuration. (These settings are only important if you are using the ASIC to interface to amplifier current feedback.)

IC Single-Channel Parameters

- Set **Gate3[*i*].Chan[*j*].PwmFreqMult** equal to **I7*m*01** for the same PWM output frequency.
- Set **Gate3[*i*].Chan[*j*].EncCtrl** equal to **I7*mn*0** for the same encoder decode method.
- Set **Gate3[*i*].Chan[*j*].Equ1Ena** equal to **I7*mn*1** for the same compare channel select.
- **I7*mn*2** to **Gate3[*i*].Chan[*j*].CaptCtrl** (for same capture signal select):
    - If **I7*mn*2** = 0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, or 14, set **Gate3[*i*].Chan[*j*].CaptCtrl** equal to **I7*mn*2**.
    - If **I7*mn*2** = 3, set **Gate3[*i*].Chan[*j*].CaptCtrl** to 15.
    - If **I7*mn*2** = 7, set **Gate3[*i*].Chan[*j*].CaptCtrl** to 11.
    - If **I7*mn*2** = 11, set **Gate3[*i*].Chan[*j*].CaptCtrl** to 7.
    - If **I7*mn*2** = 15, set **Gate3[*i*].Chan[*j*].CaptCtrl** to 3.
- Set **Gate3[*i*].Chan[*j*].CaptFlag** equal to **I7*mn*3** for the same capture flag select.
- Set **Gate3[*i*].Chan[*j*].CaptFlagChan** to ***j*** to use the channel's own flags (default).
- Set **Gate3[*i*].Chan[*j*].GatedIndexSel** equal to **I7*mn*4** for the same use of full or gated index.
- Set **Gate3[*i*].Chan[*j*].IndexGateState** equal to **I7*mn*5** for the same gating for index.
- **I7*mn*6** to **Gate3[*i*].Chan[*j*].OutputMode** (for same output signal mode select):
    - If **I7*mn*6** = 0, set **Gate3[*i*].Chan[*j*].OutputMode** = 0.
    - If **I7*mn*6** = 1, set **Gate3[*i*].Chan[*j*].OutputMode** = 7.
    - If **I7*mn*6** = 2, set **Gate3[*i*].Chan[*j*].OutputMode** = 8.
    - If **I7*mn*6** = 3, set **Gate3[*i*].Chan[*j*].OutputMode** = 15.
- **I7*mn*7** to **Gate3[*i*].Chan[*j*].OutputPol** (for same output signal polarity):
    - If **I7*mn*7** = 0, set **Gate3[*i*].Chan[*j*].OutputPol** = 0.
    - If **I7*mn*7** = 1, set **Gate3[*i*].Chan[*j*].OutputPol** = 7.
    - If **I7*mn*7** = 2, set **Gate3[*i*].Chan[*j*].OutputPol** = 8.
    - If **I7*mn*7** = 3, set **Gate3[*i*].Chan[*j*].OutputPol** = 15.

- Set **Gate3[*i*].Chan[*j*].PfmDirPol** equal to **I7*mn*8** for the same PFM direction signal polarity.

- Set **Gate3[*i*].Chan[*j*].PwmDeadTime** equal to (2.5 * **I7*m*04**) for equivalent PWM dead time.

- Set **Gate3[*i*].Chan[*j*].PfmWidth** equal to (1.25 * **I7*m*04**) for equivalent PFM pulse width with PFM clock set to closest (slightly higher) frequency.

- Set **Gate3[*i*].Chan[*j*].PackInData** to 0 to use the simpler "unpacked" ADC input format, with each ADC value in a separate register.

- Set **Gate3[*i*].Chan[*j*].PackOutData** to 0 to use the simpler "unpacked" PWM output format, with each PWM value in a separate register.

Note on Converting Non-Turbo PMAC2 Servo IC I-Variables

A few applications will be converting non-Turbo PMAC2 configurations (e.g. PMAC2-PC/104, PMAC2-PCI) to Power PMAC configurations. The Servo IC setup I-variables for these older PMAC2s are in the I900 range, not the I7000 range. This section shows the correspondence between the non-Turbo PMAC2 I-variable numbers and the equivalent Turbo PMAC2 I-variable numbers. These Turbo PMAC2 variable numbers can then be mapped to Power PMAC element names as explained above.

Multi-Channel I-Variables

- **I900 I7000**
- **I901 I7001**
- **I902 I7002**
- **I903 I7003**
- **I904 I7004**
- **I905 I7005**
- **I906 I7100**
- **I907 I7103**
- **I908 I7104**
- **I909 I7105**

Single-Channel I-variables (***n*** = 1 to 4, ***x*** = 0 to 9)

- **I9*nx* I70*nx***

Single-Channel I-variables (***n*** = 5 to 8, ***x*** = 0 to 9)

- **I9*nx* I71*n'x*** (***n'*** = ***n*** – 4)

Note on Converting Non-Turbo and Turbo PMAC(1) Servo IC I-Variables

A few applications will be converting non-Turbo PMAC(1) or Turbo PMAC(1) configurations (e.g. PMAC-PCI, Turbo PMAC-PCI) to Power PMAC configurations. The old DSPGATE(0) Servo ICs for the PMAC(1) family of controllers are not supported by any Power PMAC systems, so users will have to convert to newer-generation ICs.

Some settings in the old PMAC(1) controllers are made by hardware jumper ("E-point") settings. These have been replaced by software variables settings in the newer-generation ICs.

The following table lists these functions and how they are set in the non-Turbo and Turbo PMAC(1) controllers, along the matching Turbo PMAC2 variables. These Turbo PMAC2 variable numbers can then be mapped to Power PMAC element names as explained above.

Function	Non-Turbo PMAC(1)	Turbo PMAC(1)	Turbo PMAC2
Phase Clock Freq	E29 – E33	E29 – E33	**I7*m*00**, **I7*m*01**
Servo Clock Freq	E3 – E6	E3 – E6	**I7*m*02**
Encoder Clock Freq	E34 – E38	E34 – E38	**I7*m*03**
DAC/ADC Clock Freq	E98	E98	**I7*m*03**
Encoder Decode Control	**I900** + (5****n***)	**I7*mn*0**	**I7*mn*0**
Capture Control	**I902** + (5****n***)	**I7*mn*2**	**I7*mn*2**
Capture Flag Select	**I903** + (5****n***)	**I7*mn*3**	**I7*mn*3**

Turbo PMAC2 DSPGATE2 MACRO IC I-Variables to Power PMAC DSPGATE2 MACRO IC Elements

This conversion is necessary for Turbo PMAC2 Ultralite or Turbo UMAC ACC-5E to Power PMAC ACC-5E. These Turbo PMAC I-variables are in the range of I6800 – I6999, so the hundreds digit ***m*** is either 8 or 9.

- Power PMAC ASIC index "***i***" equals (2 * [***m*** – 8]) in Turbo PMAC I-variable number if the last two digits are between 00 and 49.

- Power PMAC ASIC index "***i***" equals (2 * [***m*** – 8] + 1) in Turbo PMAC I-variable number if the last two digits are between 50 and 99.

The data structure name shown here is the generic **Gate2[*i*]**. You can also use the hardware-specific alias name of **Acc5E[*i*]**.

Multi-Channel Parameters

- Set **Gate2[*i*].PwmPeriod** equal to **I6*m*00** or **I6*m*50** for the same PWM and MaxPhase frequency.

- Set **Gate2[*i*].PhaseClockDiv** equal to **I6*m*01** or **I6*m*51** for the same division from MaxPhase to Phase frequency.

- Set **Gate2[*i*].ServoClockDiv** equal to **I6*m*02** or **I6*m*52** for the same division from Phase to Servo frequency.

- Set **Gate2[*i*].PhaseServoDir** equal to **I6*m*07** or **I6*m*57** for the same direction of the phase and servo clocks. (These will typically be set automatically at power-on/reset.)

- Set **Gate2[*i*].MacroMode** equal to **I6*m*40** or **I6*m*90** for the same mode of MACRO operation.

- Set **Gate2[*i*].MacroEnable** equal to **I6*m*41** or **I6*m*91** for the same set of enabled MACRO nodes.

Single-Channel Parameters

If using single-channel I-variables on a DSPGATE2 IC in an ACC-5E in a Turbo UMAC application, as for a handwheel encoder input, these can be converted to **Gate2[*i*].Chan[*j*]** data structure elements in a Power PMAC with ACC-5E in the same manner as the **Gate1[*i*].Chan[*j*]** elements in an ACC-24E2A, explained in the section above.

If using single-channel I-variables on a DSPGATE2 IC in a Turbo Clipper application, as for a handwheel encoder input, these functions will have to be moved to auxiliary functions of the main **Gate3[*i*].Chan[*j*]** data structure on the Power Clipper.

Turbo PMAC2 DSPGATE2 MACRO IC I-Variables to Power PMAC DSPGATE3 Elements

This conversion is necessary for Turbo PMAC2 Ultralite or Turbo UMAC ACC-5E to Power UMAC ACC-5E3 or Power PMAC Etherlite ACC-5EP3. These Turbo PMAC I-variables are in the range of I6800 – I6999, so the hundreds digit ***m*** is either 8 or 9.

Note that one DSPGATE3 ICs can perform the work of two DSPGATE2 ICs with regard to MACRO capacity, having the registers for 32 MACRO nodes (in 16-node banks labeled "A"and "B") compared to 16 MACRO nodes in the DSPGATE2 IC.

- Power PMAC ASIC index "***i***" equals (***m*** – 8) in Turbo PMAC I-variable number.
- Power PMAC ASIC channel index "***j***" equals (***n*** – 1) in Turbo PMAC I-variable number.
- The data structure name shown here is the generic **Gate3[*i*]**. You can also use the hardware-specific alias name of **Acc5E3[*i*]** or **Acc5EP3[*i*]**.

IC Multi-Channel Parameters

- Set **Gate3[*i*].PhaseFreq** equal to [117,964,800 / (2 * **I6*m*00** + 3) / (**I6*m*01** + 1)] for the same Phase clock frequency.
- Set **Gate3[*i*].PhaseClockDiv** and **Gate3[*i*].PhaseClockMult** to 0 for the same operation.
- Set **Gate3[*i*].ServoClockDiv** equal to **I6*m*02** for the same division from Phase to Servo frequency.
- Set **Gate3[*i*].PhaseServoDir** equal to **I6*m*07** for the same direction of the phase and servo clocks. (These will typically be set automatically at power-on/reset.)
- Set **Gate3[*i*].MacroModeA** equal to **I6*m*40** for the same mode of MACRO operation.
- Set **Gate3[*i*].MacroModeB** equal to **I6*m*90** for the same mode of MACRO operation.
- Set **Gate3[*i*].MacroEnableA** equal to **I6*m*41** for the same set of enabled MACRO nodes.
- Set **Gate3[*i*].MacroEnableB** equal to **I6*m*91** for the same set of enabled MACRO nodes.

Turbo PMAC ACC-84*x* Setup Register Values to Power PMAC Acc84*x*[*i*] Saved Setup Elements

In Turbo PMAC, the values in setup registers for the ACC-84S, 84B, and 84E serial encoder interfaces are not in I-variables that can be saved to flash memory and automatically loaded on power-on/reset. Instead, the user assigns values to these registers in the application software (usually in a "power-on" PLC program) at startup, either through M-variables defined to these registers, or through direct write (**W**) commands (e.g. **WX:$07880F,$63000B**).

In Power PMAC, these setup registers are saved setup elements, so once their values are set and saved to flash memory, the loading of the desired values at startup is automatic.

For each ACC-84*x*, there is one multi-channel "control" register, and four single-channel command registers.

Turbo Clipper to Power Clipper

For the first ACC-84S in a Clipper system:

- Set **Acc84S[0].SerialEncCtrl** equal to the setup register at X:$07880F.
- Set **Acc84S[0].Chan[0].SerialEncCmd** equal to the setup register at X:$078800.
- Set **Acc84S[0].Chan[1].SerialEncCmd** equal to the setup register at X:$078804.
- Set **Acc84S[0].Chan[2].SerialEncCmd** equal to the setup register at X:$078808.
- Set **Acc84S[0].Chan[3].SerialEncCmd** equal to the setup register at X:$07880C.

For the second ACC-84S in a Clipper system:

- Set **Acc84S[1].SerialEncCtrl** equal to the setup register at X:$07882F.
- Set **Acc84S[1].Chan[0].SerialEncCmd** equal to the setup register at X:$078820.
- Set **Acc84S[1].Chan[1].SerialEncCmd** equal to the setup register at X:$078824.
- Set **Acc84S[1].Chan[2].SerialEncCmd** equal to the setup register at X:$078828.
- Set **Acc84S[1].Chan[3].SerialEncCmd** equal to the setup register at X:$07882C.

If the serial encoder interface in the Power Clipper is implemented through the built-in DSPGATE3 ASIC(s), the saved setup elements **Clipper[*i*].SerialEncCtrl** and **Clipper[*i*].Chan[*j*].SerialEncCmd** need to be set for the same functionality. The precise values of these 32-bit registers will not be exactly the same as those for the 24-bit ACC-84*x* registers; refer to the Power PMAC Software Reference Manual for details.

Turbo Brick to Power Brick

For the first ACC-84B in a Brick system:

- Set **Acc84B[0].SerialEncCtrl** equal to the setup register at X:$078B2F.
- Set **Acc84B[0].Chan[0].SerialEncCmd** equal to the setup register at X:$078B20.
- Set **Acc84B[0].Chan[1].SerialEncCmd** equal to the setup register at X:$078B24.
- Set **Acc84B[0].Chan[2].SerialEncCmd** equal to the setup register at X:$078B28.
- Set **Acc84B[0].Chan[3].SerialEncCmd** equal to the setup register at X:$078B2C.

For the second ACC-84B in a Brick system:

- Set **Acc84B[1].SerialEncCtrl** equal to the setup register at X:$078B3F.
- Set **Acc84B[1].Chan[0].SerialEncCmd** equal to the setup register at X:$078B30.
- Set **Acc84B[1].Chan[1].SerialEncCmd** equal to the setup register at X:$078B34.
- Set **Acc84B[1].Chan[2].SerialEncCmd** equal to the setup register at X:$078B38.
- Set **Acc84B[1].Chan[3].SerialEncCmd** equal to the setup register at X:$078B3C.

If the serial encoder interface in the Power Brick is implemented through the built-in DSPGATE3 ASIC(s), the saved setup elements **PowerBrick[*i*].SerialEncCtrl** and **PowerBrick[*i*].Chan[*j*].SerialEncCmd** need to be set for the same functionality. The precise values of these 32-bit registers will not be exactly the same as those for the 24-bit ACC-84*x* registers; refer to the Power PMAC Software Reference Manual for details.

Turbo UMAC to Power UMAC

For an ACC-84E in a UMAC system, if the SW1 switches are configured to provide a Turbo PMAC base address of $7*xy*00, where *x* is a hex digit that can take the values 8, 9, A (10), or B (11), and *y* is a hex digit that can take the values C (12), D (13), or E (14), then the index ***i*** of **Acc84E[*i*]** can be computed as:

$$\boldsymbol{i} = 4 * (x - 8) + (y - 12)$$

- Set **Acc84E[*i*].SerialEncCtrl** equal to the setup register at X:$07*xy*0F.
- Set **Acc84S[*i*].Chan[*j*].SerialEncCmd** equal to the setup register at X:$07*xy*0*z*, where *z* is a hex digit that can take the values 0, 4, 8, or C (12), and the channel index ***j*** can be computed as $\boldsymbol{j} = (z / 4)$.

If the serial encoder interface in the Power UMAC is implemented through the built-in DSPGATE3 ASIC(s), the saved setup elements **Acc24E3[*i*].SerialEncCtrl** and **Acc24E3[*i*].Chan[*j*].SerialEncCmd** need to be set for the same functionality. The precise values of these 32-bit registers will not be exactly the same as those for the 24-bit ACC-84*x* registers; refer to the Power PMAC Software Reference Manual for details.

Encoder Conversion Table I-Variable Equivalents

Both Turbo PMAC and Power PMAC controllers use an "encoder conversion table" (ECT) to process feedback and master data for use by the motor algorithms. The structures of the table are very different in the two controllers.

In Turbo PMAC, the ECT entries are defined by 1, 2, or 3 consecutive I-variables, starting at I8000. These are known as 1-line, 2-line, or 3-line entries.

In Power PMAC, each ECT entry is defined by an **EncTable[*n*]** structure of identical form. Each structure has multiple setup elements that define the conversion method, source address[es], and processing variables.

If you are using the common standard processing methods, the translation from Turbo PMAC to Power PMAC is quite straightforward (the default configurations require no work at all). The common cases are shown below. Some of the less common cases (not listed below) will require special attention to translate.

Note that Power PMAC does not require the user to know the numerical value of the address of a register. Instead, the user can just specify the name of the data structure element for the register, followed by the ".a" (address of) suffix.

Digital Incremental (Quadrature) Encoder Conversion

The most common type of feedback used is digital quadrature encoders. PMAC controllers can process these with "1/T" timer based sub-count interpolation. With the older DSPGATE1 ASIC, computing the interpolated position must be done in software (**type** = 3). With the newer DSPGATE3 IC, this is done in hardware in the ASIC, and the result can simply be read out of a single register (**type** = 1).

In most hardware configurations, Power PMAC will set up the default ECT for this process from each detected ASIC.

Turbo Clipper to Power Clipper

Turbo Clipper Incremental Encoder Conversion to Power Clipper

Turbo Clipper **I8*xxx***	Power Clipper **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$078000	**Clipper[0].Chan[0].ServoCapt.a**
$078008	**Clipper[0].Chan[1].ServoCapt.a**
$078010	**Clipper[0].Chan[2].ServoCapt.a**
$078018	**Clipper[0].Chan[3].ServoCapt.a**
$078100	**Clipper[1].Chan[0].ServoCapt.a**
$078108	**Clipper[1].Chan[1].ServoCapt.a**
$078110	**Clipper[1].Chan[2].ServoCapt.a**
$078118	**Clipper[1].Chan[3].ServoCapt.a**

Notes:

1. If Turbo Clipper uses default configuration, Power Clipper can use default configuration.
2. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*

EncTable[*n*].index1 .. **index6** = 0 // Default
EncTable[*n*].ScaleFactor = 1/256 // For result in whole quadrature counts

Turbo Brick to Power Brick

Turbo Brick Incremental Encoder Conversion to Power Brick

Turbo Brick **I8*xxx***	Power Clipper **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$078000	**PowerBrick[0].Chan[0].ServoCapt.a**
$078008	**PowerBrick[0].Chan[1].ServoCapt.a**
$078010	**PowerBrick[0].Chan[2].ServoCapt.a**
$078018	**PowerBrick[0].Chan[3].ServoCapt.a**
$078100	**PowerBrick[1].Chan[0].ServoCapt.a**
$078108	**PowerBrick[1].Chan[1].ServoCapt.a**
$078110	**PowerBrick[1].Chan[2].ServoCapt.a**
$078118	**PowerBrick[1].Chan[3].ServoCapt.a**

Notes:

1. If Turbo Brick uses default configuration, Power Brick can use default configuration.
2. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 .. **index6** = 0 // Default
EncTable[*n*].ScaleFactor = 1/256 // For result in whole quadrature counts

Turbo UMAC to Power UMAC ACC-24E2*x*

Turbo UMAC ACC-24E2*x* Incremental Encoder Conversion to Power UMAC ACC-24E2*x*

Turbo UMAC **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for ACC-24E2*x* Incremental Encoder **EncTable[*n*].type** = 3	Power UMAC **EncTable[*n*].pEnc1** for ACC-24E2*x* Incremental Encoder **EncTable[*n*].type** = 3
$078200	**Acc24E2*x*[4].Chan[0].ServoCapt.a**	**Acc24E2*x*[4].Chan[0].TimeBetweenCts.a**
$078208	**Acc24E2*x*[4].Chan[1].ServoCapt.a**	**Acc24E2*x*[4].Chan[1].TimeBetweenCts.a**
$078210	**Acc24E2*x*[4].Chan[2].ServoCapt.a**	**Acc24E2*x*[4].Chan[2].TimeBetweenCts.a**
$078218	**Acc24E2*x*[4].Chan[3].ServoCapt.a**	**Acc24E2*x*[4].Chan[3].TimeBetweenCts.a**
$078300	**Acc24E2*x*[6].Chan[0].ServoCapt.a**	**Acc24E2*x*[6].Chan[0].TimeBetweenCts.a**
$078308	**Acc24E2*x*[6].Chan[1].ServoCapt.a**	**Acc24E2*x*[6].Chan[1].TimeBetweenCts.a**
$078310	**Acc24E2*x*[6].Chan[2].ServoCapt.a**	**Acc24E2*x*[6].Chan[2].TimeBetweenCts.a**
$078318	**Acc24E2*x*[6].Chan[3].ServoCapt.a**	**Acc24E2*x*[6].Chan[3].TimeBetweenCts.a**
$079200	**Acc24E2*x*[8].Chan[0].ServoCapt.a**	**Acc24E2*x*[8].Chan[0].TimeBetweenCts.a**
$079208	**Acc24E2*x*[8].Chan[1].ServoCapt.a**	**Acc24E2*x*[8].Chan[1].TimeBetweenCts.a**
$079210	**Acc24E2*x*[8].Chan[2].ServoCapt.a**	**Acc24E2*x*[8].Chan[2].TimeBetweenCts.a**
$079218	**Acc24E2*x*[8].Chan[3].ServoCapt.a**	**Acc24E2*x*[8].Chan[3].TimeBetweenCts.a**
$079300	**Acc24E2*x*[10].Chan[0].ServoCapt.a**	**Acc24E2*x*[10].Chan[0].TimeBetweenCts.a**
$079308	**Acc24E2*x*[10].Chan[1].ServoCapt.a**	**Acc24E2*x*[10].Chan[1].TimeBetweenCts.a**
$079310	**Acc24E2*x*[10].Chan[2].ServoCapt.a**	**Acc24E2*x*[10].Chan[2].TimeBetweenCts.a**
$079318	**Acc24E2*x*[10].Chan[3].ServoCapt.a**	**Acc24E2*x*[10].Chan[3].TimeBetweenCts.a**
$07A200	**Acc24E2*x*[12].Chan[0].ServoCapt.a**	**Acc24E2*x*[12].Chan[0].TimeBetweenCts.a**
$07A208	**Acc24E2*x*[12].Chan[1].ServoCapt.a**	**Acc24E2*x*[12].Chan[1].TimeBetweenCts.a**
$07A210	**Acc24E2*x*[12].Chan[2].ServoCapt.a**	**Acc24E2*x*[12].Chan[2].TimeBetweenCts.a**
$07A218	**Acc24E2*x*[12].Chan[3].ServoCapt.a**	**Acc24E2*x*[12].Chan[3].TimeBetweenCts.a**
$07A300	**Acc24E2*x*[14].Chan[0].ServoCapt.a**	**Acc24E2*x*[14].Chan[0].TimeBetweenCts.a**
$07A308	**Acc24E2*x*[14].Chan[1].ServoCapt.a**	**Acc24E2*x*[14].Chan[1].TimeBetweenCts.a**
$07A310	**Acc24E2*x*[14].Chan[2].ServoCapt.a**	**Acc24E2*x*[14].Chan[2].TimeBetweenCts.a**
$07A318	**Acc24E2*x*[14].Chan[3].ServoCapt.a**	**Acc24E2*x*[14].Chan[3].TimeBetweenCts.a**
$07B200	**Acc24E2*x*[16].Chan[0].ServoCapt.a**	**Acc24E2*x*[16].Chan[0].TimeBetweenCts.a**
$07B208	**Acc24E2*x*[16].Chan[1].ServoCapt.a**	**Acc24E2*x*[16].Chan[1].TimeBetweenCts.a**
$07B210	**Acc24E2*x*[16].Chan[2].ServoCapt.a**	**Acc24E2*x*[16].Chan[2].TimeBetweenCts.a**
$07B218	**Acc24E2*x*[16].Chan[3].ServoCapt.a**	**Acc24E2*x*[16].Chan[3].TimeBetweenCts.a**
$07B300	**Acc24E2*x*[18].Chan[0].ServoCapt.a**	**Acc24E2*x*[18].Chan[0].TimeBetweenCts.a**
$07B308	**Acc24E2*x*[18].Chan[1].ServoCapt.a**	**Acc24E2*x*[18].Chan[1].TimeBetweenCts.a**
$07B310	**Acc24E2*x*[18].Chan[2].ServoCapt.a**	**Acc24E2*x*[18].Chan[2].TimeBetweenCts.a**
$07B318	**Acc24E2*x*[18].Chan[3].ServoCapt.a**	**Acc24E2*x*[18].Chan[3].TimeBetweenCts.a**

Notes:

1. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.
2. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
3. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.

EncTable[*n*].type = 3
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{from table}*
EncTable[*n*].index1 .. **index6** = 0 // Default
EncTable[*n*].ScaleFactor = 1/512 // For result in whole quadrature counts

Turbo UMAC to Power UMAC ACC-24E3 or CK3M with CK3W-AX*nnn*

Turbo UMAC Incremental Encoder Conversion to Power UMAC ACC-24E3 Axis Interface or CK3M with CK3W-AX*nnnn* Axis Interface

Turbo UMAC **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for ACC-24E3 Incremental Encoder **EncTable[*n*].type** = 1	CK3M **EncTable[*n*].pEnc** for CK3W-AX*nnnn* Incremental Encoder **EncTable[*n*].type** = 1
$078200	**Acc24E3[0].Chan[0].ServoCapt.a**	**CK3WAX[0].Chan[0].ServoCapt.a**
$078208	**Acc24E3[0].Chan[1].ServoCapt.a**	**CK3WAX[0].Chan[1].ServoCapt.a**
$078210	**Acc24E3[0].Chan[2].ServoCapt.a**	**CK3WAX[0].Chan[2].ServoCapt.a**
$078218	**Acc24E3[0].Chan[3].ServoCapt.a**	**CK3WAX[0].Chan[3].ServoCapt.a**
$078300	**Acc24E3[1].Chan[0].ServoCapt.a**	**CK3WAX[1].Chan[0].ServoCapt.a**
$078308	**Acc24E3[1].Chan[1].ServoCapt.a**	**CK3WAX[1].Chan[1].ServoCapt.a**
$078310	**Acc24E3[1].Chan[2].ServoCapt.a**	**CK3WAX[1].Chan[2].ServoCapt.a**
$078318	**Acc24E3[1].Chan[3].ServoCapt.a**	**CK3WAX[1].Chan[3].ServoCapt.a**
$079200	**Acc24E3[2].Chan[0].ServoCapt.a**	**CK3WAX[2].Chan[0].ServoCapt.a**
$079208	**Acc24E3[2].Chan[1].ServoCapt.a**	**CK3WAX[2].Chan[1].ServoCapt.a**
$079210	**Acc24E3[2].Chan[2].ServoCapt.a**	**CK3WAX[2].Chan[2].ServoCapt.a**
$079218	**Acc24E3[2].Chan[3].ServoCapt.a**	**CK3WAX[2].Chan[3].ServoCapt.a**
$079300	**Acc24E3[3].Chan[0].ServoCapt.a**	**CK3WAX[3].Chan[0].ServoCapt.a**
$079308	**Acc24E3[3].Chan[1].ServoCapt.a**	**CK3WAX[3].Chan[1].ServoCapt.a**
$079310	**Acc24E3[3].Chan[2].ServoCapt.a**	**CK3WAX[3].Chan[2].ServoCapt.a**
$079318	**Acc24E3[3].Chan[3].ServoCapt.a**	**CK3WAX[3].Chan[3].ServoCapt.a**
$07A200	**Acc24E3[4].Chan[0].ServoCapt.a**	**CK3WAX[4].Chan[0].ServoCapt.a**
$07A208	**Acc24E3[4].Chan[1].ServoCapt.a**	**CK3WAX[4].Chan[1].ServoCapt.a**
$07A210	**Acc24E3[4].Chan[2].ServoCapt.a**	**CK3WAX[4].Chan[2].ServoCapt.a**
$07A218	**Acc24E3[4].Chan[3].ServoCapt.a**	**CK3WAX[4].Chan[3].ServoCapt.a**
$07A300	**Acc24E3[5].Chan[0].ServoCapt.a**	**CK3WAX[5].Chan[0].ServoCapt.a**
$07A308	**Acc24E3[5].Chan[1].ServoCapt.a**	**CK3WAX[5].Chan[1].ServoCapt.a**
$07A310	**Acc24E3[5].Chan[2].ServoCapt.a**	**CK3WAX[5].Chan[2].ServoCapt.a**
$07A318	**Acc24E3[5].Chan[3].ServoCapt.a**	**CK3WAX[5].Chan[3].ServoCapt.a**
$07B200	**Acc24E3[6].Chan[0].ServoCapt.a**	**CK3WAX[6].Chan[0].ServoCapt.a**
$07B208	**Acc24E3[6].Chan[1].ServoCapt.a**	**CK3WAX[6].Chan[1].ServoCapt.a**
$07B210	**Acc24E3[6].Chan[2].ServoCapt.a**	**CK3WAX[6].Chan[2].ServoCapt.a**
$07B218	**Acc24E3[6].Chan[3].ServoCapt.a**	**CK3WAX[6].Chan[3].ServoCapt.a**
$07B300	**Acc24E3[7].Chan[0].ServoCapt.a**	**CK3WAX[7].Chan[0].ServoCapt.a**
$07B308	**Acc24E3[7].Chan[1].ServoCapt.a**	**CK3WAX[7].Chan[1].ServoCapt.a**
$07B310	**Acc24E3[7].Chan[2].ServoCapt.a**	**CK3WAX[7].Chan[2].ServoCapt.a**
$07B318	**Acc24E3[7].Chan[3].ServoCapt.a**	**CK3WAX[7].Chan[3].ServoCapt.a**

Notes:

1. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.
2. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
3. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 .. **index6** = 0 // Default
EncTable[*n*].ScaleFactor = 1/256 // For result in whole quadrature coutns

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Node 0 Feedback, No Change Limiting, to Power PMAC

Turbo PMAC **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for ACC-5E Node 0 Feedback	Power PMAC **EncTable[*n*].pEnc** for ACC-5E3 Node 0 Feedback
$2F8420	**Acc5E[0].Macro[0][0].a**	**Acc5E3[0].MacroInA[0][0].a**
$2F8424	**Acc5E[0].Macro[1][0].a**	**Acc5E3[0].MacroInA[1][0].a**
$2F8428	**Acc5E[0].Macro[4][0].a**	**Acc5E3[0].MacroInA[4][0].a**
$2F842C	**Acc5E[0].Macro[5][0].a**	**Acc5E3[0].MacroInA[5][0].a**
$2F8430	**Acc5E[0].Macro[8][0].a**	**Acc5E3[0].MacroInA[8][0].a**
$2F8434	**Acc5E[0].Macro[9][0].a**	**Acc5E3[0].MacroInA[9][0].a**
$2F8438	**Acc5E[0].Macro[12][0].a**	**Acc5E3[0].MacroInA[12][0].a**
$2F843C	**Acc5E[0].Macro[13][0].a**	**Acc5E3[0].MacroInA[13][0].a**
$2F9420	**Acc5E[1].Macro[0][0].a**	**Acc5E3[0].MacroInB[0][0].a**
$2F9424	**Acc5E[1].Macro[1][0].a**	**Acc5E3[0].MacroInB[1][0].a**
$2F9428	**Acc5E[1].Macro[4][0].a**	**Acc5E3[0].MacroInB[4][0].a**
$2F942C	**Acc5E[1].Macro[5][0].a**	**Acc5E3[0].MacroInB[5][0].a**
$2F9430	**Acc5E[1].Macro[8][0].a**	**Acc5E3[0].MacroInB[8][0].a**
$2F9434	**Acc5E[1].Macro[9][0].a**	**Acc5E3[0].MacroInB[9][0].a**
$2F9438	**Acc5E[1].Macro[12][0].a**	**Acc5E3[0].MacroInB[12][0].a**
$2F943C	**Acc5E[1].Macro[13][0].a**	**Acc5E3[0].MacroInB[13][0].a**
$2FA420	**Acc5E[2].Macro[0][0].a**	**Acc5E3[1].MacroInA[0][0].a**
$2FA424	**Acc5E[2].Macro[1][0].a**	**Acc5E3[1].MacroInA[1][0].a**
$2FA428	**Acc5E[2].Macro[4][0].a**	**Acc5E3[1].MacroInA[4][0].a**
$2FA42C	**Acc5E[2].Macro[5][0].a**	**Acc5E3[1].MacroInA[5][0].a**
$2FA430	**Acc5E[2].Macro[8][0].a**	**Acc5E3[1].MacroInA[8][0].a**
$2FA434	**Acc5E[2].Macro[9][0].a**	**Acc5E3[1].MacroInA[9][0].a**
$2FA438	**Acc5E[2].Macro[12][0].a**	**Acc5E3[1].MacroInA[12][0].a**
$2FA43C	**Acc5E[2].Macro[13][0].a**	**Acc5E3[1].MacroInA[13][0].a**
$2FB420	**Acc5E[3].Macro[0][0].a**	**Acc5E3[1].MacroInB[0][0].a**
$2FB424	**Acc5E[3].Macro[1][0].a**	**Acc5E3[1].MacroInB[1][0].a**
$2FB428	**Acc5E[3].Macro[4][0].a**	**Acc5E3[1].MacroInB[4][0].a**
$2FB42C	**Acc5E[3].Macro[5][0].a**	**Acc5E3[1].MacroInB[5][0].a**
$2FB430	**Acc5E[3].Macro[8][0].a**	**Acc5E3[1].MacroInB[8][0].a**
$2FB434	**Acc5E[3].Macro[9][0].a**	**Acc5E3[1].MacroInB[9][0].a**
$2FB438	**Acc5E[3].Macro[12][0].a**	**Acc5E3[1].MacroInB[12][0].a**
$2FB43C	**Acc5E[3].Macro[13][0].a**	**Acc5E3[1].MacroInB[13][0].a**

Notes:

1. If Turbo PMAC uses default configuration, Power PMAC can use default configuration.
2. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
3. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.
4. The first digit of "2" in the Turbo PMAC line can also be "3" if the entry is change-limited

EncTable[*n*].type = 1
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 8
EncTable[*n*].index2 = 8
EncTable[*n*].index3 .. **index6** = 0
EncTable[*n*].MaxDelta = 0 (no change limiting)
EncTable[*n*].MaxDelta = *{third-line value from Turbo entry}* (to specify change limiting)
EncTable[*n*].ScaleFactor = 1/8192

Interpolated Sinusoidal Encoder Conversion

With the proper accessories, PMAC controllers can perform high-resolution interpolation of sinusoidal encoders. With the older DSPGATE1 ASIC, computing the interpolated position must be done in software (**type** = 4). With the newer DSPGATE3 IC, depending on the accessory used, this can done in hardware in the ASIC, and the result can simply be read out of a single register (**type** = 1), or it must be done in software (**type** = 4).

Turbo Clipper to Power Clipper

Turbo Clipper with ACC-51S to Power Clipper with ACC-51S

Turbo Clipper First Line **I8*xxx***	Power Clipper **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 4
$FF8000	**Clipper[0].Chan[0].Status.a**
$FF8008	**Clipper[0].Chan[1].Status.a**
$FF8010	**Clipper[0].Chan[2].Status.a**
$FF8018	**Clipper[0].Chan[3].Status.a**
$FF8100	**Clipper[1].Chan[0].Status.a**
$FF8108	**Clipper[1].Chan[1].Status.a**
$FF8110	**Clipper[1].Chan[2].Status.a**
$FF8118	**Clipper[1].Chan[3].Status.a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

Turbo Clipper with ACC-51S to Power Clipper with ACC-51S

Turbo Clipper Second Line **I8*xxx***	Power Clipper **EncTable[*n*].pEnc1** for **EncTable[*n*].type** = 4
$078B00	**Sys.piom** + $000000
$078B02	**Sys.piom** + $000002
$078B04	**Sys.piom** + $000004
$078B06	**Sys.piom** + $000006
$078B08	**Sys.piom** + $000008
$078B0A	**Sys.piom** + $00000A
$078B0C	**Sys.piom** + $00000C
$078B0E	**Sys.piom** + $00000E

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

EncTable[*n*].type = 4
EncTable[*n*].pEnc = *{from first table}*
EncTable[*n*].pEnc1 = *{from second table}*
EncTable[*n*].index1 .. **index4** = 0 // Can low-pass filter with **index1** and **index2**
EncTable[*n*].index5 = 1 // DSPGATE3 ASIC
EncTable[*n*].index6 = 0
EncTable[*n*].ScaleFactor = 1/128 // For same 1/32-quad-count resolution as Turbo

Turbo Brick to Power Brick

Turbo Brick with ACC-51B to Power Brick with Interpolator

Turbo Brick First Line **I8*xxx***	Power Brick **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 4
$FF8000	**PowerBrick[0].Chan[0].Status.a**
$FF8008	**PowerBrick[0].Chan[1].Status.a**
$FF8010	**PowerBrick[0].Chan[2].Status.a**
$FF8018	**PowerBrick[0].Chan[3].Status.a**
$FF8100	**PowerBrick[1].Chan[0].Status.a**
$FF8108	**PowerBrick[1].Chan[1].Status.a**
$FF8110	**PowerBrick[1].Chan[2].Status.a**
$FF8118	**PowerBrick[1].Chan[3].Status.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

Turbo Brick with ACC-51B to Power Brick with Interpolator

Turbo Brick Second Line **I8*xxx***	Power Brick **EncTable[*n*].pEnc1** for **EncTable[*n*].type** = 4
$078800	**PowerBrick[0].Chan[0].AdcEnc[0].a**
$078802	**PowerBrick[0].Chan[1].AdcEnc[0].a**
$078804	**PowerBrick[0].Chan[2].AdcEnc[0].a**
$078806	**PowerBrick[0].Chan[3].AdcEnc[0].a**
$078808	**PowerBrick[1].Chan[0].AdcEnc[0].a**
$07880A	**PowerBrick[1].Chan[1].AdcEnc[0].a**
$07880C	**PowerBrick[1].Chan[2].AdcEnc[0].a**
$07880E	**PowerBrick[1].Chan[3].AdcEnc[0].a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

EncTable[*n*].type = 4
EncTable[*n*].pEnc = *{from first table}*
EncTable[*n*].pEnc1 = *{from second table}*
EncTable[*n*].index1 .. **index4** = 0 // Can low-pass filter with **index1** and **index2**
EncTable[*n*].index5 = 1 // DSPGATE3 ASIC
EncTable[*n*].index6 = 0
EncTable[*n*].ScaleFactor = 1/128 // For same 1/32-quad-count resolution as Turbo

Hardware interpolation in the DSPGATE3 is also possible in the Power Brick with interpolation circuitry if **PowerBrick[*i*].Chan[*j*].AtanEna** is set to 1. In this case:

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **PowerBrick[*i*].Chan[*j*].ServoCapt.a**
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 .. **index6** = 0 // Can low-pass filter with **index1** and **index2**
EncTable[*n*].ScaleFactor = 1/128 // For same 1/32-quad-count resolution as Turbo

Turbo UMAC to Power UMAC with ACC-51E Interpolator

Turbo UMAC with ACC-51E to Power UMAC with ACC-51E

Turbo UMAC First Line **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 4	Turbo UMAC First Line **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 4
$FF8200	**Acc51E[4].Chan[0].Status.a**	$FFA200	**Acc51E[12].Chan[0].Status.a**
$FF8208	**Acc51E[4].Chan[1].Status.a**	$FFA208	**Acc51E[12].Chan[1].Status.a**
$FF8210	**Acc51E[4].Chan[2].Status.a**	$FFA210	**Acc51E[12].Chan[2].Status.a**
$FF8218	**Acc51E[4].Chan[3].Status.a**	$FFA218	**Acc51E[12].Chan[3].Status.a**
$FF8300	**Acc51E[6].Chan[0].Status.a**	$FFA300	**Acc51E[14].Chan[0].Status.a**
$FF8308	**Acc51E[6].Chan[1].Status.a**	$FFA308	**Acc51E[14].Chan[1].Status.a**
$FF8310	**Acc51E[6].Chan[2].Status.a**	$FFA310	**Acc51E[14].Chan[2].Status.a**
$FF8318	**Acc51E[6].Chan[3].Status.a**	$FFA318	**Acc51E[14].Chan[3].Status.a**
$FF9200	**Acc51E[8].Chan[0].Status.a**	$FFB200	**Acc51E[16].Chan[0].Status.a**
$FF9208	**Acc51E[8].Chan[1].Status.a**	$FFB208	**Acc51E[16].Chan[1].Status.a**
$FF9210	**Acc51E[8].Chan[2].Status.a**	$FFB210	**Acc51E[16].Chan[2].Status.a**
$FF9218	**Acc51E[8].Chan[3].Status.a**	$FFB218	**Acc51E[16].Chan[3].Status.a**
$FF9300	**Acc51E[10].Chan[0].Status.a**	$FFB300	**Acc51E[18].Chan[0].Status.a**
$FF9308	**Acc51E[10].Chan[1].Status.a**	$FFB308	**Acc51E[18].Chan[1].Status.a**
$FF9310	**Acc51E[10].Chan[2].Status.a**	$FFB310	**Acc51E[18].Chan[2].Status.a**
$FF9318	**Acc51E[10].Chan[3].Status.a**	$FFB318	**Acc51E[18].Chan[3].Status.a**

Turbo UMAC with ACC-51E to Power UMAC with ACC-51E

Turbo UMAC Second Line **I8*xxx***	Power UMAC **EncTable[*n*].pEnc1** for **EncTable[*n*].type** = 4	Turbo UMAC Second Line **I8*xxx***	Power UMAC **EncTable[*n*].pEnc1** for **EncTable[*n*].type** = 4
$078205	**Acc51E[4].Chan[0].Adc[0].a**	$07A205	**Acc51E[12].Chan[0].Adc[0].a**
$07820D	**Acc51E[4].Chan[1].Adc[0].a**	$07A20D	**Acc51E[12].Chan[1].Adc[0].a**
$078215	**Acc51E[4].Chan[2].Adc[0].a**	$07A215	**Acc51E[12].Chan[2].Adc[0].a**
$07821D	**Acc51E[4].Chan[3].Adc[0].a**	$07A21D	**Acc51E[12].Chan[3].Adc[0].a**
$078305	**Acc51E[6].Chan[0].Adc[0].a**	$07A305	**Acc51E[14].Chan[0].Adc[0].a**
$07830D	**Acc51E[6].Chan[1].Adc[0].a**	$07A30D	**Acc51E[14].Chan[1].Adc[0].a**
$078315	**Acc51E[6].Chan[2].Adc[0].a**	$07A315	**Acc51E[14].Chan[2].Adc[0].a**
$07831D	**Acc51E[6].Chan[3].Adc[0].a**	$07A31D	**Acc51E[14].Chan[3].Adc[0].a**
$0798205	**Acc51E[8].Chan[0].Adc[0].a**	$07B8205	**Acc51E[16].Chan[0].Adc[0].a**
$07920D	**Acc51E[8].Chan[1].Adc[0].a**	$07B20D	**Acc51E[16].Chan[1].Adc[0].a**
$079215	**Acc51E[8].Chan[2].Adc[0].a**	$07B215	**Acc51E[16].Chan[2].Adc[0].a**
$07921D	**Acc51E[8].Chan[3].Adc[0].a**	$07B21D	**Acc51E[16].Chan[3].Adc[0].a**
$079305	**Acc51E[10].Chan[0].Adc[0].a**	$07B305	**Acc51E[18].Chan[0].Adc[0].a**
$07930D	**Acc51E[10].Chan[1].Adc[0].a**	$07B30D	**Acc51E[18].Chan[1].Adc[0].a**
$079315	**Acc51E[10].Chan[2].Adc[0].a**	$07B315	**Acc51E[18].Chan[2].Adc[0].a**
$07931D	**Acc51E[10].Chan[3].Adc[0].a**	$07B31D	**Acc51E[18].Chan[3].Adc[0].a**

Notes:

1. **Acc51E[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc51E[*i*]**.

EncTable[*n*].type = 4
EncTable[*n*].pEnc = *{from first table}*
EncTable[*n*].pEnc1 = *{from second table}*
EncTable[*n*].index1 .. **index6** = 0 // Can low-pass filter with **index1** and **index2**
EncTable[*n*].ScaleFactor = 1/1024

Turbo UMAC to Power UMAC ACC-24E3 Interpolator or CK3M with CK3W-AX*nnnn* Interpolator

Turbo UMAC ACC-51E Sine Encoder conversion to Power UMAC ACC-24E3 Axis Interface or CK3M with CK3W-AX*nnnn* Axis Interface

Turbo UMAC **I8*xxx***	Power UMAC **EncTable[*n*].pEnc** for ACC-24E3 Sine Encoder Interpolator **EncTable[*n*].type** = 1	CK3M **EncTable[*n*].pEnc** for CK3W-AX*nnnn* Sine Encoder Interpolator **EncTable[*n*].type** = 1
$FF8200	**Acc24E3[0].Chan[0].ServoCapt.a**	**CK3WAX[0].Chan[0].ServoCapt.a**
$FF8208	**Acc24E3[0].Chan[1].ServoCapt.a**	**CK3WAX[0].Chan[1].ServoCapt.a**
$FF8210	**Acc24E3[0].Chan[2].ServoCapt.a**	**CK3WAX[0].Chan[2].ServoCapt.a**
$FF8218	**Acc24E3[0].Chan[3].ServoCapt.a**	**CK3WAX[0].Chan[3].ServoCapt.a**
$FF8300	**Acc24E3[1].Chan[0].ServoCapt.a**	**CK3WAX[1].Chan[0].ServoCapt.a**
$FF8308	**Acc24E3[1].Chan[1].ServoCapt.a**	**CK3WAX[1].Chan[1].ServoCapt.a**
$FF8310	**Acc24E3[1].Chan[2].ServoCapt.a**	**CK3WAX[1].Chan[2].ServoCapt.a**
$FF8318	**Acc24E3[1].Chan[3].ServoCapt.a**	**CK3WAX[1].Chan[3].ServoCapt.a**
$FF9200	**Acc24E3[2].Chan[0].ServoCapt.a**	**CK3WAX[2].Chan[0].ServoCapt.a**
$FF9208	**Acc24E3[2].Chan[1].ServoCapt.a**	**CK3WAX[2].Chan[1].ServoCapt.a**
$FF9210	**Acc24E3[2].Chan[2].ServoCapt.a**	**CK3WAX[2].Chan[2].ServoCapt.a**
$FF9218	**Acc24E3[2].Chan[3].ServoCapt.a**	**CK3WAX[2].Chan[3].ServoCapt.a**
$FF9300	**Acc24E3[3].Chan[0].ServoCapt.a**	**CK3WAX[3].Chan[0].ServoCapt.a**
$FF9308	**Acc24E3[3].Chan[1].ServoCapt.a**	**CK3WAX[3].Chan[1].ServoCapt.a**
$FF9310	**Acc24E3[3].Chan[2].ServoCapt.a**	**CK3WAX[3].Chan[2].ServoCapt.a**
$FF9318	**Acc24E3[3].Chan[3].ServoCapt.a**	**CK3WAX[3].Chan[3].ServoCapt.a**
$FFA200	**Acc24E3[4].Chan[0].ServoCapt.a**	**CK3WAX[4].Chan[0].ServoCapt.a**
$FFA208	**Acc24E3[4].Chan[1].ServoCapt.a**	**CK3WAX[4].Chan[1].ServoCapt.a**
$FFA210	**Acc24E3[4].Chan[2].ServoCapt.a**	**CK3WAX[4].Chan[2].ServoCapt.a**
$FFA218	**Acc24E3[4].Chan[3].ServoCapt.a**	**CK3WAX[4].Chan[3].ServoCapt.a**
$FFA300	**Acc24E3[5].Chan[0].ServoCapt.a**	**CK3WAX[5].Chan[0].ServoCapt.a**
$FFA308	**Acc24E3[5].Chan[1].ServoCapt.a**	**CK3WAX[5].Chan[1].ServoCapt.a**
$FFA310	**Acc24E3[5].Chan[2].ServoCapt.a**	**CK3WAX[5].Chan[2].ServoCapt.a**
$FFA318	**Acc24E3[5].Chan[3].ServoCapt.a**	**CK3WAX[5].Chan[3].ServoCapt.a**
$FFB200	**Acc24E3[6].Chan[0].ServoCapt.a**	**CK3WAX[6].Chan[0].ServoCapt.a**
$FFB208	**Acc24E3[6].Chan[1].ServoCapt.a**	**CK3WAX[6].Chan[1].ServoCapt.a**
$FFB210	**Acc24E3[6].Chan[2].ServoCapt.a**	**CK3WAX[6].Chan[2].ServoCapt.a**
$FFB218	**Acc24E3[6].Chan[3].ServoCapt.a**	**CK3WAX[6].Chan[3].ServoCapt.a**
$FFB300	**Acc24E3[7].Chan[0].ServoCapt.a**	**CK3WAX[7].Chan[0].ServoCapt.a**
$FFB308	**Acc24E3[7].Chan[1].ServoCapt.a**	**CK3WAX[7].Chan[1].ServoCapt.a**
$FFB310	**Acc24E3[7].Chan[2].ServoCapt.a**	**CK3WAX[7].Chan[2].ServoCapt.a**
$FFB318	**Acc24E3[7].Chan[3].ServoCapt.a**	**CK3WAX[7].Chan[3].ServoCapt.a**

Notes:

1. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
2. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 .. **index6** = 0 // Can low-pass filter with **index1** and **index2**
EncTable[*n*].ScaleFactor = 1/128 // For same 1/32-quad-count resolution as Turbo

Serial Encoder Conversion

Turbo PMAC controllers require an accessory card, usually an ACC-84*x*, to process serial encoder feedback. Power PMAC controllers *can* also use an ACC-84*x* card to do this, but can use the built-in serial-encoder interface in the DSPGATE3 IC for many protocols.

Turbo Clipper to Power Clipper

Turbo Clipper with ACC-84S to Power Clipper

Turbo Clipper First Line **I8*xxx***	Power Clipper with ACC-84S **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1	Power Clipper Standard Interface **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$278800	**Acc84S[0].Chan[0].SerialEncDataA.a**	**Clipper[0].Chan[0].SerialEncDataA.a**
$278804	**Acc84S[0].Chan[1].SerialEncDataA.a**	**Clipper[0].Chan[1].SerialEncDataA.a**
$278808	**Acc84S[0].Chan[2].SerialEncDataA.a**	**Clipper[0].Chan[2].SerialEncDataA.a**
$27880C	**Acc84S[0].Chan[3].SerialEncDataA.a**	**Clipper[0].Chan[3].SerialEncDataA.a**
$278820	**Acc84S[1].Chan[0].SerialEncDataA.a**	**Clipper[1].Chan[0].SerialEncDataA.a**
$278824	**Acc84S[1].Chan[1].SerialEncDataA.a**	**Clipper[1].Chan[1].SerialEncDataA.a**
$278828	**Acc84S[1].Chan[2].SerialEncDataA.a**	**Clipper[1].Chan[2].SerialEncDataA.a**
$27882C	**Acc84S[1].Chan[3].SerialEncDataA.a**	**Clipper[1].Chan[3].SerialEncDataA.a**

Notes:

1. ACC-84S is not auto-identified by Power PMAC, so must be manually identified by setting **GateIo[*i*].PartNum** to 603936, **GateIo[*i*].PartType** to 8, **`save`**, and reset, in order to use data structure element names.
2. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
3. The first digit of "2" in the Turbo PMAC line can also be "3" if the entry is change-limited. To keep change limiting in Power PMAC, **EncTable[*n*].MaxDelta** must be set.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **Acc84S[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index2 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index3 .. **index6** = 0 // Default
EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// EncTable[*n*].MaxDelta = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1/256 // Source LSB in bit 8 of 32-bit bus

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **Clipper[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 0 // No shifting
EncTable[*n*].index2 = 0 // No shifting
EncTable[*n*].index3 .. **index6** = 0 // Default
EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// EncTable[*n*].MaxDelta = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1.0 // Source LSB in bit 0 of 32-bit bus

Turbo Brick to Power Brick

Turbo Brick with ACC-84B to Power Brick

Turbo Brick First Line **I8*xxx***	Power Brick with ACC-84B **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1	Power Brick Standard Interface **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$278B20	**Acc84B[0].Chan[0].SerialEncDataA.a**	**PowerBrick[0].Chan[0].SerialEncDataA.a**
$278B24	**Acc84B[0].Chan[1].SerialEncDataA.a**	**PowerBrick[0].Chan[1].SerialEncDataA.a**
$278B28	**Acc84B[0].Chan[2].SerialEncDataA.a**	**PowerBrick[0].Chan[2].SerialEncDataA.a**
$278B2C	**Acc84B[0].Chan[3].SerialEncDataA.a**	**PowerBrick[0].Chan[3].SerialEncDataA.a**
$278B30	**Acc84B[1].Chan[0].SerialEncDataA.a**	**PowerBrick[1].Chan[0].SerialEncDataA.a**
$278B34	**Acc84B[1].Chan[1].SerialEncDataA.a**	**PowerBrick[1].Chan[1].SerialEncDataA.a**
$278B38	**Acc84B[1].Chan[2].SerialEncDataA.a**	**PowerBrick[1].Chan[2].SerialEncDataA.a**
$278B3C	**Acc84B[1].Chan[3].SerialEncDataA.a**	**PowerBrick[1].Chan[3].SerialEncDataA.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
2. The first digit of "2" in the Turbo PMAC line can also be "3" if the entry is change-limited. To keep change limiting in Power PMAC, **EncTable[*n*].MaxDelta** must be set.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **Acc84B[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index2 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index3 .. **index6** = 0
EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// EncTable[*n*].MaxDelta = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1/256 // Source LSB in bit 8 of 32-bit bus

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **PowerBrick[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 0 // No shifting
EncTable[*n*].index2 = 0 // No shifting
EncTable[*n*].index3 .. **index6** = 0
EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// EncTable[*n*].MaxDelta = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1.0 // Source LSB in bit 0 of 32-bit bus

Turbo UMAC to Power UMAC with ACC-84E or ACC-24E3

Turbo UMAC with ACC-84E to Power UMAC

Turbo UMAC First Line **I8*xxx***	Power UMAC with ACC-84E **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1	Power UMAC Standard Interface **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$278C00	**Acc84E[0].Chan[0].SerialEncDataA.a**	**Acc24E3[0].Chan[0].SerialEncDataA.a**
$278C04	**Acc84E[0].Chan[1].SerialEncDataA.a**	**Acc24E3[0].Chan[1].SerialEncDataA.a**
$278C08	**Acc84E[0].Chan[2].SerialEncDataA.a**	**Acc24E3[0].Chan[2].SerialEncDataA.a**
$278C0C	**Acc84E[0].Chan[3].SerialEncDataA.a**	**Acc24E3[0].Chan[3].SerialEncDataA.a**
$278D00	**Acc84E[1].Chan[0].SerialEncDataA.a**	**Acc24E3[1].Chan[0].SerialEncDataA.a**
$278D04	**Acc84E[1].Chan[1].SerialEncDataA.a**	**Acc24E3[1].Chan[1].SerialEncDataA.a**
$278D08	**Acc84E[1].Chan[2].SerialEncDataA.a**	**Acc24E3[1].Chan[2].SerialEncDataA.a**
$278D0C	**Acc84E[1].Chan[3].SerialEncDataA.a**	**Acc24E3[1].Chan[3].SerialEncDataA.a**
$278E00	**Acc84E[2].Chan[0].SerialEncDataA.a**	**Acc24E3[2].Chan[0].SerialEncDataA.a**
$278E04	**Acc84E[2].Chan[1].SerialEncDataA.a**	**Acc24E3[2].Chan[1].SerialEncDataA.a**
$278E08	**Acc84E[2].Chan[2].SerialEncDataA.a**	**Acc24E3[2].Chan[2].SerialEncDataA.a**
$278E0C	**Acc84E[2].Chan[3].SerialEncDataA.a**	**Acc24E3[2].Chan[3].SerialEncDataA.a**
$279C00	**Acc84E[4].Chan[0].SerialEncDataA.a**	**Acc24E3[3].Chan[0].SerialEncDataA.a**
$279C04	**Acc84E[4].Chan[1].SerialEncDataA.a**	**Acc24E3[3].Chan[1].SerialEncDataA.a**
$279C08	**Acc84E[4].Chan[2].SerialEncDataA.a**	**Acc24E3[3].Chan[2].SerialEncDataA.a**
$279C0C	**Acc84E[4].Chan[3].SerialEncDataA.a**	**Acc24E3[3].Chan[3].SerialEncDataA.a**
$279D00	**Acc84E[5].Chan[0].SerialEncDataA.a**	**Acc24E3[4].Chan[0].SerialEncDataA.a**
$279D04	**Acc84E[5].Chan[1].SerialEncDataA.a**	**Acc24E3[4].Chan[1].SerialEncDataA.a**
$279D08	**Acc84E[5].Chan[2].SerialEncDataA.a**	**Acc24E3[4].Chan[2].SerialEncDataA.a**
$279D0C	**Acc84E[5].Chan[3].SerialEncDataA.a**	**Acc24E3[4].Chan[3].SerialEncDataA.a**
$279E00	**Acc84E[6].Chan[0].SerialEncDataA.a**	**Acc24E3[5].Chan[0].SerialEncDataA.a**
$279E04	**Acc84E[6].Chan[1].SerialEncDataA.a**	**Acc24E3[5].Chan[1].SerialEncDataA.a**
$279E08	**Acc84E[6].Chan[2].SerialEncDataA.a**	**Acc24E3[5].Chan[2].SerialEncDataA.a**
$279E0C	**Acc84E[6].Chan[3].SerialEncDataA.a**	**Acc24E3[5].Chan[3].SerialEncDataA.a**
$27AC00	**Acc84E[8].Chan[0].SerialEncDataA.a**	**Acc24E3[6].Chan[0].SerialEncDataA.a**
$27AC04	**Acc84E[8].Chan[1].SerialEncDataA.a**	**Acc24E3[6].Chan[1].SerialEncDataA.a**
$27AC08	**Acc84E[8].Chan[2].SerialEncDataA.a**	**Acc24E3[6].Chan[2].SerialEncDataA.a**
$27AC0C	**Acc84E[8].Chan[3].SerialEncDataA.a**	**Acc24E3[6].Chan[3].SerialEncDataA.a**
$27AD00	**Acc84E[9].Chan[0].SerialEncDataA.a**	**Acc24E3[7].Chan[0].SerialEncDataA.a**
$27AD04	**Acc84E[9].Chan[1].SerialEncDataA.a**	**Acc24E3[7].Chan[1].SerialEncDataA.a**
$27AD08	**Acc84E[9].Chan[2].SerialEncDataA.a**	**Acc24E3[7].Chan[2].SerialEncDataA.a**
$27AD0C	**Acc84E[9].Chan[3].SerialEncDataA.a**	**Acc24E3[7].Chan[3].SerialEncDataA.a**
$27AE00	**Acc84E[10].Chan[0].SerialEncDataA.a**	**Acc24E3[8].Chan[0].SerialEncDataA.a**
$27AE04	**Acc84E[10].Chan[1].SerialEncDataA.a**	**Acc24E3[8].Chan[1].SerialEncDataA.a**
$27AE08	**Acc84E[10].Chan[2].SerialEncDataA.a**	**Acc24E3[8].Chan[2].SerialEncDataA.a**
$27AE0C	**Acc84E[10].Chan[3].SerialEncDataA.a**	**Acc24E3[8].Chan[3].SerialEncDataA.a**

Notes:

1. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
2. The first digit of "2" in the Turbo PMAC line can also be "3" if the entry is change-limited. To keep change limiting in Power PMAC, **EncTable[*n*].MaxDelta** must be set.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **Acc84E[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index2 = 8 // Shifting to clear low 8 bits of non-existent data
EncTable[*n*].index3 .. **index6** = 0 // Default

EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// **EncTable[*n*].MaxDelta** = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1/256 // SourceLSB in bit 8 of 32-bit bus

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **Acc24E3[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 0 // No shifting
EncTable[*n*].index2 = 0 // No shifting
EncTable[*n*].index3 .. **index6** = 0 // Default
EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// **EncTable[*n*].MaxDelta** = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1.0 // Source LSB in bit 0 of 32-bit bus

Turbo UMAC to CK3M with CK3W-AX*nnnn*

Turbo UMAC with ACC-84E to CK3M with CK3W-AX*nnnn*

Turbo UMAC First Line **I8*xxx***	CK3M with CK3W-AX*nnnn* **EncTable[*n*].pEnc** for **EncTable[*n*].type** = 1
$278C00	**CK3WAX[0].Chan[0].SerialEncDataA.a**
$278C04	**CK3WAX[0].Chan[1].SerialEncDataA.a**
$278C08	**CK3WAX[0].Chan[2].SerialEncDataA.a**
$278C0C	**CK3WAX[0].Chan[3].SerialEncDataA.a**
$278D00	**CK3WAX[1].Chan[0].SerialEncDataA.a**
$278D04	**CK3WAX[1].Chan[1].SerialEncDataA.a**
$278D08	**CK3WAX[1].Chan[2].SerialEncDataA.a**
$278D0C	**CK3WAX[1].Chan[3].SerialEncDataA.a**
$278E00	**CK3WAX[2].Chan[0].SerialEncDataA.a**
$278E04	**CK3WAX[2].Chan[1].SerialEncDataA.a**
$278E08	**CK3WAX[2].Chan[2].SerialEncDataA.a**
$278E0C	**CK3WAX[2].Chan[3].SerialEncDataA.a**
$279C00	**CK3WAX[3].Chan[0].SerialEncDataA.a**
$279C04	**CK3WAX[3].Chan[1].SerialEncDataA.a**
$279C08	**CK3WAX[3].Chan[2].SerialEncDataA.a**
$279C0C	**CK3WAX[3].Chan[3].SerialEncDataA.a**
$279D00	**CK3WAX[4].Chan[0].SerialEncDataA.a**
$279D04	**CK3WAX[4].Chan[1].SerialEncDataA.a**
$279D08	**CK3WAX[4].Chan[2].SerialEncDataA.a**
$279D0C	**CK3WAX[4].Chan[3].SerialEncDataA.a**
$279E00	**CK3WAX[5].Chan[0].SerialEncDataA.a**
$279E04	**CK3WAX[5].Chan[1].SerialEncDataA.a**
$279E08	**CK3WAX[5].Chan[2].SerialEncDataA.a**
$279E0C	**CK3WAX[5].Chan[3].SerialEncDataA.a**
$27AC00	**CK3WAX[6].Chan[0].SerialEncDataA.a**
$27AC04	**CK3WAX[6].Chan[1].SerialEncDataA.a**
$27AC08	**CK3WAX[6].Chan[2].SerialEncDataA.a**
$27AC0C	**CK3WAX[6].Chan[3].SerialEncDataA.a**
$27AD00	**CK3WAX[7].Chan[0].SerialEncDataA.a**
$27AD04	**CK3WAX[7].Chan[1].SerialEncDataA.a**
$27AD08	**CK3WAX[7].Chan[2].SerialEncDataA.a**
$27AD0C	**CK3WAX[7].Chan[3].SerialEncDataA.a**
$27AE00	**CK3WAX[8].Chan[0].SerialEncDataA.a**
$27AE04	**CK3WAX[8].Chan[1].SerialEncDataA.a**
$27AE08	**CK3WAX[8].Chan[2].SerialEncDataA.a**
$27AE0C	**CK3WAX[8].Chan[3].SerialEncDataA.a**

Notes:

1. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
2. The first digit of "2" in the Turbo PMAC line can also be "3" if the entry is change-limited. To keep change limiting in Power PMAC, **EncTable[*n*].MaxDelta** must be set.

EncTable[*n*].type = 1
EncTable[*n*].pEnc = **CK3WAX[*i*].Chan[*j*].SerialEncData.a** *{from table}*
EncTable[*n*].pEnc1 = *{don't care}*
EncTable[*n*].index1 = 0 // No shifting
EncTable[*n*].index2 = 0 // No shifting
EncTable[*n*].index3 .. **index6** = 0

EncTable[*n*].MaxDelta = 0 // If Turbo method 2 (no change limiting)
// **EncTable[*n*].MaxDelta** = *{third line}* // If Turbo method 3 (change limited)
EncTable[*n*].ScaleFactor = 1.0 // Source LSB in bit 0 of 32-bit bus

Motor I-Variable Equivalents

In Turbo PMAC documentation, the motor-number specification in these I-variables is given generically as ***xx***, as in **I*xx*22**. In Power PMAC documentation, the motor-number specification is given as ***x***, as in **Motor[*x*].JogSpeed**.

Turbo PMAC motor numbers start with 1 (and go to 32). Power PMAC motor numbers start with 0 (and go to 255), but Motor 0 is rarely used to command a physical motor. The note explains the conversion from the setup of a Turbo PMAC motor to the Power PMAC motor of the same number.

In Power PMAC, motors numbered from 0 to (**Sys.MaxMotors** – 1) can be activated and configured. At the default value for **Sys.MaxMotors** of 32, Motors 0 – 31 can be used. If Motor 32 was used in the Turbo PMAC application, **Sys.MaxMotors** must be increased to a value of 33 or greater for compatibility.

(Having **Sys.MaxMotors** set to a higher value than needed introduces a very slight computational penalty and makes backup files larger than necessary. Typically, neither issue is critical, particularly in applications ported from the older Turbo PMAC.)

Turbo PMAC motor units are always in terms of feedback counts or LSBs. For the simplest possible conversion to Power PMAC configuration, Power PMAC units will also be in terms of feedback counts or LSBs.

Basic Motor Operation

- Set **Motor[*x*].ServoCtrl** to the same value (0 or 1) as **I*xx*00** to de-activate or activate the motor, respectively. If set to 0, no further settings for the motor need to be made.

- If **I*xx*01** is 0, set **Motor[*x*].PhaseCtrl** to 0. If **I*xx*01** is 1, set **Motor[*x*].PhaseCtrl** to 4. If set to 0, no commutation or current-loop settings need to be made for the motor.

- Set **Motor[*x*].MasterCtrl** equal to **I*xx*06** for the same position following functionality (enable/disable and offset/standard mode).

Motor Addressing Variables

If the Turbo PMAC application used the default input and output addresses for the motor functions, then the Power PMAC default addresses will work as well.

I*xx*02 to Motor[*x*].pDac

- Set **Motor[*x*].pDac** to the Power PMAC address that matches the **I*xx*02** setting. Common cases are shown below:

Turbo Clipper to Power Clipper

Turbo Clipper Analog or PWM Output to Power Clipper

Turbo Clipper **I*xx*02**	Power Clipper **Motor[*x*].pDac** for True-DAC, PWM Output	Power Clipper **Motor[*x*].pDac** for Filtered-PWM Analog Output
$078002	**Clipper[0].Chan[0].Pwm[0].a**	**Clipper[0].Chan[0].Pwm[2].a**
$07800A	**Clipper[0].Chan[1].Pwm[0].a**	**Clipper[0].Chan[1].Pwm[2].a**
$078012	**Clipper[0].Chan[2].Pwm[0].a**	**Clipper[0].Chan[2].Pwm[2].a**
$07801A	**Clipper[0].Chan[3].Pwm[0].a**	**Clipper[0].Chan[3].Pwm[2].a**
$078102	**Clipper[1].Chan[0].Pwm[0].a**	**Clipper[1].Chan[0].Pwm[2].a**
$07810A	**Clipper[1].Chan[1].Pwm[0].a**	**Clipper[1].Chan[1].Pwm[2].a**
$078112	**Clipper[1].Chan[2].Pwm[0].a**	**Clipper[1].Chan[2].Pwm[2].a**
$07811A	**Clipper[1].Chan[3].Pwm[0].a**	**Clipper[1].Chan[3].Pwm[2].a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **Pwm[0].a** addresses can also be entered as **Dac[0].a**, but will report back as **Pwm[0].a**.
3. Clipper **Pwm[2].a** addresses can also be entered as **Dac[2].a**, but will report back as **Pwm[2].a**.
4. Power Clipper default settings are to **Pwm[0].a**; must be changed to use **Pwm[2].a** addresses.
5. True-DAC output requires ACC-8AS; direct-PWM requires ACC-8FS.

Turbo Clipper Pulse and Direction (PFM) Output to Power Clipper

Turbo Clipper **I*xx*02**	Power Clipper **Motor[*x*].pDac** for PFM Output
$078004	**Clipper[0].Chan[0].Pwm[3].a**
$07800C	**Clipper[0].Chan[1].Pwm[3].a**
$078014	**Clipper[0].Chan[2].Pwm[3].a**
$07801C	**Clipper[0].Chan[3].Pwm[3].a**
$078104	**Clipper[1].Chan[0].Pwm[3].a**
$07810C	**Clipper[1].Chan[1].Pwm[3].a**
$078114	**Clipper[1].Chan[2].Pwm[3].a**
$07811C	**Clipper[1].Chan[3].Pwm[3].a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **Pwm[3].a** addresses can also be entered as **Pfm.a**, but will report back as **Pwm[3].a**.

Turbo Brick to Power Brick

Turbo Brick Analog or PWM Output to Power Brick

Turbo Brick **I*xx*02**	Power Brick **Motor[*x*].pDac** for True-DAC, PWM Output
$078002	**PowerBrick[0].Chan[0].Pwm[0].a**
$07800A	**PowerBrick[0].Chan[1].Pwm[0].a**
$078012	**PowerBrick[0].Chan[2].Pwm[0].a**
$07801A	**PowerBrick[0].Chan[3].Pwm[0].a**
$078102	**PowerBrick[1].Chan[0].Pwm[0].a**
$07810A	**PowerBrick[1].Chan[1].Pwm[0].a**
$078112	**PowerBrick[1].Chan[2].Pwm[0].a**
$07811A	**PowerBrick[1].Chan[3].Pwm[0].a**

Notes:

1. If Turbo Brick uses default configuration, Power Brick can use default configuration.
2. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
3. Brick **Pwm[0].a** addresses can also be entered as **Dac[0].a**, but will report back as **Pwm[0].a**.

Turbo UMAC to Power UMAC

Turbo UMAC Analog or PWM Output to Power UMAC or CK3M

Turbo UMAC **I*xx*02**	Power UMAC **Motor[*x*].pDac** for ACC-24E2*x* True-DAC, PWM Output	Power UMAC **Motor[*x*].pDac** for ACC-24E3 True-DAC, PWM Output	CK3M **Motor[*x*].pDac** for CK3W-AX*nnnn* True-DAC, Filtered PWM, or PWM Output
$078202	**Acc24E2*x*[4].Chan[0].Pwm[0].a**	**Acc24E3[0].Chan[0].Pwm[0].a**	**CK3WAX[0].Chan[0].Pwm[0].a**
$07820A	**Acc24E2*x*[4].Chan[1].Pwm[0].a**	**Acc24E3[0].Chan[1].Pwm[0].a**	**CK3WAX[0].Chan[1].Pwm[0].a**
$078212	**Acc24E2*x*[4].Chan[2].Pwm[0].a**	**Acc24E3[0].Chan[2].Pwm[0].a**	**CK3WAX[0].Chan[2].Pwm[0].a**
$07821A	**Acc24E2*x*[4].Chan[3].Pwm[0].a**	**Acc24E3[0].Chan[3].Pwm[0].a**	**CK3WAX[0].Chan[3].Pwm[0].a**
$078302	**Acc24E2*x*[6].Chan[0].Pwm[0].a**	**Acc24E3[1].Chan[0].Pwm[0].a**	**CK3WAX[1].Chan[0].Pwm[0].a**
$07830A	**Acc24E2*x*[6].Chan[1].Pwm[0].a**	**Acc24E3[1].Chan[1].Pwm[0].a**	**CK3WAX[1].Chan[1].Pwm[0].a**
$078312	**Acc24E2*x*[6].Chan[2].Pwm[0].a**	**Acc24E3[1].Chan[2].Pwm[0].a**	**CK3WAX[1].Chan[2].Pwm[0].a**
$07831A	**Acc24E2*x*[6].Chan[3].Pwm[0].a**	**Acc24E3[1].Chan[3].Pwm[0].a**	**CK3WAX[1].Chan[3].Pwm[0].a**
$079202	**Acc24E2*x*[8].Chan[0].Pwm[0].a**	**Acc24E3[2].Chan[0].Pwm[0].a**	**CK3WAX[2].Chan[0].Pwm[0].a**
$07920A	**Acc24E2*x*[8].Chan[1].Pwm[0].a**	**Acc24E3[2].Chan[1].Pwm[0].a**	**CK3WAX[2].Chan[1].Pwm[0].a**
$079212	**Acc24E2*x*[8].Chan[2].Pwm[0].a**	**Acc24E3[2].Chan[2].Pwm[0].a**	**CK3WAX[2].Chan[2].Pwm[0].a**
$07921A	**Acc24E2*x*[8].Chan[3].Pwm[0].a**	**Acc24E3[2].Chan[3].Pwm[0].a**	**CK3WAX[2].Chan[3].Pwm[0].a**
$079302	**Acc24E2*x*[10].Chan[0].Pwm[0].a**	**Acc24E3[3].Chan[0].Pwm[0].a**	**CK3WAX[3].Chan[0].Pwm[0].a**
$07930A	**Acc24E2*x*[10].Chan[1].Pwm[0].a**	**Acc24E3[3].Chan[1].Pwm[0].a**	**CK3WAX[3].Chan[1].Pwm[0].a**
$079312	**Acc24E2*x*[10].Chan[2].Pwm[0].a**	**Acc24E3[3].Chan[2].Pwm[0].a**	**CK3WAX[3].Chan[2].Pwm[0].a**
$07931A	**Acc24E2*x*[10].Chan[3].Pwm[0].a**	**Acc24E3[3].Chan[3].Pwm[0].a**	**CK3WAX[3].Chan[3].Pwm[0].a**
$07A202	**Acc24E2*x*[12].Chan[0].Pwm[0].a**	**Acc24E3[4].Chan[0].Pwm[0].a**	**CK3WAX[4].Chan[0].Pwm[0].a**
$07A20A	**Acc24E2*x*[12].Chan[1].Pwm[0].a**	**Acc24E3[4].Chan[1].Pwm[0].a**	**CK3WAX[4].Chan[1].Pwm[0].a**
$07A212	**Acc24E2*x*[12].Chan[2].Pwm[0].a**	**Acc24E3[4].Chan[2].Pwm[0].a**	**CK3WAX[4].Chan[2].Pwm[0].a**
$07A21A	**Acc24E2*x*[12].Chan[3].Pwm[0].a**	**Acc24E3[4].Chan[3].Pwm[0].a**	**CK3WAX[4].Chan[3].Pwm[0].a**
$07A302	**Acc24E2*x*[14].Chan[0].Pwm[0].a**	**Acc24E3[5].Chan[0].Pwm[0].a**	**CK3WAX[5].Chan[0].Pwm[0].a**
$07A30A	**Acc24E2*x*[14].Chan[1].Pwm[0].a**	**Acc24E3[5].Chan[1].Pwm[0].a**	**CK3WAX[5].Chan[1].Pwm[0].a**
$07A312	**Acc24E2*x*[14].Chan[2].Pwm[0].a**	**Acc24E3[5].Chan[2].Pwm[0].a**	**CK3WAX[5].Chan[2].Pwm[0].a**
$07A31A	**Acc24E2*x*[14].Chan[3].Pwm[0].a**	**Acc24E3[5].Chan[3].Pwm[0].a**	**CK3WAX[5].Chan[3].Pwm[0].a**
$07B202	**Acc24E2*x*[16].Chan[0].Pwm[0].a**	**Acc24E3[6].Chan[0].Pwm[0].a**	**CK3WAX[6].Chan[0].Pwm[0].a**
$07B20A	**Acc24E2*x*[16].Chan[1].Pwm[0].a**	**Acc24E3[6].Chan[1].Pwm[0].a**	**CK3WAX[6].Chan[1].Pwm[0].a**
$07B212	**Acc24E2*x*[16].Chan[2].Pwm[0].a**	**Acc24E3[6].Chan[2].Pwm[0].a**	**CK3WAX[6].Chan[2].Pwm[0].a**
$07B21A	**Acc24E2*x*[16].Chan[3].Pwm[0].a**	**Acc24E3[6].Chan[3].Pwm[0].a**	**CK3WAX[6].Chan[3].Pwm[0].a**
$07B302	**Acc24E2*x*[18].Chan[0].Pwm[0].a**	**Acc24E3[7].Chan[0].Pwm[0].a**	**CK3WAX[7].Chan[0].Pwm[0].a**
$07B30A	**Acc24E2*x*[18].Chan[1].Pwm[0].a**	**Acc24E3[7].Chan[1].Pwm[0].a**	**CK3WAX[7].Chan[1].Pwm[0].a**
$07B312	**Acc24E2*x*[18].Chan[2].Pwm[0].a**	**Acc24E3[7].Chan[2].Pwm[0].a**	**CK3WAX[7].Chan[2].Pwm[0].a**
$07B31A	**Acc24E2*x*[18].Chan[3].Pwm[0].a**	**Acc24E3[7].Chan[3].Pwm[0].a**	**CK3WAX[7].Chan[3].Pwm[0].a**

Notes:

1. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.
2. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
3. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
4. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
5. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
6. **Pwm[0].a** addresses can also be entered as **Dac[0].a**, but will report back as **Pwm[0].a**.

Turbo UMAC Pulse and Direction (PFM) Output to Power UMAC or CK3M

Turbo UMAC **I*xx*02**	Power UMAC **Motor[*x*].pDac** for ACC-24E2*x* PFM Output	Power UMAC **Motor[*x*].pDac** for ACC-24E3 PFM Output	Power UMAC **Motor[*x*].pDac** for CK3W-AX PFM Output
$078204	**Acc24E2*x*[4].Chan[0].Pwm[2].a**	**Acc24E3[0].Chan[0].Pwm[3].a**	**CK3WAX[0].Chan[0].Pwm[3].a**
$07820C	**Acc24E2*x*[4].Chan[1].Pwm[2].a**	**Acc24E3[0].Chan[1].Pwm[3].a**	**CK3WAX[0].Chan[1].Pwm[3].a**
$078214	**Acc24E2*x*[4].Chan[2].Pwm[2].a**	**Acc24E3[0].Chan[2].Pwm[3].a**	**CK3WAX[0].Chan[2].Pwm[3].a**
$07821C	**Acc24E2*x*[4].Chan[3].Pwm[2].a**	**Acc24E3[0].Chan[3].Pwm[3].a**	**CK3WAX[0].Chan[3].Pwm[3].a**
$078304	**Acc24E2*x*[6].Chan[0].Pwm[2].a**	**Acc24E3[1].Chan[0].Pwm[3].a**	**CK3WAX[1].Chan[0].Pwm[3].a**
$07830C	**Acc24E2*x*[6].Chan[1].Pwm[2].a**	**Acc24E3[1].Chan[1].Pwm[3].a**	**CK3WAX[1].Chan[1].Pwm[3].a**
$078314	**Acc24E2*x*[6].Chan[2].Pwm[2].a**	**Acc24E3[1].Chan[2].Pwm[3].a**	**CK3WAX[1].Chan[2].Pwm[3].a**
$07831C	**Acc24E2*x*[6].Chan[3].Pwm[2].a**	**Acc24E3[1].Chan[3].Pwm[3].a**	**CK3WAX[1].Chan[3].Pwm[3].a**
$079204	**Acc24E2*x*[8].Chan[0].Pwm[2].a**	**Acc24E3[2].Chan[0].Pwm[3].a**	**CK3WAX[2].Chan[0].Pwm[3].a**
$07920C	**Acc24E2*x*[8].Chan[1].Pwm[2].a**	**Acc24E3[2].Chan[1].Pwm[3].a**	**CK3WAX[2].Chan[1].Pwm[3].a**
$079214	**Acc24E2*x*[8].Chan[2].Pwm[2].a**	**Acc24E3[2].Chan[2].Pwm[3].a**	**CK3WAX[2].Chan[2].Pwm[3].a**
$07921C	**Acc24E2*x*[8].Chan[3].Pwm[2].a**	**Acc24E3[2].Chan[3].Pwm[3].a**	**CK3WAX[2].Chan[3].Pwm[3].a**
$079304	**Acc24E2*x*[10].Chan[0].Pwm[2].a**	**Acc24E3[3].Chan[0].Pwm[3].a**	**CK3WAX[3].Chan[0].Pwm[3].a**
$07930C	**Acc24E2*x*[10].Chan[1].Pwm[2].a**	**Acc24E3[3].Chan[1].Pwm[3].a**	**CK3WAX[3].Chan[1].Pwm[3].a**
$079314	**Acc24E2*x*[10].Chan[2].Pwm[2].a**	**Acc24E3[3].Chan[2].Pwm[3].a**	**CK3WAX[3].Chan[2].Pwm[3].a**
$07931C	**Acc24E2*x*[10].Chan[3].Pwm[2].a**	**Acc24E3[3].Chan[3].Pwm[3].a**	**CK3WAX[3].Chan[3].Pwm[3].a**
$07A204	**Acc24E2*x*[12].Chan[0].Pwm[2].a**	**Acc24E3[4].Chan[0].Pwm[3].a**	**CK3WAX[4].Chan[0].Pwm[3].a**
$07A20C	**Acc24E2*x*[12].Chan[1].Pwm[2].a**	**Acc24E3[4].Chan[1].Pwm[3].a**	**CK3WAX[4].Chan[1].Pwm[3].a**
$07A214	**Acc24E2*x*[12].Chan[2].Pwm[2].a**	**Acc24E3[4].Chan[2].Pwm[3].a**	**CK3WAX[4].Chan[2].Pwm[3].a**
$07A21C	**Acc24E2*x*[12].Chan[3].Pwm[2].a**	**Acc24E3[4].Chan[3].Pwm[3].a**	**CK3WAX[4].Chan[3].Pwm[3].a**
$07A304	**Acc24E2*x*[14].Chan[0].Pwm[2].a**	**Acc24E3[5].Chan[0].Pwm[3].a**	**CK3WAX[5].Chan[0].Pwm[3].a**
$07A30C	**Acc24E2*x*[14].Chan[1].Pwm[2].a**	**Acc24E3[5].Chan[1].Pwm[3].a**	**CK3WAX[5].Chan[1].Pwm[3].a**
$07A314	**Acc24E2*x*[14].Chan[2].Pwm[2].a**	**Acc24E3[5].Chan[2].Pwm[3].a**	**CK3WAX[5].Chan[2].Pwm[3].a**
$07A31C	**Acc24E2*x*[14].Chan[3].Pwm[2].a**	**Acc24E3[5].Chan[3].Pwm[3].a**	**CK3WAX[5].Chan[3].Pwm[3].a**
$07B204	**Acc24E2*x*[16].Chan[0].Pwm[2].a**	**Acc24E3[6].Chan[0].Pwm[3].a**	**CK3WAX[6].Chan[0].Pwm[3].a**
$07B20C	**Acc24E2*x*[16].Chan[1].Pwm[2].a**	**Acc24E3[6].Chan[1].Pwm[3].a**	**CK3WAX[6].Chan[1].Pwm[3].a**
$07B214	**Acc24E2*x*[16].Chan[2].Pwm[2].a**	**Acc24E3[6].Chan[2].Pwm[3].a**	**CK3WAX[6].Chan[2].Pwm[3].a**
$07B21C	**Acc24E2*x*[16].Chan[3].Pwm[2].a**	**Acc24E3[6].Chan[3].Pwm[3].a**	**CK3WAX[6].Chan[3].Pwm[3].a**
$07B304	**Acc24E2*x*[18].Chan[0].Pwm[2].a**	**Acc24E3[7].Chan[0].Pwm[3].a**	**CK3WAX[7].Chan[0].Pwm[3].a**
$07B30C	**Acc24E2*x*[18].Chan[1].Pwm[2].a**	**Acc24E3[7].Chan[1].Pwm[3].a**	**CK3WAX[7].Chan[1].Pwm[3].a**
$07B314	**Acc24E2*x*[18].Chan[2].Pwm[2].a**	**Acc24E3[7].Chan[2].Pwm[3].a**	**CK3WAX[7].Chan[2].Pwm[3].a**
$07B31C	**Acc24E2*x*[18].Chan[3].Pwm[2].a**	**Acc24E3[7].Chan[3].Pwm[3].a**	**CK3WAX[7].Chan[3].Pwm[3].a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
4. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
5. ACC-24E2*x* **Pwm[2].a** addresses can also be entered as **Pfm.a**, but will report back as **Pwm[2].a**.
6. ACC-24E3 **Pwm[3].a** addresses can also be entered as **Pfm.a**, but will report back as **Pwm[3].a**.
7. CK3W-AX **Pwm[3].a** addresses can also be entered as **Pfm.a**, but will report back as **Pwm[3].a**.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Node 0 Output to Power PMAC

Turbo PMAC **I*xx*02**	Power UMAC **Motor[*x*].pDac** for ACC-5E Node 0 Output	Power PMAC **Motor[*x*].pDac** for ACC-5E3 Node 0 Output
$078420	**Acc5E[0].Macro[0][0].a**	**Acc5E3[0].MacroOutA[0][0].a**
$078424	**Acc5E[0].Macro[1][0].a**	**Acc5E3[0].MacroOutA[1][0].a**
$078428	**Acc5E[0].Macro[4][0].a**	**Acc5E3[0].MacroOutA[4][0].a**
$07842C	**Acc5E[0].Macro[5][0].a**	**Acc5E3[0].MacroOutA[5][0].a**
$078430	**Acc5E[0].Macro[8][0].a**	**Acc5E3[0].MacroOutA[8][0].a**
$078434	**Acc5E[0].Macro[9][0].a**	**Acc5E3[0].MacroOutA[9][0].a**
$078438	**Acc5E[0].Macro[12][0].a**	**Acc5E3[0].MacroOutA[12][0].a**
$07843C	**Acc5E[0].Macro[13][0].a**	**Acc5E3[0].MacroOutA[13][0].a**
$079420	**Acc5E[1].Macro[0][0].a**	**Acc5E3[0].MacroOutB[0][0].a**
$079424	**Acc5E[1].Macro[1][0].a**	**Acc5E3[0].MacroOutB[1][0].a**
$079428	**Acc5E[1].Macro[4][0].a**	**Acc5E3[0].MacroOutB[4][0].a**
$07942C	**Acc5E[1].Macro[5][0].a**	**Acc5E3[0].MacroOutB[5][0].a**
$079430	**Acc5E[1].Macro[8][0].a**	**Acc5E3[0].MacroOutB[8][0].a**
$079434	**Acc5E[1].Macro[9][0].a**	**Acc5E3[0].MacroOutB[9][0].a**
$079438	**Acc5E[1].Macro[12][0].a**	**Acc5E3[0].MacroOutB[12][0].a**
$07943C	**Acc5E[1].Macro[13][0].a**	**Acc5E3[0].MacroOutB[13][0].a**
$07A420	**Acc5E[2].Macro[0][0].a**	**Acc5E3[1].MacroOutA[0][0].a**
$07A424	**Acc5E[2].Macro[1][0].a**	**Acc5E3[1].MacroOutA[1][0].a**
$07A428	**Acc5E[2].Macro[4][0].a**	**Acc5E3[1].MacroOutA[4][0].a**
$07A42C	**Acc5E[2].Macro[5][0].a**	**Acc5E3[1].MacroOutA[5][0].a**
$07A430	**Acc5E[2].Macro[8][0].a**	**Acc5E3[1].MacroOutA[8][0].a**
$07A434	**Acc5E[2].Macro[9][0].a**	**Acc5E3[1].MacroOutA[9][0].a**
$07A438	**Acc5E[2].Macro[12][0].a**	**Acc5E3[1].MacroOutA[12][0].a**
$07A43C	**Acc5E[2].Macro[13][0].a**	**Acc5E3[1].MacroOutA[13][0].a**
$07B420	**Acc5E[3].Macro[0][0].a**	**Acc5E3[1].MacroOutB[0][0].a**
$07B424	**Acc5E[3].Macro[1][0].a**	**Acc5E3[1].MacroOutB[1][0].a**
$07B428	**Acc5E[3].Macro[4][0].a**	**Acc5E3[1].MacroOutB[4][0].a**
$07B42C	**Acc5E[3].Macro[5][0].a**	**Acc5E3[1].MacroOutB[5][0].a**
$07B430	**Acc5E[3].Macro[8][0].a**	**Acc5E3[1].MacroOutB[8][0].a**
$07B434	**Acc5E[3].Macro[9][0].a**	**Acc5E3[1].MacroOutB[9][0].a**
$07B438	**Acc5E[3].Macro[12][0].a**	**Acc5E3[1].MacroOutB[12][0].a**
$07B43C	**Acc5E[3].Macro[13][0].a**	**Acc5E3[1].MacroOutB[13][0].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.
3. If ACC-5EP3 is used, the data structure name is **Acc5EP3[*i*]** instead of **Acc5E3[*i*]**.

I*xx*03, I*xx*04 to Motor[*x*].pEnc, pEnc2

- Set **Motor[*x*].pEnc** to **EncTable[*n*].a** for the encoder conversion table entry ***n*** that matches the Turbo PMAC address of **I*xx*03**.

- Set **Motor[*x*].pEnc2** to **EncTable[*n*].a** for the encoder conversion table entry ***n*** that matches the Turbo PMAC address of **I*xx*04**.

In the large majority of applications, motors use the same feedback value for both the outer (position) loop and the inner (velocity) loop, so these two variables are set the same, on both Turbo PMACs and Power PMACs. In almost all of these cases, **Motor[*x*]** uses **EncTable[*n*]** where ***x*** = ***n***. If dual feedback is used for some or all motors, the relationship will not be as simple.

Turbo Clipper to Power Clipper

Turbo Clipper Single-Line Encoder Table Feedback to Power Clipper

Turbo Clipper **I*xx*03**, **I*xx*04**	Power Clipper **Motor[*x*].pEnc**, **pEnc2**
$003501	**EncTable[1].a**
$003502	**EncTable[2].a**
$003503	**EncTable[3].a**
$003504	**EncTable[4].a**
$003505	**EncTable[5].a**
$003506	**EncTable[6].a**
$003507	**EncTable[7].a**
$003508	**EncTable[8].a**

Notes:

1. If Turbo Clipper uses default configuration, Power Clipper can use default configuration.

Turbo Clipper Double-Line Encoder Table Feedback to Power Clipper

Turbo Clipper **I*xx*03**, **I*xx*04**	Power Clipper **Motor[*x*].pEnc**, **pEnc2**
$003502	**EncTable[1].a**
$003504	**EncTable[2].a**
$003506	**EncTable[3].a**
$003508	**EncTable[4].a**
$00350A	**EncTable[5].a**
$00350C	**EncTable[6].a**
$00350E	**EncTable[7].a**
$003510	**EncTable[8].a**

Turbo Clipper Triple-Line Encoder Table Feedback to Power Clipper

Turbo Clipper **I*xx*03**, **I*xx*04**	Power Clipper **Motor[*x*].pEnc**, **pEnc2**
$003503	**EncTable[1].a**
$003506	**EncTable[2].a**
$003509	**EncTable[3].a**
$00350C	**EncTable[4].a**
$00350F	**EncTable[5].a**
$003512	**EncTable[6].a**
$003515	**EncTable[7].a**
$003518	**EncTable[8].a**

Turbo Brick to Power Brick

Turbo Brick Single-Line Encoder Table Feedback to Power Brick

Turbo Brick **I*xx*03**, **I*xx*04**	Power Brick **Motor[*x*].pEnc**, **pEnc2**
$003501	**EncTable[1].a**
$003502	**EncTable[2].a**
$003503	**EncTable[3].a**
$003504	**EncTable[4].a**
$003505	**EncTable[5].a**
$003506	**EncTable[6].a**
$003507	**EncTable[7].a**
$003508	**EncTable[8].a**

Notes:

1. If Turbo Brick uses default configuration, Power Brick can use default configuration.

Turbo Brick Double-Line Encoder Table Feedback to Power Brick

Turbo Brick **I*xx*03**, **I*xx*04**	Power Brick **Motor[*x*].pEnc**, **pEnc2**
$003502	**EncTable[1].a**
$003504	**EncTable[2].a**
$003506	**EncTable[3].a**
$003508	**EncTable[4].a**
$00350A	**EncTable[5].a**
$00350C	**EncTable[6].a**
$00350E	**EncTable[7].a**
$003510	**EncTable[8].a**

Turbo Brick Triple-Line Encoder Table Feedback to Power Brick

Turbo Brick **I*xx*03**, **I*xx*04**	Power Brick **Motor[*x*].pEnc**, **pEnc2**
$003503	**EncTable[1].a**
$003506	**EncTable[2].a**
$003509	**EncTable[3].a**
$00350C	**EncTable[4].a**
$00350F	**EncTable[5].a**
$003512	**EncTable[6].a**
$003515	**EncTable[7].a**
$003518	**EncTable[8].a**

Turbo UMAC to Power UMAC, Turbo PMAC MACRO to Power PMAC MACRO

Turbo UMAC Single-Line Encoder Table Feedback to Power UMAC

Turbo UMAC **I*xx*03**, **I*xx*04**	Power UMAC **Motor[*x*].pEnc**, **pEnc2**	Turbo UMAC **I*xx*03**, **I*xx*04**	Power UMAC **Motor[*x*].pEnc**, **pEnc2**
$003501	**EncTable[1].a**	$003511	**EncTable[17].a**
$003502	**EncTable[2].a**	$003512	**EncTable[18].a**
$003503	**EncTable[3].a**	$003513	**EncTable[19].a**
$003504	**EncTable[4].a**	$003514	**EncTable[20].a**
$003505	**EncTable[5].a**	$003515	**EncTable[21].a**
$003506	**EncTable[6].a**	$003516	**EncTable[22].a**
$003507	**EncTable[7].a**	$003517	**EncTable[23].a**
$003508	**EncTable[8].a**	$003518	**EncTable[24].a**
$003509	**EncTable[9].a**	$003519	**EncTable[25].a**
$00350A	**EncTable[10].a**	$00351A	**EncTable[26].a**
$00350B	**EncTable[11].a**	$00351B	**EncTable[27].a**
$00350C	**EncTable[12].a**	$00351C	**EncTable[28].a**
$00350D	**EncTable[13].a**	$00351D	**EncTable[29].a**
$00350E	**EncTable[14].a**	$00351E	**EncTable[30].a**
$00350F	**EncTable[15].a**	$00351F	**EncTable[31].a**
$003510	**EncTable[16].a**	$003520	**EncTable[32].a**

Notes:

1. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.
2. Turbo PMAC MACRO generally cannot use single-line encoder table entries

Turbo PMAC Double-Line Encoder Table Feedback to Power PMAC

Turbo UMAC **I*xx*03**, **I*xx*04**	Power UMAC **Motor[*x*].pEnc**, **pEnc2**	Turbo UMAC **I*xx*03**, **I*xx*04**	Power UMAC **Motor[*x*].pEnc**, **pEnc2**
$003502	**EncTable[1].a**	$003522	**EncTable[17].a**
$003504	**EncTable[2].a**	$003524	**EncTable[18].a**
$003506	**EncTable[3].a**	$003526	**EncTable[19].a**
$003508	**EncTable[4].a**	$003528	**EncTable[20].a**
$00350A	**EncTable[5].a**	$00352A	**EncTable[21].a**
$00350C	**EncTable[6].a**	$00352C	**EncTable[22].a**
$00350E	**EncTable[7].a**	$00352E	**EncTable[23].a**
$003510	**EncTable[8].a**	$003530	**EncTable[24].a**
$003512	**EncTable[9].a**	$003532	**EncTable[25].a**
$003514	**EncTable[10].a**	$003534	**EncTable[26].a**
$003516	**EncTable[11].a**	$003536	**EncTable[27].a**
$003518	**EncTable[12].a**	$003538	**EncTable[28].a**
$00351A	**EncTable[13].a**	$00353A	**EncTable[29].a**
$00351C	**EncTable[14].a**	$00353C	**EncTable[30].a**
$00351E	**EncTable[15].a**	$00353E	**EncTable[31].a**
$003520	**EncTable[16].a**	$003540	**EncTable[32].a**

Notes:

1. This is the standard configuration for Turbo PMAC Ultralites with MACRO feedback
2. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.

Turbo PMAC Triple-Line Encoder Table Feedback to Power PMAC

Turbo PMAC **Ixx03**, **Ixx04**	Power PMAC **Motor[*x*].pEnc**, **pEnc2**	Turbo PMAC **Ixx03**, **Ixx04**	Power PMAC **Motor[*x*].pEnc**, **pEnc2**
$003503	**EncTable[1].a**	$003533	**EncTable[17].a**
$003506	**EncTable[2].a**	$003536	**EncTable[18].a**
$003509	**EncTable[3].a**	$003539	**EncTable[19].a**
$00350C	**EncTable[4].a**	$00353C	**EncTable[20].a**
$00350F	**EncTable[5].a**	$00353F	**EncTable[21].a**
$003512	**EncTable[6].a**	$003542	**EncTable[22].a**
$003515	**EncTable[7].a**	$003545	**EncTable[23].a**
$003518	**EncTable[8].a**	$003548	**EncTable[24].a**
$00351B	**EncTable[9].a**	$00354B	**EncTable[25].a**
$00351E	**EncTable[10].a**	$00354E	**EncTable[26].a**
$003521	**EncTable[11].a**	$003551	**EncTable[27].a**
$003524	**EncTable[12].a**	$003554	**EncTable[28].a**
$003527	**EncTable[13].a**	$003557	**EncTable[29].a**
$00352A	**EncTable[14].a**	$00355A	**EncTable[30].a**
$00352D	**EncTable[15].a**	$00355D	**EncTable[31].a**
$003530	**EncTable[16].a**	$003560	**EncTable[32].a**

Notes:

1. This is the configuration for Turbo PMAC Ultralites with filtered MACRO feedback

I*xx*05 to Motor[*x*].pMasterEnc

- If **Ix*x*05** is set to the default value of $35C0 or other address that will not have changing data, **Motor[*x*].pMasterEnc** can be set to the default value of **EncTable[0].a**, which typically also does not have changing data.

- Otherwise, set **Motor[*x*].pMasterEnc** to **EncTable[*n*].a** for the encoder conversion table entry ***n*** that matches the Turbo PMAC address of **Ix*x*05**.

I*xx*10 to Motor[*x*].pAbsPos

- If **I*xx*10** is set to 0, set **Motor[*x*].pAbsPos** to 0 (default) to disable absolute power-on position read.

- If **I*xx*10** is not set to 0, set **Motor[*x*].pAbsPos** to the Power PMAC address that matches the **I*xx*10** setting below. For absolute sensors other than serial encoders, refer to the Power PMAC manuals.

ACC-84x Turbo PMAC Serial Encoder Interface

The most common absolute position feedback for Turbo PMAC systems uses an ACC-84S, ACC-84B, or ACC-84E serial encoder interface board. These ACC-84*x* boards can also be used on the comparable Power PMAC systems, but for many serial encoder protocols, the built-in standard axis interface can process the serial encoder signal without need for an accessory board.

Turbo Clipper with ACC-84S to Power Clipper

Turbo Clipper **I*xx*10**	Power Clipper with ACC-84S **Motor[*x*].pAbsPos**	Power Clipper Standard Interface **Motor[*x*].pAbsPos**
$078800	**Acc84S[0].Chan[0].SerialEncDataA.a**	**Clipper[0].Chan[0].SerialEncDataA.a**
$078804	**Acc84S[0].Chan[1].SerialEncDataA.a**	**Clipper[0].Chan[1].SerialEncDataA.a**
$078808	**Acc84S[0].Chan[2].SerialEncDataA.a**	**Clipper[0].Chan[2].SerialEncDataA.a**
$07880C	**Acc84S[0].Chan[3].SerialEncDataA.a**	**Clipper[0].Chan[3].SerialEncDataA.a**
$078820	**Acc84S[1].Chan[0].SerialEncDataA.a**	**Clipper[1].Chan[0].SerialEncDataA.a**
$078824	**Acc84S[1].Chan[1].SerialEncDataA.a**	**Clipper[1].Chan[1].SerialEncDataA.a**
$078828	**Acc84S[1].Chan[2].SerialEncDataA.a**	**Clipper[1].Chan[2].SerialEncDataA.a**
$07882C	**Acc84S[1].Chan[3].SerialEncDataA.a**	**Clipper[1].Chan[3].SerialEncDataA.a**

Notes:

1. ACC-84S is not auto-identified by Power PMAC, so must be manually identified by setting **GateIo[*i*].PartNum** to 603936, **GateIo[*i*].PartType** to 8, **`save`**, and reset.
2. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

Turbo Brick with ACC-84B to Power Brick

Turbo Brick **I*xx*10**	Power Brick with ACC-84B **Motor[*x*].pAbsPos**	Power Brick Standard Interface **Motor[*x*].pAbsPos**
$078B20	**Acc84B[0].Chan[0].SerialEncDataA.a**	**PowerBrick[0].Chan[0].SerialEncDataA.a**
$078B24	**Acc84B[0].Chan[1].SerialEncDataA.a**	**PowerBrick[0].Chan[1].SerialEncDataA.a**
$078B28	**Acc84B[0].Chan[2].SerialEncDataA.a**	**PowerBrick[0].Chan[2].SerialEncDataA.a**
$078B2C	**Acc84B[0].Chan[3].SerialEncDataA.a**	**PowerBrick[0].Chan[3].SerialEncDataA.a**
$078B30	**Acc84B[1].Chan[0].SerialEncDataA.a**	**PowerBrick[1].Chan[0].SerialEncDataA.a**
$078B34	**Acc84B[1].Chan[1].SerialEncDataA.a**	**PowerBrick[1].Chan[1].SerialEncDataA.a**
$078B38	**Acc84B[1].Chan[2].SerialEncDataA.a**	**PowerBrick[1].Chan[2].SerialEncDataA.a**
$078B3C	**Acc84B[1].Chan[3].SerialEncDataA.a**	**PowerBrick[1].Chan[3].SerialEncDataA.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

Turbo UMAC with ACC-84E to Power UMAC

Turbo UMAC **I*xx*10**	Power UMAC with ACC-84E **Motor[*x*].pAbsPos**	Power UMAC Standard Interface **Motor[*x*].pAbsPos**
$078C00	**Acc84E[0].Chan[0].SerialEncDataA.a**	**Acc24E3[0].Chan[0].SerialEncDataA.a**
$078C04	**Acc84E[0].Chan[1].SerialEncDataA.a**	**Acc24E3[0].Chan[1].SerialEncDataA.a**
$078C08	**Acc84E[0].Chan[2].SerialEncDataA.a**	**Acc24E3[0].Chan[2].SerialEncDataA.a**
$078C0C	**Acc84E[0].Chan[3].SerialEncDataA.a**	**Acc24E3[0].Chan[3].SerialEncDataA.a**
$078D00	**Acc84E[1].Chan[0].SerialEncDataA.a**	**Acc24E3[1].Chan[0].SerialEncDataA.a**
$078D04	**Acc84E[1].Chan[1].SerialEncDataA.a**	**Acc24E3[1].Chan[1].SerialEncDataA.a**
$078D08	**Acc84E[1].Chan[2].SerialEncDataA.a**	**Acc24E3[1].Chan[2].SerialEncDataA.a**
$078D0C	**Acc84E[1].Chan[3].SerialEncDataA.a**	**Acc24E3[1].Chan[3].SerialEncDataA.a**
$078E00	**Acc84E[2].Chan[0].SerialEncDataA.a**	**Acc24E3[2].Chan[0].SerialEncDataA.a**
$078E04	**Acc84E[2].Chan[1].SerialEncDataA.a**	**Acc24E3[2].Chan[1].SerialEncDataA.a**
$078E08	**Acc84E[2].Chan[2].SerialEncDataA.a**	**Acc24E3[2].Chan[2].SerialEncDataA.a**
$078E0C	**Acc84E[2].Chan[3].SerialEncDataA.a**	**Acc24E3[2].Chan[3].SerialEncDataA.a**
$079C00	**Acc84E[4].Chan[0].SerialEncDataA.a**	**Acc24E3[3].Chan[0].SerialEncDataA.a**
$079C04	**Acc84E[4].Chan[1].SerialEncDataA.a**	**Acc24E3[3].Chan[1].SerialEncDataA.a**
$079C08	**Acc84E[4].Chan[2].SerialEncDataA.a**	**Acc24E3[3].Chan[2].SerialEncDataA.a**
$079C0C	**Acc84E[4].Chan[3].SerialEncDataA.a**	**Acc24E3[3].Chan[3].SerialEncDataA.a**
$079D00	**Acc84E[5].Chan[0].SerialEncDataA.a**	**Acc24E3[4].Chan[0].SerialEncDataA.a**
$079D04	**Acc84E[5].Chan[1].SerialEncDataA.a**	**Acc24E3[4].Chan[1].SerialEncDataA.a**
$079D08	**Acc84E[5].Chan[2].SerialEncDataA.a**	**Acc24E3[4].Chan[2].SerialEncDataA.a**
$079D0C	**Acc84E[5].Chan[3].SerialEncDataA.a**	**Acc24E3[4].Chan[3].SerialEncDataA.a**
$079E00	**Acc84E[6].Chan[0].SerialEncDataA.a**	**Acc24E3[5].Chan[0].SerialEncDataA.a**
$079E04	**Acc84E[6].Chan[1].SerialEncDataA.a**	**Acc24E3[5].Chan[1].SerialEncDataA.a**
$079E08	**Acc84E[6].Chan[2].SerialEncDataA.a**	**Acc24E3[5].Chan[2].SerialEncDataA.a**
$079E0C	**Acc84E[6].Chan[3].SerialEncDataA.a**	**Acc24E3[5].Chan[3].SerialEncDataA.a**
$07AC00	**Acc84E[8].Chan[0].SerialEncDataA.a**	**Acc24E3[6].Chan[0].SerialEncDataA.a**
$07AC04	**Acc84E[8].Chan[1].SerialEncDataA.a**	**Acc24E3[6].Chan[1].SerialEncDataA.a**
$07AC08	**Acc84E[8].Chan[2].SerialEncDataA.a**	**Acc24E3[6].Chan[2].SerialEncDataA.a**
$07AC0C	**Acc84E[8].Chan[3].SerialEncDataA.a**	**Acc24E3[6].Chan[3].SerialEncDataA.a**
$07AD00	**Acc84E[9].Chan[0].SerialEncDataA.a**	**Acc24E3[7].Chan[0].SerialEncDataA.a**
$07AD04	**Acc84E[9].Chan[1].SerialEncDataA.a**	**Acc24E3[7].Chan[1].SerialEncDataA.a**
$07AD08	**Acc84E[9].Chan[2].SerialEncDataA.a**	**Acc24E3[7].Chan[2].SerialEncDataA.a**
$07AD0C	**Acc84E[9].Chan[3].SerialEncDataA.a**	**Acc24E3[7].Chan[3].SerialEncDataA.a**
$07AE00	**Acc84E[10].Chan[0].SerialEncDataA.a**	**Acc24E3[8].Chan[0].SerialEncDataA.a**
$07AE04	**Acc84E[10].Chan[1].SerialEncDataA.a**	**Acc24E3[8].Chan[1].SerialEncDataA.a**
$07AE08	**Acc84E[10].Chan[2].SerialEncDataA.a**	**Acc24E3[8].Chan[2].SerialEncDataA.a**
$07AE0C	**Acc84E[10].Chan[3].SerialEncDataA.a**	**Acc24E3[8].Chan[3].SerialEncDataA.a**

Notes:

1. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.

Turbo UMAC with ACC-84E to CK3M with CK3W-AX*nnnn* Axis Interface

Turbo UMAC **I*xx*10**	CK3W Standard Interface **Motor[*x*].pAbsPos**
$078C00	**CK3WAX[0].Chan[0].SerialEncDataA.a**
$078C04	**CK3WAX[0].Chan[1].SerialEncDataA.a**
$078C08	**CK3WAX[0].Chan[2].SerialEncDataA.a**
$078C0C	**CK3WAX[0].Chan[3].SerialEncDataA.a**
$078D00	**CK3WAX[1].Chan[0].SerialEncDataA.a**
$078D04	**CK3WAX[1].Chan[1].SerialEncDataA.a**
$078D08	**CK3WAX[1].Chan[2].SerialEncDataA.a**
$078D0C	**CK3WAX[1].Chan[3].SerialEncDataA.a**
$078E00	**CK3WAX[2].Chan[0].SerialEncDataA.a**
$078E04	**CK3WAX[2].Chan[1].SerialEncDataA.a**
$078E08	**CK3WAX[2].Chan[2].SerialEncDataA.a**
$078E0C	**CK3WAX[2].Chan[3].SerialEncDataA.a**
$079C00	**CK3WAX[3].Chan[0].SerialEncDataA.a**
$079C04	**CK3WAX[3].Chan[1].SerialEncDataA.a**
$079C08	**CK3WAX[3].Chan[2].SerialEncDataA.a**
$079C0C	**CK3WAX[3].Chan[3].SerialEncDataA.a**
$079D00	**CK3WAX[4].Chan[0].SerialEncDataA.a**
$079D04	**CK3WAX[4].Chan[1].SerialEncDataA.a**
$079D08	**CK3WAX[4].Chan[2].SerialEncDataA.a**
$079D0C	**CK3WAX[4].Chan[3].SerialEncDataA.a**
$079E00	**CK3WAX[5].Chan[0].SerialEncDataA.a**
$079E04	**CK3WAX[5].Chan[1].SerialEncDataA.a**
$079E08	**CK3WAX[5].Chan[2].SerialEncDataA.a**
$079E0C	**CK3WAX[5].Chan[3].SerialEncDataA.a**
$07AC00	**CK3WAX[6].Chan[0].SerialEncDataA.a**
$07AC04	**CK3WAX[6].Chan[1].SerialEncDataA.a**
$07AC08	**CK3WAX[6].Chan[2].SerialEncDataA.a**
$07AC0C	**CK3WAX[6].Chan[3].SerialEncDataA.a**
$07AD00	**CK3WAX[7].Chan[0].SerialEncDataA.a**
$07AD04	**CK3WAX[7].Chan[1].SerialEncDataA.a**
$07AD08	**CK3WAX[7].Chan[2].SerialEncDataA.a**
$07AD0C	**CK3WAX[7].Chan[3].SerialEncDataA.a**
$07AE00	**CK3WAX[8].Chan[0].SerialEncDataA.a**
$07AE04	**CK3WAX[8].Chan[1].SerialEncDataA.a**
$07AE08	**CK3WAX[8].Chan[2].SerialEncDataA.a**
$07AE0C	**CK3WAX[8].Chan[3].SerialEncDataA.a**

Notes:

1. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

I*xx*95 to Motor[*x*].AbsPosFormat, AbsPosSf

This parameter tells the PMAC how to interpret the absolute position data read from the specified register. It is only used if an absolute position read is specified (**I*xx*10** > 0 in Turbo PMAC, **Motor[*x*].pAbsPos** > 0 in Power PMAC).

ACC-84x Turbo PMAC Serial Encoder Interface

When using a Turbo PMAC with an ACC-84*x* board to read the absolute position from a serial encoder, **I*xx*95** is set to $*nn*0000, where the two hex digits $*nn* represent the number of bits to read, and whether to interpret the value as unsigned or signed.

- If $*nn* is in the range $08 to $30 (8 to 48), it specifies the number of bits directly, and instructs Turbo PMAC to interpret these bits as an unsigned value.
    - If the Power PMAC system also uses an ACC-84*x* interface, set **Motor[*x*].AbsPosFormat** to $0008*nn*08 for the same formatting, and **Motor[*x*].AbsPosSf** to 1.0 to scale in LSBs.
    - If the Power PMAC uses the built-in standard interface, set **Motor[*x*].AbsPosFormat** to $0000*nn*00 for the same formatting, and **Motor[*x*].AbsPosSf** to 1.0 to scale in LSBs.
- If $*nn* is in the range $88 to $B0, then $*mm* = ($*nn* - $80) specifies the number of bits (8 to 48), and instructs Turbo PMAC to interpret these bits as a signed value.
    - If the Power PMAC system also uses an ACC-84*x* interface, set **Motor[*x*].AbsPosFormat** to $0108*mm*08 for the same formatting, and **Motor[*x*].AbsPosSf** to 1.0 to scale in LSBs.
    - If the Power PMAC uses the built-in standard interface, set **Motor[*x*].AbsPosFormat** to $0100*mm*00 for the same formatting, and **Motor[*x*].AbsPosSf** to 1.0 to scale in LSBs.

I*xx*25 to Motor[*x*].pEncStatus, pCaptFlag, etc.

In Turbo PMAC, I*xx*25 specifies the address of the register for the encoder position-capture flags.

If I*xx*42 is set to its default value of 0, I*xx*25 also specifies the register for the amplifier fault input.

If I*xx*43 is set to its default value of 0, I*xx*25 also specifies the register for the hardware overtravel limit inputs.

- Set **Motor[*x*].pEncStatus** and **Motor[*x*].pCaptFlag** to the Power PMAC address that matches the **I*xx*25** setting, from one of the tables below.

Turbo Clipper to Power Clipper

Turbo Clipper Flags to Power Clipper Flags

Turbo Clipper **I*xx*25**	Power Clipper **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**
$078000	**Clipper[0].Chan[0].Status.a**
$078008	**Clipper[0].Chan[1].Status.a**
$078010	**Clipper[0].Chan[2].Status.a**
$078018	**Clipper[0].Chan[3].Status.a**
$078100	**Clipper[1].Chan[0].Status.a**
$078108	**Clipper[1].Chan[1].Status.a**
$078110	**Clipper[1].Chan[2].Status.a**
$078118	**Clipper[1].Chan[3].Status.a**

Notes:

1. If Turbo Clipper uses default configuration, Power Clipper can use default configuration.
2. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
3. Clipper **Status.a** addresses can also be entered as **CaptFlag.a**, but will report back as **Status.a**.

- If there is no ACC-51S used for the channel, set **Motor[*x*].EncType** to 5, **Motor[*x*]CaptPosLeftShift** to 8, **Motor[*x*]CaptPosRightShift** to 8, and **Motor[*x*].CaptPosRound** to 1.

- If there is an ACC-51S used for the channel, set **Motor[*x*].EncType** to 6, **Motor[*x*]CaptPosLeftShift** to 4, **Motor[*x*]CaptPosRightShift** to 0, and **Motor[*x*].CaptPosRound** to 0.

- Set **Motor[*x*].CaptFlagBit** to 20.

- Set **Motor[*x*].CaptFlagInvert** and **Motor[*x*].CaptToggle** to 0 (defaults).

Turbo Brick to Power Brick

Turbo Brick Flags to Power Brick Flags

Turbo Brick **I*xx*25**	Power Brick **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**
$078000	**PowerBrick[0].Chan[0].Status.a**
$078008	**PowerBrick[0].Chan[1].Status.a**
$078010	**PowerBrick[0].Chan[2].Status.a**
$078018	**PowerBrick[0].Chan[3].Status.a**
$078100	**PowerBrick[1].Chan[0].Status.a**
$078108	**PowerBrick[1].Chan[1].Status.a**
$078110	**PowerBrick[1].Chan[2].Status.a**
$078118	**PowerBrick[1].Chan[3].Status.a**

Notes:

1. If Turbo Brick uses default configuration, Power Brick can use default configuration.
2. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
3. Brick **Status.a** addresses can also be entered as **CaptFlag.a**, but will report back as **Status.a**.

- If there is no sine-encoder interpolator used for the channel, set **Motor[*x*].EncType** to 5, **Motor[*x*]CaptPosLeftShift** to 8, **Motor[*x*]CaptPosRightShift** to 8, and **Motor[*x*].CaptPosRound** to 1.

- If there is an ACC-51B used for the channel, set **Motor[*x*].EncType** to 6, **Motor[*x*]CaptPosLeftShift** to 4, **Motor[*x*]CaptPosRightShift** to 0, and **Motor[*x*].CaptPosRound** to 0.

- If there is an auto-correcting interpolator used for the channel, set **Motor[*x*].EncType** to 7, **Motor[*x*]CaptPosLeftShift** to 6, **Motor[*x*]CaptPosRightShift** to 0, and **Motor[*x*].CaptPosRound** to 0.

- Set **Motor[*x*].CaptFlagBit** to 20.

- Set **Motor[*x*].CaptFlagInvert** and **Motor[*x*].CaptToggle** to 0 (defaults).

Turbo UMAC to Power UMAC or CK3M

Turbo UMAC Flags to Power UMAC, CK3M Flags

Turbo UMAC **I*xx*25**	Power UMAC ACC-24E2*x* **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**	Power UMAC ACC-24E3 **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**
$078200	**Acc24E2*x*[4].Chan[0].Status.a**	**Acc24E3[0].Chan[0].Status.a**	**CK3WAX[0].Chan[0].Status.a**
$078208	**Acc24E2*x*[4].Chan[1].Status.a**	**Acc24E3[0].Chan[1].Status.a**	**CK3WAX[0].Chan[1].Status.a**
$078210	**Acc24E2*x*[4].Chan[2].Status.a**	**Acc24E3[0].Chan[2].Status.a**	**CK3WAX[0].Chan[2].Status.a**
$078218	**Acc24E2*x*[4].Chan[3].Status.a**	**Acc24E3[0].Chan[3].Status.a**	**CK3WAX[0].Chan[3].Status.a**
$078300	**Acc24E2*x*[6].Chan[0].Status.a**	**Acc24E3[1].Chan[0].Status.a**	**CK3WAX[1].Chan[0].Status.a**
$078308	**Acc24E2*x*[6].Chan[1].Status.a**	**Acc24E3[1].Chan[1].Status.a**	**CK3WAX[1].Chan[1].Status.a**
$078310	**Acc24E2*x*[6].Chan[2].Status.a**	**Acc24E3[1].Chan[2].Status.a**	**CK3WAX[1].Chan[2].Status.a**
$078318	**Acc24E2*x*[6].Chan[3].Status.a**	**Acc24E3[1].Chan[3].Status.a**	**CK3WAX[1].Chan[3].Status.a**
$079200	**Acc24E2*x*[8].Chan[0].Status.a**	**Acc24E3[2].Chan[0].Status.a**	**CK3WAX[2].Chan[0].Status.a**
$079208	**Acc24E2*x*[8].Chan[1].Status.a**	**Acc24E3[2].Chan[1].Status.a**	**CK3WAX[2].Chan[1].Status.a**
$079210	**Acc24E2*x*[8].Chan[2].Status.a**	**Acc24E3[2].Chan[2].Status.a**	**CK3WAX[2].Chan[2].Status.a**
$079218	**Acc24E2*x*[8].Chan[3].Status.a**	**Acc24E3[2].Chan[3].Status.a**	**CK3WAX[2].Chan[3].Status.a**
$079300	**Acc24E2*x*[10].Chan[0].Status.a**	**Acc24E3[3].Chan[0].Status.a**	**CK3WAX[3].Chan[0].Status.a**
$079308	**Acc24E2*x*[10].Chan[1].Status.a**	**Acc24E3[3].Chan[1].Status.a**	**CK3WAX[3].Chan[1].Status.a**
$079310	**Acc24E2*x*[10].Chan[2].Status.a**	**Acc24E3[3].Chan[2].Status.a**	**CK3WAX[3].Chan[2].Status.a**
$079318	**Acc24E2*x*[10].Chan[3].Status.a**	**Acc24E3[3].Chan[3].Status.a**	**CK3WAX[3].Chan[3].Status.a**
$07A200	**Acc24E2*x*[12].Chan[0].Status.a**	**Acc24E3[4].Chan[0].Status.a**	**CK3WAX[4].Chan[0].Status.a**
$07A208	**Acc24E2*x*[12].Chan[1].Status.a**	**Acc24E3[4].Chan[1].Status.a**	**CK3WAX[4].Chan[1].Status.a**
$07A210	**Acc24E2*x*[12].Chan[2].Status.a**	**Acc24E3[4].Chan[2].Status.a**	**CK3WAX[4].Chan[2].Status.a**
$07A218	**Acc24E2*x*[12].Chan[3].Status.a**	**Acc24E3[4].Chan[3].Status.a**	**CK3WAX[4].Chan[3].Status.a**
$07A300	**Acc24E2*x*[14].Chan[0].Status.a**	**Acc24E3[5].Chan[0].Status.a**	**CK3WAX[5].Chan[0].Status.a**
$07A308	**Acc24E2*x*[14].Chan[1].Status.a**	**Acc24E3[5].Chan[1].Status.a**	**CK3WAX[5].Chan[1].Status.a**
$07A310	**Acc24E2*x*[14].Chan[2].Status.a**	**Acc24E3[5].Chan[2].Status.a**	**CK3WAX[5].Chan[2].Status.a**
$07A318	**Acc24E2*x*[14].Chan[3].Status.a**	**Acc24E3[5].Chan[3].Status.a**	**CK3WAX[5].Chan[3].Status.a**
$07B200	**Acc24E2*x*[16].Chan[0].Status.a**	**Acc24E3[6].Chan[0].Status.a**	**CK3WAX[6].Chan[0].Status.a**
$07B208	**Acc24E2*x*[16].Chan[1].Status.a**	**Acc24E3[6].Chan[1].Status.a**	**CK3WAX[6].Chan[1].Status.a**
$07B210	**Acc24E2*x*[16].Chan[2].Status.a**	**Acc24E3[6].Chan[2].Status.a**	**CK3WAX[6].Chan[2].Status.a**
$07B218	**Acc24E2*x*[16].Chan[3].Status.a**	**Acc24E3[6].Chan[3].Status.a**	**CK3WAX[6].Chan[3].Status.a**
$07B300	**Acc24E2*x*[18].Chan[0].Status.a**	**Acc24E3[7].Chan[0].Status.a**	**CK3WAX[7].Chan[0].Status.a**
$07B308	**Acc24E2*x*[18].Chan[1].Status.a**	**Acc24E3[7].Chan[1].Status.a**	**CK3WAX[7].Chan[1].Status.a**
$07B310	**Acc24E2*x*[18].Chan[2].Status.a**	**Acc24E3[7].Chan[2].Status.a**	**CK3WAX[7].Chan[2].Status.a**
$07B318	**Acc24E2*x*[18].Chan[3].Status.a**	**Acc24E3[7].Chan[3].Status.a**	**CK3WAX[7].Chan[3].Status.a**

Notes:

1. If Turbo UMAC uses default configuration, Power UMAC can use default configuration.
2. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
3. If an ACC-51E is used for position capture, you can substitute **Acc51E[*i*]** for **Acc24E2*x*[*i*]** here.
4. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
5. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
6. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
7. **Status.a** addresses can also be entered as **CaptFlag.a**, but will report back as **Status.a**.

- If ACC-24E2*x* is used for position capture for the motor, set **Motor[*x*].EncType** to 2. This will automatically set:
    - **Motor[*x*].CaptFlagBit** to 19
    - **Motor[*x*]CaptPosLeftShift** to 9
    - **Motor[*x*]CaptPosRightShift** to 8
    - **Motor[*x*].CaptPosRound** to 1

- If ACC-51E is used for position capture for the motor, set **Motor[*x*].EncType** to 3 This will automatically set:
    - **Motor[*x*].CaptFlagBit** to 19
    - **Motor[*x*]CaptPosLeftShift** to 10
    - **Motor[*x*]CaptPosRightShift** to 8
    - **Motor[*x*].CaptPosRound** to 1

- If ACC-24E3 or CK3W-AX*nnnn* with quadrature feedback is used for position capture for the motor, set **Motor[*x*].EncType** to 5. This will automatically set:
    - **Motor[*x*].CaptFlagBit** to 20
    - **Motor[*x*]CaptPosLeftShift** to 8
    - **Motor[*x*]CaptPosRightShift** to 8
    - **Motor[*x*].CaptPosRound** to 1

- If ACC-24E3 or CK3W-AX*nnnn* with a standard sine-encoder interpolator is used for position capture for the motor, set **Motor[*x*].EncType** to 6. This will automatically set:
    - **Motor[*x*].CaptFlagBit** to 20
    - **Motor[*x*]CaptPosLeftShift** to 4
    - **Motor[*x*]CaptPosRightShift** to 0
    - **Motor[*x*].CaptPosRound** to 0

- If ACC-24E3 with an auto-correcting interpolator used for the position capture for the motor, set **Motor[*x*].EncType** to 7. This will automatically set:
    - **Motor[*x*].CaptFlagBit** to 20
    - **Motor[*x*]CaptPosLeftShift** to 6
    - **Motor[*x*]CaptPosRightShift** to 0
    - **Motor[*x*].CaptPosRound** to 0

- In all cases set **Motor[*x*].CaptFlagInvert** and **Motor[*x*].CaptToggle** to 0 (defaults).

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Input Flags to Power PMAC MACRO Input Flags

Turbo PMAC **I*xx*25**	Power PMAC ACC-5E **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**	Power PMAC ACC-5E3 **Motor[*x*].pEncStatus**, **pCaptFlag**; possibly **pAmpFault**, **pLimits**
$003440	**Acc5E[0].Macro[0][3].a**	**Acc5E3[0].MacroInA[0][3].a**
$003441	**Acc5E[0].Macro[1][3].a**	**Acc5E3[0].MacroInA[1][3].a**
$003444	**Acc5E[0].Macro[4][3].a**	**Acc5E3[0].MacroInA[4][3].a**
$003445	**Acc5E[0].Macro[5][3].a**	**Acc5E3[0].MacroInA[5][3].a**
$003448	**Acc5E[0].Macro[8][3].a**	**Acc5E3[0].MacroInA[8][3].a**
$003449	**Acc5E[0].Macro[9][3].a**	**Acc5E3[0].MacroInA[9][3].a**
$00344C	**Acc5E[0].Macro[12][3].a**	**Acc5E3[0].MacroInA[12][3].a**
$00344D	**Acc5E[0].Macro[13][3].a**	**Acc5E3[0].MacroInA[13][3].a**
$003450	**Acc5E[1].Macro[0][3].a**	**Acc5E3[0].MacroInB[0][3].a**
$003451	**Acc5E[1].Macro[1][3].a**	**Acc5E3[0].MacroInB[1][3].a**
$003454	**Acc5E[1].Macro[4][3].a**	**Acc5E3[0].MacroInB[4][3].a**
$003455	**Acc5E[1].Macro[5][3].a**	**Acc5E3[0].MacroInB[5][3].a**
$003458	**Acc5E[1].Macro[8][3].a**	**Acc5E3[0].MacroInB[8][3].a**
$003459	**Acc5E[1].Macro[9][3].a**	**Acc5E3[0].MacroInB[9][3].a**
$00345C	**Acc5E[1].Macro[12][3].a**	**Acc5E3[0].MacroInB[12][3].a**
$00345D	**Acc5E[1].Macro[13][3].a**	**Acc5E3[0].MacroInB[13][3].a**
$003460	**Acc5E[2].Macro[0][3].a**	**Acc5E3[1].MacroInA[0][3].a**
$003461	**Acc5E[2].Macro[1][3].a**	**Acc5E3[1].MacroInA[1][3].a**
$003464	**Acc5E[2].Macro[4][3].a**	**Acc5E3[1].MacroInA[4][3].a**
$003465	**Acc5E[2].Macro[5][3].a**	**Acc5E3[1].MacroInA[5][3].a**
$003468	**Acc5E[2].Macro[8][3].a**	**Acc5E3[1].MacroInA[8][3].a**
$003469	**Acc5E[2].Macro[9][3].a**	**Acc5E3[1].MacroInA[9][3].a**
$00346C	**Acc5E[2].Macro[12][3].a**	**Acc5E3[1].MacroInA[12][3].a**
$00346D	**Acc5E[2].Macro[13][3].a**	**Acc5E3[1].MacroInA[13][3].a**
$003470	**Acc5E[3].Macro[0][3].a**	**Acc5E3[1].MacroInB[0][3].a**
$003471	**Acc5E[3].Macro[1][3].a**	**Acc5E3[1].MacroInB[1][3].a**
$003474	**Acc5E[3].Macro[4][3].a**	**Acc5E3[1].MacroInB[4][3].a**
$003475	**Acc5E[3].Macro[5][3].a**	**Acc5E3[1].MacroInB[5][3].a**
$003478	**Acc5E[3].Macro[8][3].a**	**Acc5E3[1].MacroInB[8][3].a**
$003479	**Acc5E[3].Macro[9][3].a**	**Acc5E3[1].MacroInB[9][3].a**
$00347C	**Acc5E[3].Macro[12][3].a**	**Acc5E3[1].MacroInB[12][3].a**
$00347D	**Acc5E[3].Macro[13][3].a**	**Acc5E3[1].MacroInB[13][3].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

- Set **Motor[*x*].EncType** to 4 when the motor is controlled through a MACRO slave device.

- Set **Motor[*x*].EncType** to 12 when the motor is controlled through a PMAC acting as a MACRO auxiliary slave device.

- Set **Motor[*x*].CaptFlagBit** to 19, **Motor[*x*]CaptPosLeftShift** to 13, **Motor[*x*]CaptPosRightShift** to 0, and **Motor[*x*].CaptPosRound** to 1 for MACRO interfaces.

- Set **Motor[*x*].CaptFlagInvert** and **Motor[*x*].CaptToggle** to 0 for MACRO interfaces.

Turbo PMAC MACRO Output Flags to Power PMAC MACRO Output Flags

Turbo PMAC **I*xx*25**	Power PMAC ACC-5E **Motor[*x*].pEncCtrl**	Power PMAC ACC-5E3 **Motor[*x*].pEncCtrl**
$003440	**Acc5E[0].Macro[0][3].a**	**Acc5E3[0].MacroOutA[0][3].a**
$003441	**Acc5E[0].Macro[1][3].a**	**Acc5E3[0].MacroOutA[1][3].a**
$003444	**Acc5E[0].Macro[4][3].a**	**Acc5E3[0].MacroOutA[4][3].a**
$003445	**Acc5E[0].Macro[5][3].a**	**Acc5E3[0].MacroOutA[5][3].a**
$003448	**Acc5E[0].Macro[8][3].a**	**Acc5E3[0].MacroOutA[8][3].a**
$003449	**Acc5E[0].Macro[9][3].a**	**Acc5E3[0].MacroOutA[9][3].a**
$00344C	**Acc5E[0].Macro[12][3].a**	**Acc5E3[0].MacroOutA[12][3].a**
$00344D	**Acc5E[0].Macro[13][3].a**	**Acc5E3[0].MacroOutA[13][3].a**
$003450	**Acc5E[1].Macro[0][3].a**	**Acc5E3[0].MacroOutB[0][3].a**
$003451	**Acc5E[1].Macro[1][3].a**	**Acc5E3[0].MacroOutB[1][3].a**
$003454	**Acc5E[1].Macro[4][3].a**	**Acc5E3[0].MacroOutB[4][3].a**
$003455	**Acc5E[1].Macro[5][3].a**	**Acc5E3[0].MacroOutB[5][3].a**
$003458	**Acc5E[1].Macro[8][3].a**	**Acc5E3[0].MacroOutB[8][3].a**
$003459	**Acc5E[1].Macro[9][3].a**	**Acc5E3[0].MacroOutB[9][3].a**
$00345C	**Acc5E[1].Macro[12][3].a**	**Acc5E3[0].MacroOutB[12][3].a**
$00345D	**Acc5E[1].Macro[13][3].a**	**Acc5E3[0].MacroOutB[13][3].a**
$003460	**Acc5E[2].Macro[0][3].a**	**Acc5E3[1].MacroOutA[0][3].a**
$003461	**Acc5E[2].Macro[1][3].a**	**Acc5E3[1].MacroOutA[1][3].a**
$003464	**Acc5E[2].Macro[4][3].a**	**Acc5E3[1].MacroOutA[4][3].a**
$003465	**Acc5E[2].Macro[5][3].a**	**Acc5E3[1].MacroOutA[5][3].a**
$003468	**Acc5E[2].Macro[8][3].a**	**Acc5E3[1].MacroOutA[8][3].a**
$003469	**Acc5E[2].Macro[9][3].a**	**Acc5E3[1].MacroOutA[9][3].a**
$00346C	**Acc5E[2].Macro[12][3].a**	**Acc5E3[1].MacroOutA[12][3].a**
$00346D	**Acc5E[2].Macro[13][3].a**	**Acc5E3[1].MacroOutA[13][3].a**
$003470	**Acc5E[3].Macro[0][3].a**	**Acc5E3[1].MacroOutB[0][3].a**
$003471	**Acc5E[3].Macro[1][3].a**	**Acc5E3[1].MacroOutB[1][3].a**
$003474	**Acc5E[3].Macro[4][3].a**	**Acc5E3[1].MacroOutB[4][3].a**
$003475	**Acc5E[3].Macro[5][3].a**	**Acc5E3[1].MacroOutB[5][3].a**
$003478	**Acc5E[3].Macro[8][3].a**	**Acc5E3[1].MacroOutB[8][3].a**
$003479	**Acc5E[3].Macro[9][3].a**	**Acc5E3[1].MacroOutB[9][3].a**
$00347C	**Acc5E[3].Macro[12][3].a**	**Acc5E3[1].MacroOutB[12][3].a**
$00347D	**Acc5E[3].Macro[13][3].a**	**Acc5E3[1].MacroOutB[13][3].a**

Notes:

1. **Motor[*x*].pEncCtrl** is only used for MACRO interfaces.
2. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
3. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

I*xx*24, I*xx*42 to Motor[*x*].pAmpFault

- If **I*xx*24** bit 20 (value $100000) is set to 1, set **Motor[*x*].pAmpFault** to 0 to disable amplifier fault input functionality.

- If **I*xx*24** bit 20 (value $100000) is set to 0 and **I*xx*42** is set to 0, set **Motor[*x*].pAmpFault** to same address as **Motor[*x*].pEncStatus** (from **I*xx*25**) for equivalent operation.

- If **I*xx*24** bit 20 (value $100000) is set to 0 and **I*xx*42** is not set to 0, set **Motor[*x*].pAmpFault** to matching Power PMAC address for **I*xx*42**.

Turbo Clipper to Power Clipper

Turbo Clipper Amp Fault Input Flag to Power Clipper

Turbo Clipper **I*xx*25**, **I*xx*42**	Power Clipper **Motor[*x*].pAmpFault**
$078000	**Clipper[0].Chan[0].Status.a**
$078008	**Clipper[0].Chan[1].Status.a**
$078010	**Clipper[0].Chan[2].Status.a**
$078018	**Clipper[0].Chan[3].Status.a**
$078100	**Clipper[1].Chan[0].Status.a**
$078108	**Clipper[1].Chan[1].Status.a**
$078110	**Clipper[1].Chan[2].Status.a**
$078118	**Clipper[1].Chan[3].Status.a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **Status.a** addresses can also be entered as **AmpFault.a**, but will report back as **Status.a**.

- Set **Motor[*x*].AmpFaultBit** to 7 for Power Clipper boards.

- Set **Motor[*x*].AmpFaultLevel** equal to value of bit 23 ($800000) of **I*xx*24**.

Turbo Brick to Power Brick

Turbo Brick Amp Fault Input Flag to Power Brick

Turbo Brick **I*xx*25**, **I*xx*42**	Power Brick **Motor[*x*]. pAmpFault**
$078000	**PowerBrick[0].Chan[0].Status.a**
$078008	**PowerBrick[0].Chan[1].Status.a**
$078010	**PowerBrick[0].Chan[2].Status.a**
$078018	**PowerBrick[0].Chan[3].Status.a**
$078100	**PowerBrick[1].Chan[0].Status.a**
$078108	**PowerBrick[1].Chan[1].Status.a**
$078110	**PowerBrick[1].Chan[2].Status.a**
$078118	**PowerBrick[1].Chan[3].Status.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
2. Brick **Status.a** addresses can also be entered as **AmpFault.a**, but will report back as **Status.a**.

- Set **Motor[*x*].AmpFaultBit** to 7 for Power Brick systems.

- Set **Motor[*x*].AmpFaultLevel** equal to value of bit 23 ($800000) of **I*xx*24**.

Turbo UMAC to Power UMAC

Turbo UMAC Amp Fault Input Flag to Power UMAC or CK3M

Turbo UMAC **I*xx*25**, **I*xx*42**	Power UMAC ACC-24E2*x* **Motor[*x*].pAmpFault**	Power UMAC ACC-24E3 **Motor[*x*].pAmpFault**	CK3M CK3W-AX*nnnn* **Motor[*x*].pAmpFault**
$078200	**Acc24E2*x*[4].Chan[0].Status.a**	**Acc24E3[0].Chan[0].Status.a**	**CK3WAX[0].Chan[0].Status.a**
$078208	**Acc24E2*x*[4].Chan[1].Status.a**	**Acc24E3[0].Chan[1].Status.a**	**CK3WAX[0].Chan[1].Status.a**
$078210	**Acc24E2*x*[4].Chan[2].Status.a**	**Acc24E3[0].Chan[2].Status.a**	**CK3WAX[0].Chan[2].Status.a**
$078218	**Acc24E2*x*[4].Chan[3].Status.a**	**Acc24E3[0].Chan[3].Status.a**	**CK3WAX[0].Chan[3].Status.a**
$078300	**Acc24E2*x*[6].Chan[0].Status.a**	**Acc24E3[1].Chan[0].Status.a**	**CK3WAX[1].Chan[0].Status.a**
$078308	**Acc24E2*x*[6].Chan[1].Status.a**	**Acc24E3[1].Chan[1].Status.a**	**CK3WAX[1].Chan[1].Status.a**
$078310	**Acc24E2*x*[6].Chan[2].Status.a**	**Acc24E3[1].Chan[2].Status.a**	**CK3WAX[1].Chan[2].Status.a**
$078318	**Acc24E2*x*[6].Chan[3].Status.a**	**Acc24E3[1].Chan[3].Status.a**	**CK3WAX[1].Chan[3].Status.a**
$079200	**Acc24E2*x*[8].Chan[0].Status.a**	**Acc24E3[2].Chan[0].Status.a**	**CK3WAX[2].Chan[0].Status.a**
$079208	**Acc24E2*x*[8].Chan[1].Status.a**	**Acc24E3[2].Chan[1].Status.a**	**CK3WAX[2].Chan[1].Status.a**
$079210	**Acc24E2*x*[8].Chan[2].Status.a**	**Acc24E3[2].Chan[2].Status.a**	**CK3WAX[2].Chan[2].Status.a**
$079218	**Acc24E2*x*[8].Chan[3].Status.a**	**Acc24E3[2].Chan[3].Status.a**	**CK3WAX[2].Chan[3].Status.a**
$079300	**Acc24E2*x*[10].Chan[0].Status.a**	**Acc24E3[3].Chan[0].Status.a**	**CK3WAX[3].Chan[0].Status.a**
$079308	**Acc24E2*x*[10].Chan[1].Status.a**	**Acc24E3[3].Chan[1].Status.a**	**CK3WAX[3].Chan[1].Status.a**
$079310	**Acc24E2*x*[10].Chan[2].Status.a**	**Acc24E3[3].Chan[2].Status.a**	**CK3WAX[3].Chan[2].Status.a**
$079318	**Acc24E2*x*[10].Chan[3].Status.a**	**Acc24E3[3].Chan[3].Status.a**	**CK3WAX[3].Chan[3].Status.a**
$07A200	**Acc24E2*x*[12].Chan[0].Status.a**	**Acc24E3[4].Chan[0].Status.a**	**CK3WAX[4].Chan[0].Status.a**
$07A208	**Acc24E2*x*[12].Chan[1].Status.a**	**Acc24E3[4].Chan[1].Status.a**	**CK3WAX[4].Chan[1].Status.a**
$07A210	**Acc24E2*x*[12].Chan[2].Status.a**	**Acc24E3[4].Chan[2].Status.a**	**CK3WAX[4].Chan[2].Status.a**
$07A218	**Acc24E2*x*[12].Chan[3].Status.a**	**Acc24E3[4].Chan[3].Status.a**	**CK3WAX[4].Chan[3].Status.a**
$07A300	**Acc24E2*x*[14].Chan[0].Status.a**	**Acc24E3[5].Chan[0].Status.a**	**CK3WAX[5].Chan[0].Status.a**
$07A308	**Acc24E2*x*[14].Chan[1].Status.a**	**Acc24E3[5].Chan[1].Status.a**	**CK3WAX[5].Chan[1].Status.a**
$07A310	**Acc24E2*x*[14].Chan[2].Status.a**	**Acc24E3[5].Chan[2].Status.a**	**CK3WAX[5].Chan[2].Status.a**
$07A318	**Acc24E2*x*[14].Chan[3].Status.a**	**Acc24E3[5].Chan[3].Status.a**	**CK3WAX[5].Chan[3].Status.a**
$07B200	**Acc24E2*x*[16].Chan[0].Status.a**	**Acc24E3[6].Chan[0].Status.a**	**CK3WAX[6].Chan[0].Status.a**
$07B208	**Acc24E2*x*[16].Chan[1].Status.a**	**Acc24E3[6].Chan[1].Status.a**	**CK3WAX[6].Chan[1].Status.a**
$07B210	**Acc24E2*x*[16].Chan[2].Status.a**	**Acc24E3[6].Chan[2].Status.a**	**CK3WAX[6].Chan[2].Status.a**
$07B218	**Acc24E2*x*[16].Chan[3].Status.a**	**Acc24E3[6].Chan[3].Status.a**	**CK3WAX[6].Chan[3].Status.a**
$07B300	**Acc24E2*x*[18].Chan[0].Status.a**	**Acc24E3[7].Chan[0].Status.a**	**CK3WAX[7].Chan[0].Status.a**
$07B308	**Acc24E2*x*[18].Chan[1].Status.a**	**Acc24E3[7].Chan[1].Status.a**	**CK3WAX[7].Chan[1].Status.a**
$07B310	**Acc24E2*x*[18].Chan[2].Status.a**	**Acc24E3[7].Chan[2].Status.a**	**CK3WAX[7].Chan[2].Status.a**
$07B318	**Acc24E2*x*[18].Chan[3].Status.a**	**Acc24E3[7].Chan[3].Status.a**	**CK3WAX[7].Chan[3].Status.a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
4. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
5. **Status.a** addresses can also be entered as **AmpFault.a**, but will report back as **Status.a**.

- Set **Motor[*x*].AmpFaultBit** to 23 for ACC-24E2*x* boards.

- Set **Motor[*x*].AmpFaultBit** to 7 for ACC-24E3 boards or CK3M CK3W-AX*nnnn* boards.

- Set **Motor[*x*].AmpFaultLevel** equal to value of bit 23 ($800000) of **I*xx*24**.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Amp Fault Input Flags to Power PMAC

Turbo PMAC **I*xx*25**, **I*xx*42**	Power PMAC ACC-5E **Motor[*x*].pAmpFault**	Power PMAC ACC-5E3 **Motor[*x*].pAmpFault**
$003440	**Acc5E[0].Macro[0][3].a**	**Acc5E3[0].MacroInA[0][3].a**
$003441	**Acc5E[0].Macro[1][3].a**	**Acc5E3[0].MacroInA[1][3].a**
$003444	**Acc5E[0].Macro[4][3].a**	**Acc5E3[0].MacroInA[4][3].a**
$003445	**Acc5E[0].Macro[5][3].a**	**Acc5E3[0].MacroInA[5][3].a**
$003448	**Acc5E[0].Macro[8][3].a**	**Acc5E3[0].MacroInA[8][3].a**
$003449	**Acc5E[0].Macro[9][3].a**	**Acc5E3[0].MacroInA[9][3].a**
$00344C	**Acc5E[0].Macro[12][3].a**	**Acc5E3[0].MacroInA[12][3].a**
$00344D	**Acc5E[0].Macro[13][3].a**	**Acc5E3[0].MacroInA[13][3].a**
$003450	**Acc5E[1].Macro[0][3].a**	**Acc5E3[0].MacroInB[0][3].a**
$003451	**Acc5E[1].Macro[1][3].a**	**Acc5E3[0].MacroInB[1][3].a**
$003454	**Acc5E[1].Macro[4][3].a**	**Acc5E3[0].MacroInB[4][3].a**
$003455	**Acc5E[1].Macro[5][3].a**	**Acc5E3[0].MacroInB[5][3].a**
$003458	**Acc5E[1].Macro[8][3].a**	**Acc5E3[0].MacroInB[8][3].a**
$003459	**Acc5E[1].Macro[9][3].a**	**Acc5E3[0].MacroInB[9][3].a**
$00345C	**Acc5E[1].Macro[12][3].a**	**Acc5E3[0].MacroInB[12][3].a**
$00345D	**Acc5E[1].Macro[13][3].a**	**Acc5E3[0].MacroInB[13][3].a**
$003460	**Acc5E[2].Macro[0][3].a**	**Acc5E3[1].MacroInA[0][3].a**
$003461	**Acc5E[2].Macro[1][3].a**	**Acc5E3[1].MacroInA[1][3].a**
$003464	**Acc5E[2].Macro[4][3].a**	**Acc5E3[1].MacroInA[4][3].a**
$003465	**Acc5E[2].Macro[5][3].a**	**Acc5E3[1].MacroInA[5][3].a**
$003468	**Acc5E[2].Macro[8][3].a**	**Acc5E3[1].MacroInA[8][3].a**
$003469	**Acc5E[2].Macro[9][3].a**	**Acc5E3[1].MacroInA[9][3].a**
$00346C	**Acc5E[2].Macro[12][3].a**	**Acc5E3[1].MacroInA[12][3].a**
$00346D	**Acc5E[2].Macro[13][3].a**	**Acc5E3[1].MacroInA[13][3].a**
$003470	**Acc5E[3].Macro[0][3].a**	**Acc5E3[1].MacroInB[0][3].a**
$003471	**Acc5E[3].Macro[1][3].a**	**Acc5E3[1].MacroInB[1][3].a**
$003474	**Acc5E[3].Macro[4][3].a**	**Acc5E3[1].MacroInB[4][3].a**
$003475	**Acc5E[3].Macro[5][3].a**	**Acc5E3[1].MacroInB[5][3].a**
$003478	**Acc5E[3].Macro[8][3].a**	**Acc5E3[1].MacroInB[8][3].a**
$003479	**Acc5E[3].Macro[9][3].a**	**Acc5E3[1].MacroInB[9][3].a**
$00347C	**Acc5E[3].Macro[12][3].a**	**Acc5E3[1].MacroInB[12][3].a**
$00347D	**Acc5E[3].Macro[13][3].a**	**Acc5E3[1].MacroInB[13][3].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

- Set **Motor[*x*].AmpFaultBit** to 23 for MACRO input signals.

- Set **Motor[*x*].AmpFaultLevel** equal to value of bit 23 ($800000) of **I*xx*24**.

I*xx*24, I*xx*42 to Motor[*x*].pAmpEnable

- If **I*xx*24** bit 16 (value $10000) is set to 1, set **Motor[*x*].pAmpEnable** to 0 to disable amplifier enable output functionality.

- If **I*xx*24** bit 16 (value $10000) is set to 0 and **I*xx*42** is set to 0, set **Motor[*x*].pAmpEnable** to same address as **Motor[*x*].pEncCtrl** for equivalent operation.

- If **I*xx*24** bit 16 (value $10000) is set to 0 and **I*xx*42** is not set to 0, set **Motor[*x*].pAmpEnable** to matching Power PMAC address for **I*xx*42**.

Turbo Clipper to Power Clipper

Turbo Clipper Flags to Power Clipper **pAmpEnable**

Turbo Clipper **I*xx*25**, **I*xx*42**	Power Clipper **Motor[*x*].pAmpEnable**
$078000	**Clipper[0].Chan[0].OutCtrl.a**
$078008	**Clipper[0].Chan[1].OutCtrl.a**
$078010	**Clipper[0].Chan[2].OutCtrl.a**
$078018	**Clipper[0].Chan[3].OutCtrl.a**
$078100	**Clipper[1].Chan[0].OutCtrl.a**
$078108	**Clipper[1].Chan[1].OutCtrl.a**
$078110	**Clipper[1].Chan[2].OutCtrl.a**
$078118	**Clipper[1].Chan[3].OutCtrl.a**

Notes:
1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **OutCtrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **OutCtrl.a**.

- Set **Motor[*x*].AmpEnableBit** to 8 for Power Clipper boards.

Turbo Brick to Power Brick

Turbo Brick Flags to Power Brick **pAmpEnable**

Turbo Brick **I*xx*25**, **I*xx*42**	Power Brick **Motor[*x*].pAmpEnable**
$078000	**PowerBrick[0].Chan[0].OutCtrl.a**
$078008	**PowerBrick[0].Chan[1].OutCtrl.a**
$078010	**PowerBrick[0].Chan[2].OutCtrl.a**
$078018	**PowerBrick[0].Chan[3].OutCtrl.a**
$078100	**PowerBrick[1].Chan[0].OutCtrl.a**
$078108	**PowerBrick[1].Chan[1].OutCtrl.a**
$078110	**PowerBrick[1].Chan[2].OutCtrl.a**
$078118	**PowerBrick[1].Chan[3].OutCtrl.a**

Notes:
1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
2. Clipper **OutCtrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **OutCtrl.a**.

- Set **Motor[*x*].AmpEnableBit** to 8 for Power Brick systems.

Turbo UMAC to Power UMAC

Turbo UMAC Flags to Power UMAC **pAmpEnable**

Turbo UMAC **I*xx*25**, **I*xx*42**	Power UMAC ACC-24E2*x* **Motor[*x*].pAmpEnable**	Power UMAC ACC-24E3 **Motor[*x*].pAmpEnable**	CK3M CK3W-AX*nnnn* **Motor[*x*].pAmpEnable**
$078200	**Acc24E2*x*[4].Chan[0].Ctrl.a**	**Acc24E3[0].Chan[0].OutCtrl.a**	**CK3WAX[0].Chan[0].OutCtrl.a**
$078208	**Acc24E2*x*[4].Chan[1].Ctrl.a**	**Acc24E3[0].Chan[1].OutCtrl.a**	**CK3WAX[0].Chan[1].OutCtrl.a**
$078210	**Acc24E2*x*[4].Chan[2].Ctrl.a**	**Acc24E3[0].Chan[2].OutCtrl.a**	**CK3WAX[0].Chan[2].OutCtrl.a**
$078218	**Acc24E2*x*[4].Chan[3].Ctrl.a**	**Acc24E3[0].Chan[3].OutCtrl.a**	**CK3WAX[0].Chan[3].OutCtrl.a**
$078300	**Acc24E2*x*[6].Chan[0].Ctrl.a**	**Acc24E3[1].Chan[0].OutCtrl.a**	**CK3WAX[1].Chan[0].OutCtrl.a**
$078308	**Acc24E2*x*[6].Chan[1].Ctrl.a**	**Acc24E3[1].Chan[1].OutCtrl.a**	**CK3WAX[1].Chan[1].OutCtrl.a**
$078310	**Acc24E2*x*[6].Chan[2].Ctrl.a**	**Acc24E3[1].Chan[2].OutCtrl.a**	**CK3WAX[1].Chan[2].OutCtrl.a**
$078318	**Acc24E2*x*[6].Chan[3].Ctrl.a**	**Acc24E3[1].Chan[3].OutCtrl.a**	**CK3WAX[1].Chan[3].OutCtrl.a**
$079200	**Acc24E2*x*[8].Chan[0].Ctrl.a**	**Acc24E3[2].Chan[0].OutCtrl.a**	**CK3WAX[2].Chan[0].OutCtrl.a**
$079208	**Acc24E2*x*[8].Chan[1].Ctrl.a**	**Acc24E3[2].Chan[1].OutCtrl.a**	**CK3WAX[2].Chan[1].OutCtrl.a**
$079210	**Acc24E2*x*[8].Chan[2].Ctrl.a**	**Acc24E3[2].Chan[2].OutCtrl.a**	**CK3WAX[2].Chan[2].OutCtrl.a**
$079218	**Acc24E2*x*[8].Chan[3].Ctrl.a**	**Acc24E3[2].Chan[3].OutCtrl.a**	**CK3WAX[2].Chan[3].OutCtrl.a**
$079300	**Acc24E2*x*[10].Chan[0].Ctrl.a**	**Acc24E3[3].Chan[0].OutCtrl.a**	**CK3WAX[3].Chan[0].OutCtrl.a**
$079308	**Acc24E2*x*[10].Chan[1].Ctrl.a**	**Acc24E3[3].Chan[1].OutCtrl.a**	**CK3WAX[3].Chan[1].OutCtrl.a**
$079310	**Acc24E2*x*[10].Chan[2].Ctrl.a**	**Acc24E3[3].Chan[2].OutCtrl.a**	**CK3WAX[3].Chan[2].OutCtrl.a**
$079318	**Acc24E2*x*[10].Chan[3].Ctrl.a**	**Acc24E3[3].Chan[3].OutCtrl.a**	**CK3WAX[3].Chan[3].OutCtrl.a**
$07A200	**Acc24E2*x*[12].Chan[0].Ctrl.a**	**Acc24E3[4].Chan[0].OutCtrl.a**	**CK3WAX[4].Chan[0].OutCtrl.a**
$07A208	**Acc24E2*x*[12].Chan[1].Ctrl.a**	**Acc24E3[4].Chan[1].OutCtrl.a**	**CK3WAX[4].Chan[1].OutCtrl.a**
$07A210	**Acc24E2*x*[12].Chan[2].Ctrl.a**	**Acc24E3[4].Chan[2].OutCtrl.a**	**CK3WAX[4].Chan[2].OutCtrl.a**
$07A218	**Acc24E2*x*[12].Chan[3].Ctrl.a**	**Acc24E3[4].Chan[3].OutCtrl.a**	**CK3WAX[4].Chan[3].OutCtrl.a**
$07A300	**Acc24E2*x*[14].Chan[0].Ctrl.a**	**Acc24E3[5].Chan[0].OutCtrl.a**	**CK3WAX[5].Chan[0].OutCtrl.a**
$07A308	**Acc24E2*x*[14].Chan[1].Ctrl.a**	**Acc24E3[5].Chan[1].OutCtrl.a**	**CK3WAX[5].Chan[1].OutCtrl.a**
$07A310	**Acc24E2*x*[14].Chan[2].Ctrl.a**	**Acc24E3[5].Chan[2].OutCtrl.a**	**CK3WAX[5].Chan[2].OutCtrl.a**
$07A318	**Acc24E2*x*[14].Chan[3].Ctrl.a**	**Acc24E3[5].Chan[3].OutCtrl.a**	**CK3WAX[5].Chan[3].OutCtrl.a**
$07B200	**Acc24E2*x*[16].Chan[0].Ctrl.a**	**Acc24E3[6].Chan[0].OutCtrl.a**	**CK3WAX[6].Chan[0].OutCtrl.a**
$07B208	**Acc24E2*x*[16].Chan[1].Ctrl.a**	**Acc24E3[6].Chan[1].OutCtrl.a**	**CK3WAX[6].Chan[1].OutCtrl.a**
$07B210	**Acc24E2*x*[16].Chan[2].Ctrl.a**	**Acc24E3[6].Chan[2].OutCtrl.a**	**CK3WAX[6].Chan[2].OutCtrl.a**
$07B218	**Acc24E2*x*[16].Chan[3].Ctrl.a**	**Acc24E3[6].Chan[3].OutCtrl.a**	**CK3WAX[6].Chan[3].OutCtrl.a**
$07B300	**Acc24E2*x*[18].Chan[0].Ctrl.a**	**Acc24E3[7].Chan[0].OutCtrl.a**	**CK3WAX[7].Chan[0].OutCtrl.a**
$07B308	**Acc24E2*x*[18].Chan[1].Ctrl.a**	**Acc24E3[7].Chan[1].OutCtrl.a**	**CK3WAX[7].Chan[1].OutCtrl.a**
$07B310	**Acc24E2*x*[18].Chan[2].Ctrl.a**	**Acc24E3[7].Chan[2].OutCtrl.a**	**CK3WAX[7].Chan[2].OutCtrl.a**
$07B318	**Acc24E2*x*[18].Chan[3].Ctrl.a**	**Acc24E3[7].Chan[3].OutCtrl.a**	**CK3WAX[].Chan[3].OutCtrl.a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
4. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
5. **Ctrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **Ctrl.a**.
6. **OutCtrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **OutCtrl.a**.

- Set **Motor[*x*].AmpEnableBit** to 22 for ACC-24E2*x* boards.

- Set **Motor[*x*].AmpEnableBit** to 8 for ACC-24E3 boards or CK3M CK3W-AX*nnnn* boards.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Flags to Power PMAC **pAmpEnable**

Turbo PMAC **I*xx*25**, **I*xx*42**	Power PMAC ACC-5E **Motor[*x*].pAmpEnable**	Power PMAC ACC-5E3 **Motor[*x*].pAmpEnable**
$003440	**Acc5E[0].Macro[0][3].a**	**Acc5E3[0].MacroOutA[0][3].a**
$003441	**Acc5E[0].Macro[1][3].a**	**Acc5E3[0].MacroOutA[1][3].a**
$003444	**Acc5E[0].Macro[4][3].a**	**Acc5E3[0].MacroOutA[4][3].a**
$003445	**Acc5E[0].Macro[5][3].a**	**Acc5E3[0].MacroOutA[5][3].a**
$003448	**Acc5E[0].Macro[8][3].a**	**Acc5E3[0].MacroOutA[8][3].a**
$003449	**Acc5E[0].Macro[9][3].a**	**Acc5E3[0].MacroOutA[9][3].a**
$00344C	**Acc5E[0].Macro[12][3].a**	**Acc5E3[0].MacroOutA[12][3].a**
$00344D	**Acc5E[0].Macro[13][3].a**	**Acc5E3[0].MacroOutA[13][3].a**
$003450	**Acc5E[1].Macro[0][3].a**	**Acc5E3[0].MacroOutB[0][3].a**
$003451	**Acc5E[1].Macro[1][3].a**	**Acc5E3[0].MacroOutB[1][3].a**
$003454	**Acc5E[1].Macro[4][3].a**	**Acc5E3[0].MacroOutB[4][3].a**
$003455	**Acc5E[1].Macro[5][3].a**	**Acc5E3[0].MacroOutB[5][3].a**
$003458	**Acc5E[1].Macro[8][3].a**	**Acc5E3[0].MacroOutB[8][3].a**
$003459	**Acc5E[1].Macro[9][3].a**	**Acc5E3[0].MacroOutB[9][3].a**
$00345C	**Acc5E[1].Macro[12][3].a**	**Acc5E3[0].MacroOutB[12][3].a**
$00345D	**Acc5E[1].Macro[13][3].a**	**Acc5E3[0].MacroOutB[13][3].a**
$003460	**Acc5E[2].Macro[0][3].a**	**Acc5E3[1].MacroOutA[0][3].a**
$003461	**Acc5E[2].Macro[1][3].a**	**Acc5E3[1].MacroOutA[1][3].a**
$003464	**Acc5E[2].Macro[4][3].a**	**Acc5E3[1].MacroOutA[4][3].a**
$003465	**Acc5E[2].Macro[5][3].a**	**Acc5E3[1].MacroOutA[5][3].a**
$003468	**Acc5E[2].Macro[8][3].a**	**Acc5E3[1].MacroOutA[8][3].a**
$003469	**Acc5E[2].Macro[9][3].a**	**Acc5E3[1].MacroOutA[9][3].a**
$00346C	**Acc5E[2].Macro[12][3].a**	**Acc5E3[1].MacroOutA[12][3].a**
$00346D	**Acc5E[2].Macro[13][3].a**	**Acc5E3[1].MacroOutA[13][3].a**
$003470	**Acc5E[3].Macro[0][3].a**	**Acc5E3[1].MacroOutB[0][3].a**
$003471	**Acc5E[3].Macro[1][3].a**	**Acc5E3[1].MacroOutB[1][3].a**
$003474	**Acc5E[3].Macro[4][3].a**	**Acc5E3[1].MacroOutB[4][3].a**
$003475	**Acc5E[3].Macro[5][3].a**	**Acc5E3[1].MacroOutB[5][3].a**
$003478	**Acc5E[3].Macro[8][3].a**	**Acc5E3[1].MacroOutB[8][3].a**
$003479	**Acc5E[3].Macro[9][3].a**	**Acc5E3[1].MacroOutB[9][3].a**
$00347C	**Acc5E[3].Macro[12][3].a**	**Acc5E3[1].MacroOutB[12][3].a**
$00347D	**Acc5E[3].Macro[13][3].a**	**Acc5E3[1].MacroOutB[13][3].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.
3. **Ctrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **Ctrl.a**.
4. **OutCtrl.a** addresses can also be entered as **AmpEna.a**, but will report back as **OutCtrl.a**.

- Set **Motor[*x*].AmpEnableBit** to 22 for MACRO outputs.

I*xx*24, I*xx*43 to Motor[*x*].pLimits

- If **I*xx*24** bit 17 (value $20000) is set to 1, set **Motor[*x*].pLimits** to 0 to disable hardware limits.

- If **I*xx*24** bit 17 (value $20000) is set to 0 and **I*xx*43** is set to 0, set **Motor[*x*].pLimits** to control register of same channel or MACRO node as **Motor[*x*].pEncStatus** (from **I*xx*25**) for equivalent operation.

- If **I*xx*24** bit 17 (value $20000) is set to 0 and **I*xx*43** is not set to 0, set **Motor[*x*].pLimits** to matching Power PMAC address for **I*xx*43**.

Turbo Clipper to Power Clipper

Turbo Clipper Limit Input Flags to Power Clipper

Turbo Clipper **I*xx*25**, **I*xx*43**	Power Clipper **Motor[*x*].pLimits**
$078000	**Clipper[0].Chan[0].Status.a**
$078008	**Clipper[0].Chan[1].Status.a**
$078010	**Clipper[0].Chan[2].Status.a**
$078018	**Clipper[0].Chan[3].Status.a**
$078100	**Clipper[1].Chan[0].Status.a**
$078108	**Clipper[1].Chan[1].Status.a**
$078110	**Clipper[1].Chan[2].Status.a**
$078118	**Clipper[1].Chan[3].Status.a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **Status.a** addresses can also be entered as **PlusLimit.a**, but will report back as **Status.a**.

- Set **Motor[*x*].LimitBits** to 9 for Power Clipper boards.

Turbo Brick to Power Brick

Turbo Brick Limit Input Flags to Power Brick

Turbo Brick **I*xx*25**, **I*xx*43**	Power Brick **Motor[*x*]. pLimits**
$078000	**PowerBrick[0].Chan[0].Status.a**
$078008	**PowerBrick[0].Chan[1].Status.a**
$078010	**PowerBrick[0].Chan[2].Status.a**
$078018	**PowerBrick[0].Chan[3].Status.a**
$078100	**PowerBrick[1].Chan[0].Status.a**
$078108	**PowerBrick[1].Chan[1].Status.a**
$078110	**PowerBrick[1].Chan[2].Status.a**
$078118	**PowerBrick[1].Chan[3].Status.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
2. Brick **Status.a** addresses can also be entered as **PlusLimit.a**, but will report back as **Status.a**.

- Set **Motor[*x*].LimitBits** to 9 for Power Brick systems.

Turbo UMAC to Power UMAC

Turbo UMAC Limit Input Flags to Power UMAC

Turbo UMAC **I*xx*25**, **I*xx*43**	Power UMAC ACC-24E2*x* **Motor[*x*].pLimits**	Power UMAC ACC-24E3 **Motor[*x*].pLimits**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pLimits**
$078200	**Acc24E2*x*[4].Chan[0].Status.a**	**Acc24E3[0].Chan[0].Status.a**	**CK3WAX[0].Chan[0].Status.a**
$078208	**Acc24E2*x*[4].Chan[1].Status.a**	**Acc24E3[0].Chan[1].Status.a**	**CK3WAX[0].Chan[1].Status.a**
$078210	**Acc24E2*x*[4].Chan[2].Status.a**	**Acc24E3[0].Chan[2].Status.a**	**CK3WAX[0].Chan[2].Status.a**
$078218	**Acc24E2*x*[4].Chan[3].Status.a**	**Acc24E3[0].Chan[3].Status.a**	**CK3WAX[0].Chan[3].Status.a**
$078300	**Acc24E2*x*[6].Chan[0].Status.a**	**Acc24E3[1].Chan[0].Status.a**	**CK3WAX[1].Chan[0].Status.a**
$078308	**Acc24E2*x*[6].Chan[1].Status.a**	**Acc24E3[1].Chan[1].Status.a**	**CK3WAX[1].Chan[1].Status.a**
$078310	**Acc24E2*x*[6].Chan[2].Status.a**	**Acc24E3[1].Chan[2].Status.a**	**CK3WAX[1].Chan[2].Status.a**
$078318	**Acc24E2*x*[6].Chan[3].Status.a**	**Acc24E3[1].Chan[3].Status.a**	**CK3WAX[1].Chan[3].Status.a**
$079200	**Acc24E2*x*[8].Chan[0].Status.a**	**Acc24E3[2].Chan[0].Status.a**	**CK3WAX[2].Chan[0].Status.a**
$079208	**Acc24E2*x*[8].Chan[1].Status.a**	**Acc24E3[2].Chan[1].Status.a**	**CK3WAX[2].Chan[1].Status.a**
$079210	**Acc24E2*x*[8].Chan[2].Status.a**	**Acc24E3[2].Chan[2].Status.a**	**CK3WAX[2].Chan[2].Status.a**
$079218	**Acc24E2*x*[8].Chan[3].Status.a**	**Acc24E3[2].Chan[3].Status.a**	**CK3WAX[2].Chan[3].Status.a**
$079300	**Acc24E2*x*[10].Chan[0].Status.a**	**Acc24E3[3].Chan[0].Status.a**	**CK3WAX[3].Chan[0].Status.a**
$079308	**Acc24E2*x*[10].Chan[1].Status.a**	**Acc24E3[3].Chan[1].Status.a**	**CK3WAX[3].Chan[1].Status.a**
$079310	**Acc24E2*x*[10].Chan[2].Status.a**	**Acc24E3[3].Chan[2].Status.a**	**CK3WAX[3].Chan[2].Status.a**
$079318	**Acc24E2*x*[10].Chan[3].Status.a**	**Acc24E3[3].Chan[3].Status.a**	**CK3WAX[3].Chan[3].Status.a**
$07A200	**Acc24E2*x*[12].Chan[0].Status.a**	**Acc24E3[4].Chan[0].Status.a**	**CK3WAX[4].Chan[0].Status.a**
$07A208	**Acc24E2*x*[12].Chan[1].Status.a**	**Acc24E3[4].Chan[1].Status.a**	**CK3WAX[4].Chan[1].Status.a**
$07A210	**Acc24E2*x*[12].Chan[2].Status.a**	**Acc24E3[4].Chan[2].Status.a**	**CK3WAX[4].Chan[2].Status.a**
$07A218	**Acc24E2*x*[12].Chan[3].Status.a**	**Acc24E3[4].Chan[3].Status.a**	**CK3WAX[4].Chan[3].Status.a**
$07A300	**Acc24E2*x*[14].Chan[0].Status.a**	**Acc24E3[5].Chan[0].Status.a**	**CK3WAX[5].Chan[0].Status.a**
$07A308	**Acc24E2*x*[14].Chan[1].Status.a**	**Acc24E3[5].Chan[1].Status.a**	**CK3WAX[5].Chan[1].Status.a**
$07A310	**Acc24E2*x*[14].Chan[2].Status.a**	**Acc24E3[5].Chan[2].Status.a**	**CK3WAX[5].Chan[2].Status.a**
$07A318	**Acc24E2*x*[14].Chan[3].Status.a**	**Acc24E3[5].Chan[3].Status.a**	**CK3WAX[5].Chan[3].Status.a**
$07B200	**Acc24E2*x*[16].Chan[0].Status.a**	**Acc24E3[6].Chan[0].Status.a**	**CK3WAX[6].Chan[0].Status.a**
$07B208	**Acc24E2*x*[16].Chan[1].Status.a**	**Acc24E3[6].Chan[1].Status.a**	**CK3WAX[6].Chan[1].Status.a**
$07B210	**Acc24E2*x*[16].Chan[2].Status.a**	**Acc24E3[6].Chan[2].Status.a**	**CK3WAX[6].Chan[2].Status.a**
$07B218	**Acc24E2*x*[16].Chan[3].Status.a**	**Acc24E3[6].Chan[3].Status.a**	**CK3WAX[6].Chan[3].Status.a**
$07B300	**Acc24E2*x*[18].Chan[0].Status.a**	**Acc24E3[7].Chan[0].Status.a**	**CK3WAX[7].Chan[0].Status.a**
$07B308	**Acc24E2*x*[18].Chan[1].Status.a**	**Acc24E3[7].Chan[1].Status.a**	**CK3WAX[7].Chan[1].Status.a**
$07B310	**Acc24E2*x*[18].Chan[2].Status.a**	**Acc24E3[7].Chan[2].Status.a**	**CK3WAX[7].Chan[2].Status.a**
$07B318	**Acc24E2*x*[18].Chan[3].Status.a**	**Acc24E3[7].Chan[3].Status.a**	**CK3WAX[7].Chan[3].Status.a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
4. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
5. **Status.a** addresses can also be entered as **PlusLimit.a**, but will report back as **Status.a**.

- Set **Motor[*x*].LimitBits** to 25 for Power UMAC ACC-24E2*x* boards.

- Set **Motor[*x*].LimitBits** to 9 for Power UMAC ACC-24E3 boards or CK3M CK3W-AX*nnnn* boards.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Limit Input Flags to Power PMAC

Turbo PMAC **I*xx*25**, **I*xx*43**	Power PMAC ACC-5E **Motor[*x*].pLimits**	Power PMAC ACC-5E3 **Motor[*x*].pLimits**
$003440	**Acc5E[0].Macro[0][3].a**	**Acc5E3[0].MacroInA[0][3].a**
$003441	**Acc5E[0].Macro[1][3].a**	**Acc5E3[0].MacroInA[1][3].a**
$003444	**Acc5E[0].Macro[4][3].a**	**Acc5E3[0].MacroInA[4][3].a**
$003445	**Acc5E[0].Macro[5][3].a**	**Acc5E3[0].MacroInA[5][3].a**
$003448	**Acc5E[0].Macro[8][3].a**	**Acc5E3[0].MacroInA[8][3].a**
$003449	**Acc5E[0].Macro[9][3].a**	**Acc5E3[0].MacroInA[9][3].a**
$00344C	**Acc5E[0].Macro[12][3].a**	**Acc5E3[0].MacroInA[12][3].a**
$00344D	**Acc5E[0].Macro[13][3].a**	**Acc5E3[0].MacroInA[13][3].a**
$003450	**Acc5E[1].Macro[0][3].a**	**Acc5E3[0].MacroInB[0][3].a**
$003451	**Acc5E[1].Macro[1][3].a**	**Acc5E3[0].MacroInB[1][3].a**
$003454	**Acc5E[1].Macro[4][3].a**	**Acc5E3[0].MacroInB[4][3].a**
$003455	**Acc5E[1].Macro[5][3].a**	**Acc5E3[0].MacroInB[5][3].a**
$003458	**Acc5E[1].Macro[8][3].a**	**Acc5E3[0].MacroInB[8][3].a**
$003459	**Acc5E[1].Macro[9][3].a**	**Acc5E3[0].MacroInB[9][3].a**
$00345C	**Acc5E[1].Macro[12][3].a**	**Acc5E3[0].MacroInB[12][3].a**
$00345D	**Acc5E[1].Macro[13][3].a**	**Acc5E3[0].MacroInB[13][3].a**
$003460	**Acc5E[2].Macro[0][3].a**	**Acc5E3[1].MacroInA[0][3].a**
$003461	**Acc5E[2].Macro[1][3].a**	**Acc5E3[1].MacroInA[1][3].a**
$003464	**Acc5E[2].Macro[4][3].a**	**Acc5E3[1].MacroInA[4][3].a**
$003465	**Acc5E[2].Macro[5][3].a**	**Acc5E3[1].MacroInA[5][3].a**
$003468	**Acc5E[2].Macro[8][3].a**	**Acc5E3[1].MacroInA[8][3].a**
$003469	**Acc5E[2].Macro[9][3].a**	**Acc5E3[1].MacroInA[9][3].a**
$00346C	**Acc5E[2].Macro[12][3].a**	**Acc5E3[1].MacroInA[12][3].a**
$00346D	**Acc5E[2].Macro[13][3].a**	**Acc5E3[1].MacroInA[13][3].a**
$003470	**Acc5E[3].Macro[0][3].a**	**Acc5E3[1].MacroInB[0][3].a**
$003471	**Acc5E[3].Macro[1][3].a**	**Acc5E3[1].MacroInB[1][3].a**
$003474	**Acc5E[3].Macro[4][3].a**	**Acc5E3[1].MacroInB[4][3].a**
$003475	**Acc5E[3].Macro[5][3].a**	**Acc5E3[1].MacroInB[5][3].a**
$003478	**Acc5E[3].Macro[8][3].a**	**Acc5E3[1].MacroInB[8][3].a**
$003479	**Acc5E[3].Macro[9][3].a**	**Acc5E3[1].MacroInB[9][3].a**
$00347C	**Acc5E[3].Macro[12][3].a**	**Acc5E3[1].MacroInB[12][3].a**
$00347D	**Acc5E[3].Macro[13][3].a**	**Acc5E3[1].MacroInB[13][3].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

- Set **Motor[*x*].LimitBits** to 25 for Power PMAC MACRO limit input flags.

Motor Scaling Variables

In Turbo PMAC, the motor position units must be "counts" or "LSBs" of the feedback device, with matching axis position units permitted to be scaled to user units such as millimeters, inches, or degrees.

Power PMAC permits the motor position units to be scaled to user units as well. However, for the fastest conversion from Turbo PMAC, it is easiest to keep the Power PMAC position units in "counts". Otherwise, several dozen motor setup elements must be rescaled to the new user units.

- Set position-loop scaling factor **Motor[*x*].PosSf** to 1.0 to specify Power PMAC motor units as feedback counts or LSBs. (This assumes the Encoder Conversion Table entry output used for position loop feedback is also scaled in units of feedback counts or LSBs.) *Many of the other settings recommended in this note depend on this scaling.*

- Set velocity-loop scaling factor **Motor[*x*].Pos2Sf** equal to (**I*xx*09** / **I*xx*08**) to keep the same relative scaling between position-loop feedback and velocity-loop feedback. In the vast majority of cases, especially those with the same feedback source for both loops, **Motor[*x*].Pos2Sf** will be set to 1.0. *Many of the other settings recommended in this note depend on this setting.*

- Set **Motor[*x*].MasterPosSf** equal to (**I*xx*07** / **I*xx*08**) to keep the same master position following ratio.

Motor Power-On Mode

I*xx*80 to **Motor[*x*].PowerOnMode**:

- If **I*xx*80** = 0 or 2, set **Motor[*x*].PowerOnMode** to 0.

- If **I*xx*80** = 1 or 3, set **Motor[*x*].PowerOnMode** to 3.

- If **I*xx*80** = 4 or 6, set **Motor[*x*].PowerOnMode** to 4.

- If **I*xx*80** = 5 or 7, set **Motor[*x*].PowerOnMode** to 7.

Motor Commutation and Current Loop I-Variable Equivalents

The elements in this section only need to be set in Power PMAC if **I*xx*01** in Turbo PMAC is equal to 1, specifying that PMAC is performing phase commutation for the motor.

If **I*xx*82** in Turbo PMAC is set to a value greater than 0, PMAC is closing the current loop as well, so the Power PMAC elements specifying the current loop closure will need to be set as well. These are covered in a sub-section below.

If **I*xx*01** is set to 1 and **I*xx*82** is set to 0 in Turbo PMAC, then PMAC is performing the commutation but not the current loop closure for the motor. This mode is often called "sine wave output" mode. In this case, **Motor[*x*].PwmSf** should be set to 32,767 for the same output scaling as on Turbo PMAC.

(Note: If using the "direct microstepping" PWM control of stepper motors, as with the Brick LV or Clipper Stack amplifier, the technique is substantially different in Power PMAC. Refer to the User's Manual chapter *Setting Up Commutation and/or Current Loop* for step-by-step instructions.)

Phase Cycle Parameters

- Set **Motor[*x*].PhasePosSf** equal to [2048 / (256 * **I*xx*71** / **I*xx*70**)] for equivalent commutation cycle size. (If **I*xx*70** = 0 to disable AC commutation for DC brush motor control, set **Motor[*x*].PhasePosSf** to 0 regardless of the **I*xx*71** setting.)

- Set **Motor[*x*].PhaseOffset** equal to **I*xx*72** for equivalent angle offset between phases.

Phasing Search Move Parameters

These parameters are only used for synchronous motor commutation with no absolute phase position sensor.

- Set **Motor[*x*].PhaseFindingDac** equal to **I*xx*73** for equivalent command magnitude in phasing search moves.

- **I*xx*74** to **Motor[*x*].PhaseFindingTime**:

    - If **I*xx*74** is set to 0, set **Motor[*x*].PhaseFindingTime** to 0 to disable any phasing search moves.

    - If **I*xx*74** is greater than 0 and **I*xx*80** bit 1 (value 2) is set to 0, set **Motor[*x*].PhaseFindingTime** equal to **I*xx*74** for the same move time in the "guess" phasing search move.

    - If **I*xx*74** is greater than 0 and **I*xx*80** bit 1 (value 2) is set to 1, set **Motor[*x*].PhaseFindingTime** equal to (256 * **I*xx*74**) for the same move time in the "guess" phasing search move.

Induction Motor Parameters

- Set **Motor[*x*].IdCmd** equal to **I*xx*77** for equivalent commanded direct current.

- Set **Motor[*x*].DtOverRotorTc** equal to **I*xx*78** for equivalent slip scaling.

Phase Current Offset Parameters

- Set **Motor[*x*].IaBias** equal to **I*xx*29** for the same Phase A current offset.
- Set **Motor[*x*].IbBias** equal to **I*xx*79** for the same Phase B current offset.

Other Parameters

- If **I*xx*55** = 0, set **Motor[*x*].pSineTable** and **Motor[*x*].pVoltSineTable** to **Sys.SineTable.a** (default) to use standard "pure sine" commutation lookup tables.
- If **I*xx*55** is not equal to 0, a matching custom sine table must be built and loaded into the Power PMAC, with **Motor[*x*].pSineTable** and **Motor[*x*].pVoltSineTable** set to the address of the start of the table.
- Set **Motor[*x*].AdvGain** equal to (**I*xx*56** * **I*xx*09** *32) for equivalent velocity phase advance.

I*xx*83 to Motor[*x*].pPhaseEnc

This element specifies the register used for ongoing commutation feedback position. The following tables show the most common (and typically default) settings for this parameter, using an incremental encoder for feedback, whether digital quadrature or analog sinusoidal.

If the Turbo PMAC application uses a serial encoder through an ACC-84*x* interface board, the Power PMAC settings of **Motor[*x*].pPhaseEnc** for a given value of I*xx*83 are the same as the Power PMAC settings of **Motor[*x*].pAbsPhasePos** for a given value of I*xx*81 absolute commutation feedback, shown in tables in the next section.

Turbo Clipper to Power Clipper

Turbo Clipper Commutation Feedback to Power Clipper

Turbo Clipper **I*xx*83**	Power Clipper **Motor[*x*].pPhaseEnc**
$078001	**Clipper[0].Chan[0].PhaseCapt.a**
$078009	**Clipper[0].Chan[1].PhaseCapt.a**
$078011	**Clipper[0].Chan[2].PhaseCapt.a**
$078019	**Clipper[0].Chan[3].PhaseCapt.a**
$078101	**Clipper[1].Chan[0].PhaseCapt.a**
$078109	**Clipper[1].Chan[1].PhaseCapt.a**
$078111	**Clipper[1].Chan[2].PhaseCapt.a**
$078119	**Clipper[1].Chan[3].PhaseCapt.a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

Turbo Brick to Power Brick

Turbo Brick Commutation Feedback to Power Brick

Turbo Brick **I*xx*83**	Power Brick **Motor[*x*].pPhaseEnc**
$078001	**PowerBrick[0].Chan[0].PhaseCapt.a**
$078009	**PowerBrick[0].Chan[1].PhaseCapt.a**
$078011	**PowerBrick[0].Chan[2].PhaseCapt.a**
$078019	**PowerBrick[0].Chan[3].PhaseCapt.a**
$078101	**PowerBrick[1].Chan[0].PhaseCapt.a**
$078109	**PowerBrick[1].Chan[1].PhaseCapt.a**
$078111	**PowerBrick[1].Chan[2].PhaseCapt.a**
$078119	**PowerBrick[1].Chan[3].PhaseCapt.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

Turbo UMAC to Power UMAC

Turbo UMAC Commutation Feedback to Power UMAC

Turbo UMAC **I*xx*83**	Power UMAC ACC-24E2*x* **Motor[*x*].pPhaseEnc**	Power UMAC ACC-24E3 **Motor[*x*].pPhaseEnc**
$078201	**Acc24E2*x*[4].Chan[0].PhaseCapt.a**	**Acc24E3[0].Chan[0].PhaseCapt.a**
$078209	**Acc24E2*x*[4].Chan[1].PhaseCapt.a**	**Acc24E3[0].Chan[1].PhaseCapt.a**
$078211	**Acc24E2*x*[4].Chan[2].PhaseCapt.a**	**Acc24E3[0].Chan[2].PhaseCapt.a**
$078219	**Acc24E2*x*[4].Chan[3].PhaseCapt.a**	**Acc24E3[0].Chan[3].PhaseCapt.a**
$078301	**Acc24E2*x*[6].Chan[0].PhaseCapt.a**	**Acc24E3[1].Chan[0].PhaseCapt.a**
$078309	**Acc24E2*x*[6].Chan[1].PhaseCapt.a**	**Acc24E3[1].Chan[1].PhaseCapt.a**
$078311	**Acc24E2*x*[6].Chan[2].PhaseCapt.a**	**Acc24E3[1].Chan[2].PhaseCapt.a**
$078319	**Acc24E2*x*[6].Chan[3].PhaseCapt.a**	**Acc24E3[1].Chan[3].PhaseCapt.a**
$079201	**Acc24E2*x*[8].Chan[0].PhaseCapt.a**	**Acc24E3[2].Chan[0].PhaseCapt.a**
$079209	**Acc24E2*x*[8].Chan[1].PhaseCapt.a**	**Acc24E3[2].Chan[1].PhaseCapt.a**
$079211	**Acc24E2*x*[8].Chan[2].PhaseCapt.a**	**Acc24E3[2].Chan[2].PhaseCapt.a**
$079219	**Acc24E2*x*[8].Chan[3].PhaseCapt.a**	**Acc24E3[2].Chan[3].PhaseCapt.a**
$079301	**Acc24E2*x*[10].Chan[0].PhaseCapt.a**	**Acc24E3[3].Chan[0].PhaseCapt.a**
$079309	**Acc24E2*x*[10].Chan[1].PhaseCapt.a**	**Acc24E3[3].Chan[1].PhaseCapt.a**
$079311	**Acc24E2*x*[10].Chan[2].PhaseCapt.a**	**Acc24E3[3].Chan[2].PhaseCapt.a**
$079319	**Acc24E2*x*[10].Chan[3].PhaseCapt.a**	**Acc24E3[3].Chan[3].PhaseCapt.a**
$07A201	**Acc24E2*x*[12].Chan[0].PhaseCapt.a**	**Acc24E3[4].Chan[0].PhaseCapt.a**
$07A209	**Acc24E2*x*[12].Chan[1].PhaseCapt.a**	**Acc24E3[4].Chan[1].PhaseCapt.a**
$07A211	**Acc24E2*x*[12].Chan[2].PhaseCapt.a**	**Acc24E3[4].Chan[2].PhaseCapt.a**
$07A219	**Acc24E2*x*[12].Chan[3].PhaseCapt.a**	**Acc24E3[4].Chan[3].PhaseCapt.a**
$07A301	**Acc24E2*x*[14].Chan[0].PhaseCapt.a**	**Acc24E3[5].Chan[0].PhaseCapt.a**
$07A309	**Acc24E2*x*[14].Chan[1].PhaseCapt.a**	**Acc24E3[5].Chan[1].PhaseCapt.a**
$07A311	**Acc24E2*x*[14].Chan[2].PhaseCapt.a**	**Acc24E3[5].Chan[2].PhaseCapt.a**
$07A319	**Acc24E2*x*[14].Chan[3].PhaseCapt.a**	**Acc24E3[5].Chan[3].PhaseCapt.a**
$07B201	**Acc24E2*x*[16].Chan[0].PhaseCapt.a**	**Acc24E3[6].Chan[0].PhaseCapt.a**
$07B209	**Acc24E2*x*[16].Chan[1].PhaseCapt.a**	**Acc24E3[6].Chan[1].PhaseCapt.a**
$07B211	**Acc24E2*x*[16].Chan[2].PhaseCapt.a**	**Acc24E3[6].Chan[2].PhaseCapt.a**
$07B219	**Acc24E2*x*[16].Chan[3].PhaseCapt.a**	**Acc24E3[6].Chan[3].PhaseCapt.a**
$07B301	**Acc24E2*x*[18].Chan[0].PhaseCapt.a**	**Acc24E3[7].Chan[0].PhaseCapt.a**
$07B309	**Acc24E2*x*[18].Chan[1].PhaseCapt.a**	**Acc24E3[7].Chan[1].PhaseCapt.a**
$07B311	**Acc24E2*x*[18].Chan[2].PhaseCapt.a**	**Acc24E3[7].Chan[2].PhaseCapt.a**
$07B319	**Acc24E2*x*[18].Chan[3].PhaseCapt.a**	**Acc24E3[7].Chan[3].PhaseCapt.a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.

Turbo UMAC to CK3M

Turbo UMAC Commutation Feedback to CK3M

Turbo UMAC **I*xx*83**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pPhaseEnc**
$078201	**CK3WAX[0].Chan[0].PhaseCapt.a**
$078209	**CK3WAX[0].Chan[1].PhaseCapt.a**
$078211	**CK3WAX[0].Chan[2].PhaseCapt.a**
$078219	**CK3WAX[0].Chan[3].PhaseCapt.a**
$078301	**CK3WAX[1].Chan[0].PhaseCapt.a**
$078309	**CK3WAX[1].Chan[1].PhaseCapt.a**
$078311	**CK3WAX[1].Chan[2].PhaseCapt.a**
$078319	**CK3WAX[1].Chan[3].PhaseCapt.a**
$079201	**CK3WAX[2].Chan[0].PhaseCapt.a**
$079209	**CK3WAX[2].Chan[1].PhaseCapt.a**
$079211	**CK3WAX[2].Chan[2].PhaseCapt.a**
$079219	**CK3WAX[2].Chan[3].PhaseCapt.a**
$079301	**CK3WAX[3].Chan[0].PhaseCapt.a**
$079309	**CK3WAX[3].Chan[1].PhaseCapt.a**
$079311	**CK3WAX[3].Chan[2].PhaseCapt.a**
$079319	**CK3WAX[3].Chan[3].PhaseCapt.a**
$07A201	**CK3WAX[4].Chan[0].PhaseCapt.a**
$07A209	**CK3WAX[4].Chan[1].PhaseCapt.a**
$07A211	**CK3WAX[4].Chan[2].PhaseCapt.a**
$07A219	**CK3WAX[4].Chan[3].PhaseCapt.a**
$07A301	**CK3WAX[5].Chan[0].PhaseCapt.a**
$07A309	**CK3WAX[5].Chan[1].PhaseCapt.a**
$07A311	**CK3WAX[5].Chan[2].PhaseCapt.a**
$07A319	**CK3WAX[5].Chan[3].PhaseCapt.a**
$07B201	**CK3WAX[6].Chan[0].PhaseCapt.a**
$07B209	**CK3WAX[6].Chan[1].PhaseCapt.a**
$07B211	**CK3WAX[6].Chan[2].PhaseCapt.a**
$07B219	**CK3WAX[6].Chan[3].PhaseCapt.a**
$07B301	**CK3WAX[7].Chan[0].PhaseCapt.a**
$07B309	**CK3WAX[7].Chan[1].PhaseCapt.a**
$07B311	**CK3WAX[7].Chan[2].PhaseCapt.a**
$07B319	**CK3WAX[7].Chan[3].PhaseCapt.a**

Notes:

1. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Commutation Feedback to Power PMAC

Turbo PMAC **I*xx*83**	Power UMAC **Motor[*x*].pPhaseEnc** for ACC-5E Node 0 Feedback	Power PMAC **Motor[*x*].pPhaseEnc** for ACC-5E3 Node 0 Feedback
$078420	**Acc5E[0].Macro[0][0].a**	**Acc5E3[0].MacroInA[0][0].a**
$078424	**Acc5E[0].Macro[1][0].a**	**Acc5E3[0].MacroInA[1][0].a**
$078428	**Acc5E[0].Macro[4][0].a**	**Acc5E3[0].MacroInA[4][0].a**
$07842C	**Acc5E[0].Macro[5][0].a**	**Acc5E3[0].MacroInA[5][0].a**
$078430	**Acc5E[0].Macro[8][0].a**	**Acc5E3[0].MacroInA[8][0].a**
$078434	**Acc5E[0].Macro[9][0].a**	**Acc5E3[0].MacroInA[9][0].a**
$078438	**Acc5E[0].Macro[12][0].a**	**Acc5E3[0].MacroInA[12][0].a**
$07843C	**Acc5E[0].Macro[13][0].a**	**Acc5E3[0].MacroInA[13][0].a**
$079420	**Acc5E[1].Macro[0][0].a**	**Acc5E3[0].MacroInB[0][0].a**
$079424	**Acc5E[1].Macro[1][0].a**	**Acc5E3[0].MacroInB[1][0].a**
$079428	**Acc5E[1].Macro[4][0].a**	**Acc5E3[0].MacroInB[4][0].a**
$07942C	**Acc5E[1].Macro[5][0].a**	**Acc5E3[0].MacroInB[5][0].a**
$079430	**Acc5E[1].Macro[8][0].a**	**Acc5E3[0].MacroInB[8][0].a**
$079434	**Acc5E[1].Macro[9][0].a**	**Acc5E3[0].MacroInB[9][0].a**
$079438	**Acc5E[1].Macro[12][0].a**	**Acc5E3[0].MacroInB[12][0].a**
$07943C	**Acc5E[1].Macro[13][0].a**	**Acc5E3[0].MacroInB[13][0].a**
$07A420	**Acc5E[2].Macro[0][0].a**	**Acc5E3[1].MacroInA[0][0].a**
$07A424	**Acc5E[2].Macro[1][0].a**	**Acc5E3[1].MacroInA[1][0].a**
$07A428	**Acc5E[2].Macro[4][0].a**	**Acc5E3[1].MacroInA[4][0].a**
$07A42C	**Acc5E[2].Macro[5][0].a**	**Acc5E3[1].MacroInA[5][0].a**
$07A430	**Acc5E[2].Macro[8][0].a**	**Acc5E3[1].MacroInA[8][0].a**
$07A434	**Acc5E[2].Macro[9][0].a**	**Acc5E3[1].MacroInA[9][0].a**
$07A438	**Acc5E[2].Macro[12][0].a**	**Acc5E3[1].MacroInA[12][0].a**
$07A43C	**Acc5E[2].Macro[13][0].a**	**Acc5E3[1].MacroInA[13][0].a**
$07B420	**Acc5E[3].Macro[0][0].a**	**Acc5E3[1].MacroInB[0][0].a**
$07B424	**Acc5E[3].Macro[1][0].a**	**Acc5E3[1].MacroInB[1][0].a**
$07B428	**Acc5E[3].Macro[4][0].a**	**Acc5E3[1].MacroInB[4][0].a**
$07B42C	**Acc5E[3].Macro[5][0].a**	**Acc5E3[1].MacroInB[5][0].a**
$07B430	**Acc5E[3].Macro[8][0].a**	**Acc5E3[1].MacroInB[8][0].a**
$07B434	**Acc5E[3].Macro[9][0].a**	**Acc5E3[1].MacroInB[9][0].a**
$07B438	**Acc5E[3].Macro[12][0].a**	**Acc5E3[1].MacroInB[12][0].a**
$07B43C	**Acc5E[3].Macro[13][0].a**	**Acc5E3[1].MacroInB[13][0].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

I*xx*81 to Motor[*x*].pAbsPhasePos

- If **I*xx*81** is set to 0, set **Motor[*x*].pAbsPhasePos** to 0 to disable absolute power-on phase position read.

- If **I*xx*81** is not set to 0, set **Motor[*x*].pAbsPhasePos** to the matching Power PMAC address. The following tables show the settings for serial encoders and Hall sensors, the most common sources of absolute phase position.

  If setting up for direct-PWM control of a DC brush motor, set **Motor[*x*].pAbsPhasePos** to the address of any register that will return a true numerical value when read. (The value will be multiplied by a scale factor of 0.0.) It is fine to set it to **Sys.pushm** for any of these motors.

ACC-84x Turbo PMAC Serial Encoder Interface

A common absolute phase position feedback for Turbo PMAC systems uses an ACC-84S, ACC-84B, or ACC-84E interface board. These ACC-84*x* boards can also be used on the comparable Power PMAC systems, but for many serial encoder protocols, the built-in standard axis interface can process the serial encoder signal without need for an accessory board.

Turbo Clipper with ACC-84S to Power Clipper

Turbo Clipper **I*xx*81**	Power Clipper with ACC-84S **Motor[*x*].pAbsPhasePos**	Power Clipper Standard Interface **Motor[*x*].pAbsPhasePos**
$078800	**Acc84S[0].Chan[0].SerialEncDataA.a**	**Clipper[0].Chan[0].SerialEncDataA.a**
$078804	**Acc84S[0].Chan[1].SerialEncDataA.a**	**Clipper[0].Chan[1].SerialEncDataA.a**
$078808	**Acc84S[0].Chan[2].SerialEncDataA.a**	**Clipper[0].Chan[2].SerialEncDataA.a**
$07880C	**Acc84S[0].Chan[3].SerialEncDataA.a**	**Clipper[0].Chan[3].SerialEncDataA.a**
$078820	**Acc84S[1].Chan[0].SerialEncDataA.a**	**Clipper[1].Chan[0].SerialEncDataA.a**
$078824	**Acc84S[1].Chan[1].SerialEncDataA.a**	**Clipper[1].Chan[1].SerialEncDataA.a**
$078828	**Acc84S[1].Chan[2].SerialEncDataA.a**	**Clipper[1].Chan[2].SerialEncDataA.a**
$07882C	**Acc84S[1].Chan[3].SerialEncDataA.a**	**Clipper[1].Chan[3].SerialEncDataA.a**

Notes:
1. ACC-84S is not auto-identified by Power PMAC, so must be manually identified by setting **GateIo[*i*].PartNum** to 603936, **GateIo[*i*].PartType** to 8, **`save`**, and reset.
2. Structure for standard interface can be entered as **Gate3[*i*]** or **Clipper[*i*]**, but will report back as **Clipper[*i*]**.

Turbo Brick with ACC-84B to Power Brick

Turbo Brick **I*xx*81**	Power Brick with ACC-84B **Motor[*x*].pAbsPhasePos**	Power Brick Standard Interface **Motor[*x*].pAbsPhasePos**
$078B20	**Acc84B[0].Chan[0].SerialEncDataA.a**	**Gate3[0].Chan[0].SerialEncDataA.a**
$078B24	**Acc84B[0].Chan[1].SerialEncDataA.a**	**Gate3[0].Chan[1].SerialEncDataA.a**
$078B28	**Acc84B[0].Chan[2].SerialEncDataA.a**	**Gate3[0].Chan[2].SerialEncDataA.a**
$078B2C	**Acc84B[0].Chan[3].SerialEncDataA.a**	**Gate3[0].Chan[3].SerialEncDataA.a**
$078B30	**Acc84B[1].Chan[0].SerialEncDataA.a**	**Gate3[1].Chan[0].SerialEncDataA.a**
$078B34	**Acc84B[1].Chan[1].SerialEncDataA.a**	**Gate3[1].Chan[1].SerialEncDataA.a**
$078B38	**Acc84B[1].Chan[2].SerialEncDataA.a**	**Gate3[1].Chan[2].SerialEncDataA.a**
$078B3C	**Acc84B[1].Chan[3].SerialEncDataA.a**	**Gate3[1].Chan[3].SerialEncDataA.a**

Notes:
1. Structure for standard interface can be entered as **Gate3[*i*]** or **PowerBrick[*i*]**, but will report back as **PowerBrick[*i*]**.

Turbo UMAC with ACC-84E to Power UMAC

Turbo UMAC **I*xx*81**	Power UMAC with ACC-84E **Motor[*x*].pAbsPhasePos**	Power UMAC Standard Interface **Motor[*x*].pAbsPhasePos**
$078C00	**Acc84E[0].Chan[0].SerialEncDataA.a**	**Acc24E3[0].Chan[0].SerialEncDataA.a**
$078C04	**Acc84E[0].Chan[1].SerialEncDataA.a**	**Acc24E3[0].Chan[1].SerialEncDataA.a**
$078C08	**Acc84E[0].Chan[2].SerialEncDataA.a**	**Acc24E3[0].Chan[2].SerialEncDataA.a**
$078C0C	**Acc84E[0].Chan[3].SerialEncDataA.a**	**Acc24E3[0].Chan[3].SerialEncDataA.a**
$078D00	**Acc84E[1].Chan[0].SerialEncDataA.a**	**Acc24E3[1].Chan[0].SerialEncDataA.a**
$078D04	**Acc84E[1].Chan[1].SerialEncDataA.a**	**Acc24E3[1].Chan[1].SerialEncDataA.a**
$078D08	**Acc84E[1].Chan[2].SerialEncDataA.a**	**Acc24E3[1].Chan[2].SerialEncDataA.a**
$078D0C	**Acc84E[1].Chan[3].SerialEncDataA.a**	**Acc24E3[1].Chan[3].SerialEncDataA.a**
$078E00	**Acc84E[2].Chan[0].SerialEncDataA.a**	**Acc24E3[2].Chan[0].SerialEncDataA.a**
$078E04	**Acc84E[2].Chan[1].SerialEncDataA.a**	**Acc24E3[2].Chan[1].SerialEncDataA.a**
$078E08	**Acc84E[2].Chan[2].SerialEncDataA.a**	**Acc24E3[2].Chan[2].SerialEncDataA.a**
$078E0C	**Acc84E[2].Chan[3].SerialEncDataA.a**	**Acc24E3[2].Chan[3].SerialEncDataA.a**
$079C00	**Acc84E[4].Chan[0].SerialEncDataA.a**	**Acc24E3[3].Chan[0].SerialEncDataA.a**
$079C04	**Acc84E[4].Chan[1].SerialEncDataA.a**	**Acc24E3[3].Chan[1].SerialEncDataA.a**
$079C08	**Acc84E[4].Chan[2].SerialEncDataA.a**	**Acc24E3[3].Chan[2].SerialEncDataA.a**
$079C0C	**Acc84E[4].Chan[3].SerialEncDataA.a**	**Acc24E3[3].Chan[3].SerialEncDataA.a**
$079D00	**Acc84E[5].Chan[0].SerialEncDataA.a**	**Acc24E3[4].Chan[0].SerialEncDataA.a**
$079D04	**Acc84E[5].Chan[1].SerialEncDataA.a**	**Acc24E3[4].Chan[1].SerialEncDataA.a**
$079D08	**Acc84E[5].Chan[2].SerialEncDataA.a**	**Acc24E3[4].Chan[2].SerialEncDataA.a**
$079D0C	**Acc84E[5].Chan[3].SerialEncDataA.a**	**Acc24E3[4].Chan[3].SerialEncDataA.a**
$079E00	**Acc84E[6].Chan[0].SerialEncDataA.a**	**Acc24E3[5].Chan[0].SerialEncDataA.a**
$079E04	**Acc84E[6].Chan[1].SerialEncDataA.a**	**Acc24E3[5].Chan[1].SerialEncDataA.a**
$079E08	**Acc84E[6].Chan[2].SerialEncDataA.a**	**Acc24E3[5].Chan[2].SerialEncDataA.a**
$079E0C	**Acc84E[6].Chan[3].SerialEncDataA.a**	**Acc24E3[5].Chan[3].SerialEncDataA.a**
$07AC00	**Acc84E[8].Chan[0].SerialEncDataA.a**	**Acc24E3[6].Chan[0].SerialEncDataA.a**
$07AC04	**Acc84E[8].Chan[1].SerialEncDataA.a**	**Acc24E3[6].Chan[1].SerialEncDataA.a**
$07AC08	**Acc84E[8].Chan[2].SerialEncDataA.a**	**Acc24E3[6].Chan[2].SerialEncDataA.a**
$07AC0C	**Acc84E[8].Chan[3].SerialEncDataA.a**	**Acc24E3[6].Chan[3].SerialEncDataA.a**
$07AD00	**Acc84E[9].Chan[0].SerialEncDataA.a**	**Acc24E3[7].Chan[0].SerialEncDataA.a**
$07AD04	**Acc84E[9].Chan[1].SerialEncDataA.a**	**Acc24E3[7].Chan[1].SerialEncDataA.a**
$07AD08	**Acc84E[9].Chan[2].SerialEncDataA.a**	**Acc24E3[7].Chan[2].SerialEncDataA.a**
$07AD0C	**Acc84E[9].Chan[3].SerialEncDataA.a**	**Acc24E3[7].Chan[3].SerialEncDataA.a**
$07AE00	**Acc84E[10].Chan[0].SerialEncDataA.a**	**Acc24E3[8].Chan[0].SerialEncDataA.a**
$07AE04	**Acc84E[10].Chan[1].SerialEncDataA.a**	**Acc24E3[8].Chan[1].SerialEncDataA.a**
$07AE08	**Acc84E[10].Chan[2].SerialEncDataA.a**	**Acc24E3[8].Chan[2].SerialEncDataA.a**
$07AE0C	**Acc84E[10].Chan[3].SerialEncDataA.a**	**Acc24E3[8].Chan[3].SerialEncDataA.a**

Notes:

1. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.

Turbo UMAC with ACC-84E to CK3M Serial Encoders

Turbo UMAC **I*xx*81**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pAbsPhasePos**
$078C00	**CK3WAX[0].Chan[0].SerialEncDataA.a**
$078C04	**CK3WAX[0].Chan[1].SerialEncDataA.a**
$078C08	**CK3WAX[0].Chan[2].SerialEncDataA.a**
$078C0C	**CK3WAX[0].Chan[3].SerialEncDataA.a**
$078D00	**CK3WAX[1].Chan[0].SerialEncDataA.a**
$078D04	**CK3WAX[1].Chan[1].SerialEncDataA.a**
$078D08	**CK3WAX[1].Chan[2].SerialEncDataA.a**
$078D0C	**CK3WAX[1].Chan[3].SerialEncDataA.a**
$078E00	**CK3WAX[2].Chan[0].SerialEncDataA.a**
$078E04	**CK3WAX[2].Chan[1].SerialEncDataA.a**
$078E08	**CK3WAX[2].Chan[2].SerialEncDataA.a**
$078E0C	**CK3WAX[2].Chan[3].SerialEncDataA.a**
$079C00	**CK3WAX[3].Chan[0].SerialEncDataA.a**
$079C04	**CK3WAX[3].Chan[1].SerialEncDataA.a**
$079C08	**CK3WAX[3].Chan[2].SerialEncDataA.a**
$079C0C	**CK3WAX[3].Chan[3].SerialEncDataA.a**
$079D00	**CK3WAX[4].Chan[0].SerialEncDataA.a**
$079D04	**CK3WAX[4].Chan[1].SerialEncDataA.a**
$079D08	**CK3WAX[4].Chan[2].SerialEncDataA.a**
$079D0C	**CK3WAX[4].Chan[3].SerialEncDataA.a**
$079E00	**CK3WAX[5].Chan[0].SerialEncDataA.a**
$079E04	**CK3WAX[5].Chan[1].SerialEncDataA.a**
$079E08	**CK3WAX[5].Chan[2].SerialEncDataA.a**
$079E0C	**CK3WAX[5].Chan[3].SerialEncDataA.a**
$07AC00	**CK3WAX[6].Chan[0].SerialEncDataA.a**
$07AC04	**CK3WAX[6].Chan[1].SerialEncDataA.a**
$07AC08	**CK3WAX[6].Chan[2].SerialEncDataA.a**
$07AC0C	**CK3WAX[6].Chan[3].SerialEncDataA.a**
$07AD00	**CK3WAX[7].Chan[0].SerialEncDataA.a**
$07AD04	**CK3WAX[7].Chan[1].SerialEncDataA.a**
$07AD08	**CK3WAX[7].Chan[2].SerialEncDataA.a**
$07AD0C	**CK3WAX[7].Chan[3].SerialEncDataA.a**
$07AE00	**CK3WAX[8].Chan[0].SerialEncDataA.a**
$07AE04	**CK3WAX[8].Chan[1].SerialEncDataA.a**
$07AE08	**CK3WAX[8].Chan[2].SerialEncDataA.a**
$07AE0C	**CK3WAX[8].Chan[3].SerialEncDataA.a**

Notes:

1. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

Hall Commutation Sensors (UVW)

When incremental encoder feedback is used for ongoing commutation position feedback, it is common to use Hall sensors or equivalent for the initial rough phasings. The three digital inputs from these Hall sensors are typically brought into the U, V, and W flag inputs for a servo channel.

Note that this only provides a rough phasing (+/-30°e). A "fine phasing" correction is required for reliable and efficient operation; this is accomplished with **Motor[*x*].AbsPhasePosForce**, covered below.

Turbo Clipper Hall Inputs to Power Clipper Hall Inputs

Turbo Clipper **I*xx*81**	Power Clipper **Motor[*x*].pAbsPhasePos**
$078000	**Clipper[0].Chan[0].Status.a**
$078008	**Clipper[0].Chan[1].Status.a**
$078010	**Clipper[0].Chan[2].Status.a**
$078018	**Clipper[0].Chan[3].Status.a**
$078100	**Clipper[1].Chan[0].Status.a**
$078108	**Clipper[1].Chan[1].Status.a**
$078110	**Clipper[1].Chan[2].Status.a**
$078118	**Clipper[1].Chan[3].Status.a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.
2. Clipper **Status.a** addresses can also be entered as **UVW.a**, but will report back as **Status.a**.

Turbo Brick Hall Inputs to Power Brick Hall Inputs

Turbo Brick **I*xx*81**	Power Brick **Motor[*x*].pAbsPhasePos**
$078000	**PowerBrick[0].Chan[0].Status.a**
$078008	**PowerBrick[0].Chan[1].Status.a**
$078010	**PowerBrick[0].Chan[2].Status.a**
$078018	**PowerBrick[0].Chan[3].Status.a**
$078100	**PowerBrick[1].Chan[0].Status.a**
$078108	**PowerBrick[1].Chan[1].Status.a**
$078110	**PowerBrick[1].Chan[2].Status.a**
$078118	**PowerBrick[1].Chan[3].Status.a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.
2. Brick **Status.a** addresses can also be entered as **UVW.a**, but will report back as **Status.a**.

Turbo UMAC Hall Inputs to Power UMAC or CK3M Hall Inputs

Turbo UMAC **I*xx*81**	Power UMAC ACC-24E2*x* **Motor[*x*].pAbsPhasePos**	Power UMAC ACC-24E3 **Motor[*x*].pAbsPhasePos**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pAbsPhasePos**
$078200	**Acc24E2*x*[4].Chan[0].Status.a**	**Acc24E3[0].Chan[0].Status.a**	**CK3WAX[0].Chan[0].Status.a**
$078208	**Acc24E2*x*[4].Chan[1].Status.a**	**Acc24E3[0].Chan[1].Status.a**	**CK3WAX[0].Chan[1].Status.a**
$078210	**Acc24E2*x*[4].Chan[2].Status.a**	**Acc24E3[0].Chan[2].Status.a**	**CK3WAX[0].Chan[2].Status.a**
$078218	**Acc24E2*x*[4].Chan[3].Status.a**	**Acc24E3[0].Chan[3].Status.a**	**CK3WAX[0].Chan[3].Status.a**
$078300	**Acc24E2*x*[6].Chan[0].Status.a**	**Acc24E3[1].Chan[0].Status.a**	**CK3WAX[1].Chan[0].Status.a**
$078308	**Acc24E2*x*[6].Chan[1].Status.a**	**Acc24E3[1].Chan[1].Status.a**	**CK3WAX[1].Chan[1].Status.a**
$078310	**Acc24E2*x*[6].Chan[2].Status.a**	**Acc24E3[1].Chan[2].Status.a**	**CK3WAX[1].Chan[2].Status.a**
$078318	**Acc24E2*x*[6].Chan[3].Status.a**	**Acc24E3[1].Chan[3].Status.a**	**CK3WAX[1].Chan[3].Status.a**
$079200	**Acc24E2*x*[8].Chan[0].Status.a**	**Acc24E3[2].Chan[0].Status.a**	**CK3WAX[2].Chan[0].Status.a**
$079208	**Acc24E2*x*[8].Chan[1].Status.a**	**Acc24E3[2].Chan[1].Status.a**	**CK3WAX[2].Chan[1].Status.a**
$079210	**Acc24E2*x*[8].Chan[2].Status.a**	**Acc24E3[2].Chan[2].Status.a**	**CK3WAX[2].Chan[2].Status.a**
$079218	**Acc24E2*x*[8].Chan[3].Status.a**	**Acc24E3[2].Chan[3].Status.a**	**CK3WAX[2].Chan[3].Status.a**
$079300	**Acc24E2*x*[10].Chan[0].Status.a**	**Acc24E3[3].Chan[0].Status.a**	**CK3WAX[3].Chan[0].Status.a**
$079308	**Acc24E2*x*[10].Chan[1].Status.a**	**Acc24E3[3].Chan[1].Status.a**	**CK3WAX[3].Chan[1].Status.a**
$079310	**Acc24E2*x*[10].Chan[2].Status.a**	**Acc24E3[3].Chan[2].Status.a**	**CK3WAX[3].Chan[2].Status.a**
$079318	**Acc24E2*x*[10].Chan[3].Status.a**	**Acc24E3[3].Chan[3].Status.a**	**CK3WAX[3].Chan[3].Status.a**
$07A200	**Acc24E2*x*[12].Chan[0].Status.a**	**Acc24E3[4].Chan[0].Status.a**	**CK3WAX[4].Chan[0].Status.a**
$07A208	**Acc24E2*x*[12].Chan[1].Status.a**	**Acc24E3[4].Chan[1].Status.a**	**CK3WAX[4].Chan[1].Status.a**
$07A210	**Acc24E2*x*[12].Chan[2].Status.a**	**Acc24E3[4].Chan[2].Status.a**	**CK3WAX[4].Chan[2].Status.a**
$07A218	**Acc24E2*x*[12].Chan[3].Status.a**	**Acc24E3[4].Chan[3].Status.a**	**CK3WAX[4].Chan[3].Status.a**
$07A300	**Acc24E2*x*[14].Chan[0].Status.a**	**Acc24E3[5].Chan[0].Status.a**	**CK3WAX[5].Chan[0].Status.a**
$07A308	**Acc24E2*x*[14].Chan[1].Status.a**	**Acc24E3[5].Chan[1].Status.a**	**CK3WAX[5].Chan[1].Status.a**
$07A310	**Acc24E2*x*[14].Chan[2].Status.a**	**Acc24E3[5].Chan[2].Status.a**	**CK3WAX[5].Chan[2].Status.a**
$07A318	**Acc24E2*x*[14].Chan[3].Status.a**	**Acc24E3[5].Chan[3].Status.a**	**CK3WAX[5].Chan[3].Status.a**
$07B200	**Acc24E2*x*[16].Chan[0].Status.a**	**Acc24E3[6].Chan[0].Status.a**	**CK3WAX[6].Chan[0].Status.a**
$07B208	**Acc24E2*x*[16].Chan[1].Status.a**	**Acc24E3[6].Chan[1].Status.a**	**CK3WAX[6].Chan[1].Status.a**
$07B210	**Acc24E2*x*[16].Chan[2].Status.a**	**Acc24E3[6].Chan[2].Status.a**	**CK3WAX[6].Chan[2].Status.a**
$07B218	**Acc24E2*x*[16].Chan[3].Status.a**	**Acc24E3[6].Chan[3].Status.a**	**CK3WAX[6].Chan[3].Status.a**
$07B300	**Acc24E2*x*[18].Chan[0].Status.a**	**Acc24E3[7].Chan[0].Status.a**	**CK3WAX[7].Chan[0].Status.a**
$07B308	**Acc24E2*x*[18].Chan[1].Status.a**	**Acc24E3[7].Chan[1].Status.a**	**CK3WAX[7].Chan[1].Status.a**
$07B310	**Acc24E2*x*[18].Chan[2].Status.a**	**Acc24E3[7].Chan[2].Status.a**	**CK3WAX[7].Chan[2].Status.a**
$07B318	**Acc24E2*x*[18].Chan[3].Status.a**	**Acc24E3[7].Chan[3].Status.a**	**CK3WAX[7].Chan[3].Status.a**

Notes:

1. The "***x***" in **Acc24E2*x*** above is "**A**" for ACC-24E2A boards, "**S**" for ACC-24E2S boards, or (null) for ACC-24E2 boards.
2. **Acc24E2*x*[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2*x*[*i*]**.
3. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.
4. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.
5. **Status.a** addresses can also be entered as **UVW.a**, but will report back as **Status.a**.

I*xx*91 to Motor[*x*].AbsPhasePosFormat

This parameter tells the PMAC how to interpret the absolute position data read from the specified register. (If setting up for direct-PWM control of a DC brush motor, the setting of this parameter does not matter.

ACC-84x Turbo PMAC Serial Encoder Interface

When using a Turbo PMAC with an ACC-84*x* board to read the absolute position from a serial encoder, **I*xx*91** is set to $*nn*0000, where the two hex digits $*nn* represent the number of bits to read. $*nn* can be in the range $08 to $20 (8 to 32).

- If the Power PMAC system also uses an ACC-84*x* interface, set **Motor[*x*].AbsPhasePosFormat** to $0008*nn*08 for the same formatting.

- If the Power PMAC uses the built-in standard interface, set **Motor[*x*].AbsPhasePosFormat** to $0000*nn*00 for the same formatting.

Hall Sensor Interface

- When using an ACC-24E2*x* UMAC board to receive (120°) Hall sensor inputs in a Power PMAC system, **Motor[*x*].AbsPhasePosFormat** should be set to $0400031C.

- When using a Power Clipper, Power Brick, or an ACC-24E3 UMAC board to receive (120°) Hall sensor inputs in a Power PMAC system, **Motor[*x*].AbsPhasePosFormat** should be set to $0400030C.

Motor[*x*].AbsPhasePosSf

In Turbo PMAC, the scaling of the absolute phase position sensor must be the same as the ongoing phase position sensor (with a special exception for Hall sensors). Therefore, there is not separate term for absolute phase position scaling in Turbo PMAC.

- If the absolute phase position sensor is the same as the ongoing phase position sensor (as with serial encoders), set **Motor[*x*].AbsPhasePosSf** equal to **Motor[*x*].PhasePosSf**.

- If Hall sensors are used for the absolute phase position, **Motor[*x*].AbsPhasePosSf** should be set to 170.667 if bit 22 (value $400000) of **I*xx*91** is 0, or to -170.667 if bit 22 of **I*xx*91** is set to 1 (for inverted direction sense).

- If setting up for direct-PWM control of a brush motor, set **Motor[*x*].AbsPhasePosSf** to 0.0 to force the commutation angle to zero.

I*xx*75 to Motor[*x*].AbsPhasePosOffset, Motor[*x*].AbsPhasePosForce

- For full-resolution absolute position sensors (e.g. serial encoders), set **Motor[*x*].AbsPhasePosOffset** equal to (**I*xx*75** / **I*xx*70**) for equivalent offset between sensor zero and commutation zero for absolute phase position reads.

- For Hall sensors, set **Motor[*x*].AbsPhasePosOffset** equal to 32 times the combined value in bits 16 – 21 of **I*xx*91** (which together have a range of 0 – 63).

- Set **Motor[*x*].AbsPhasePosForce** equal to ([**I*xx*75** * 2048] / [**I*xx*71** / **I*xx*70**]) for equivalent operation if using the **SETPHASE** command for phase correction in Turbo PMAC. (In Power PMAC, will use **Motor[*x*].PhasePos** = **Motor[*x*].AbsPhasePosForce** command instead.)

Motor Current Loop I-Variable Equivalents

These elements only need to be set in Power PMAC if **I*xx*01** in Turbo PMAC is equal to 1 and **I*xx*82** is greater than 0 to enable the digital current loop in Power PMAC.

Current-Loop Feedback Address

If **I*xx*82** is set to 0, set **Motor[*x*].pAdc** to 0 to disable the current loop in the Power PMAC motor. In this case, set **Motor[*x*].PwmSf** to its maximum value of 32,767 to match the Turbo PMAC sinewave mode output scaling.

If **I*xx*82** is greater than 0, the following tables provide the values for the standard configurations.

Turbo Clipper to Power Clipper

Turbo Clipper Current Feedback to Power Clipper

Turbo Clipper **I*xx*82**	Power Clipper **Motor[*x*].pAdc**
$078006	**Clipper[0].Chan[0].AdcAmp[0].a**
$07800E	**Clipper[0].Chan[1].AdcAmp[0].a**
$078016	**Clipper[0].Chan[2].AdcAmp[0].a**
$07801E	**Clipper[0].Chan[3].AdcAmp[0].a**
$078106	**Clipper[1].Chan[0].AdcAmp[0].a**
$07810E	**Clipper[1].Chan[1].AdcAmp[0].a**
$078116	**Clipper[1].Chan[2].AdcAmp[0].a**
$07811E	**Clipper[1].Chan[3].AdcAmp[0].a**

Notes:

1. **Clipper[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Clipper[*i*]**.

Turbo Brick to Power Brick

Turbo Brick Current Feedback to Power Brick

Turbo Brick **I*xx*82**	Power Brick **Motor[*x*].pAdc**
$078006	**PowerBrick[0].Chan[0].AdcAmp[0].a**
$07800E	**PowerBrick[0].Chan[1].AdcAmp[0].a**
$078016	**PowerBrick[0].Chan[2].AdcAmp[0].a**
$07801E	**PowerBrick[0].Chan[3].AdcAmp[0].a**
$078106	**PowerBrick[1].Chan[0].AdcAmp[0].a**
$07810E	**PowerBrick[1].Chan[1].AdcAmp[0].a**
$078116	**PowerBrick[1].Chan[2].AdcAmp[0].a**
$07811E	**PowerBrick[1].Chan[3].AdcAmp[0].a**

Notes:

1. **PowerBrick[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **PowerBrick[*i*]**.

Turbo UMAC to Power UMAC

Turbo UMAC Current Feedback to Power UMAC

Turbo UMAC **I*xx*82**	Power UMAC ACC-24E2 **Motor[*x*].pAdc**	Power UMAC ACC-24E3 **Motor[*x*].pAdc**
$078206	**Acc24E2[4].Chan[0].Adc[0].a**	**Acc24E3[0].Chan[0].AdcAmp[0].a**
$07820E	**Acc24E2[4].Chan[1].Adc[0].a**	**Acc24E3[0].Chan[1].AdcAmp[0].a**
$078216	**Acc24E2[4].Chan[2].Adc[0].a**	**Acc24E3[0].Chan[2].AdcAmp[0].a**
$07821E	**Acc24E2[4].Chan[3].Adc[0].a**	**Acc24E3[0].Chan[3].AdcAmp[0].a**
$078306	**Acc24E2[6].Chan[0].Adc[0].a**	**Acc24E3[1].Chan[0].AdcAmp[0].a**
$07830E	**Acc24E2[6].Chan[1].Adc[0].a**	**Acc24E3[1].Chan[1].AdcAmp[0].a**
$078316	**Acc24E2[6].Chan[2].Adc[0].a**	**Acc24E3[1].Chan[2].AdcAmp[0].a**
$07831E	**Acc24E2[6].Chan[3].Adc[0].a**	**Acc24E3[1].Chan[3].AdcAmp[0].a**
$079206	**Acc24E2[8].Chan[0].Adc[0].a**	**Acc24E3[2].Chan[0].AdcAmp[0].a**
$07920E	**Acc24E2[8].Chan[1].Adc[0].a**	**Acc24E3[2].Chan[1].AdcAmp[0].a**
$079216	**Acc24E2[8].Chan[2].Adc[0].a**	**Acc24E3[2].Chan[2].AdcAmp[0].a**
$07921E	**Acc24E2[8].Chan[3].Adc[0].a**	**Acc24E3[2].Chan[3].AdcAmp[0].a**
$079306	**Acc24E2[10].Chan[0].Adc[0].a**	**Acc24E3[3].Chan[0].AdcAmp[0].a**
$07930E	**Acc24E2[10].Chan[1].Adc[0].a**	**Acc24E3[3].Chan[1].AdcAmp[0].a**
$079316	**Acc24E2[10].Chan[2].Adc[0].a**	**Acc24E3[3].Chan[2].AdcAmp[0].a**
$07931E	**Acc24E2[10].Chan[3].Adc[0].a**	**Acc24E3[3].Chan[3].AdcAmp[0].a**
$07A206	**Acc24E2[12].Chan[0].Adc[0].a**	**Acc24E3[4].Chan[0].AdcAmp[0].a**
$07A20E	**Acc24E2[12].Chan[1].Adc[0].a**	**Acc24E3[4].Chan[1].AdcAmp[0].a**
$07A216	**Acc24E2[12].Chan[2].Adc[0].a**	**Acc24E3[4].Chan[2].AdcAmp[0].a**
$07A21E	**Acc24E2[12].Chan[3].Adc[0].a**	**Acc24E3[4].Chan[3].AdcAmp[0].a**
$07A306	**Acc24E2[14].Chan[0].Adc[0].a**	**Acc24E3[5].Chan[0].AdcAmp[0].a**
$07A30E	**Acc24E2[14].Chan[1].Adc[0].a**	**Acc24E3[5].Chan[1].AdcAmp[0].a**
$07A316	**Acc24E2[14].Chan[2].Adc[0].a**	**Acc24E3[5].Chan[2].AdcAmp[0].a**
$07A31E	**Acc24E2[14].Chan[3].Adc[0].a**	**Acc24E3[5].Chan[3].AdcAmp[0].a**
$07B206	**Acc24E2[16].Chan[0].Adc[0].a**	**Acc24E3[6].Chan[0].AdcAmp[0].a**
$07B20E	**Acc24E2[16].Chan[1].Adc[0].a**	**Acc24E3[6].Chan[1].AdcAmp[0].a**
$07B216	**Acc24E2[16].Chan[2].Adc[0].a**	**Acc24E3[6].Chan[2].AdcAmp[0].a**
$07B21E	**Acc24E2[16].Chan[3].Adc[0].a**	**Acc24E3[6].Chan[3].AdcAmp[0].a**
$07B306	**Acc24E2[18].Chan[0].Adc[0].a**	**Acc24E3[7].Chan[0].AdcAmp[0].a**
$07B30E	**Acc24E2[18].Chan[1].Adc[0].a**	**Acc24E3[7].Chan[1].AdcAmp[0].a**
$07B316	**Acc24E2[18].Chan[2].Adc[0].a**	**Acc24E3[7].Chan[2].AdcAmp[0].a**
$07B31E	**Acc24E2[18].Chan[3].Adc[0].a**	**Acc24E3[7].Chan[3].AdcAmp[0].a**

Notes:

1. **Acc24E2[*i*]** data structure can also be entered as **Gate1[*i*]**, but will report back as **Acc24E2[*i*]**.
2. **Acc24E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc24E3[*i*]**.

Turbo UMAC to CK3M

Turbo UMAC Current Feedback to CK3M

Turbo UMAC **I*xx*82**	CK3M with CK3W-AX*nnnn* **Motor[*x*].pAdc**
$078206	**CK3WAX[0].Chan[0].AdcAmp[0].a**
$07820E	**CK3WAX[0].Chan[1].AdcAmp[0].a**
$078216	**CK3WAX[0].Chan[2].AdcAmp[0].a**
$07821E	**CK3WAX[0].Chan[3].AdcAmp[0].a**
$078306	**CK3WAX[1].Chan[0].AdcAmp[0].a**
$07830E	**CK3WAX[1].Chan[1].AdcAmp[0].a**
$078316	**CK3WAX[1].Chan[2].AdcAmp[0].a**
$07831E	**CK3WAX[1].Chan[3].AdcAmp[0].a**
$079206	**CK3WAX[2].Chan[0].AdcAmp[0].a**
$07920E	**CK3WAX[2].Chan[1].AdcAmp[0].a**
$079216	**CK3WAX[2].Chan[2].AdcAmp[0].a**
$07921E	**CK3WAX[2].Chan[3].AdcAmp[0].a**
$079306	**CK3WAX[3].Chan[0].AdcAmp[0].a**
$07930E	**CK3WAX[3].Chan[1].AdcAmp[0].a**
$079316	**CK3WAX[3].Chan[2].AdcAmp[0].a**
$07931E	**CK3WAX[3].Chan[3].AdcAmp[0].a**
$07A206	**CK3WAX[4].Chan[0].AdcAmp[0].a**
$07A20E	**CK3WAX[4].Chan[1].AdcAmp[0].a**
$07A216	**CK3WAX[4].Chan[2].AdcAmp[0].a**
$07A21E	**CK3WAX[4].Chan[3].AdcAmp[0].a**
$07A306	**CK3WAX[5].Chan[0].AdcAmp[0].a**
$07A30E	**CK3WAX[5].Chan[1].AdcAmp[0].a**
$07A316	**CK3WAX[5].Chan[2].AdcAmp[0].a**
$07A31E	**CK3WAX[5].Chan[3].AdcAmp[0].a**
$07B206	**CK3WAX[6].Chan[0].AdcAmp[0].a**
$07B20E	**CK3WAX[6].Chan[1].AdcAmp[0].a**
$07B216	**CK3WAX[6].Chan[2].AdcAmp[0].a**
$07B21E	**CK3WAX[6].Chan[3].AdcAmp[0].a**
$07B306	**CK3WAX[7].Chan[0].AdcAmp[0].a**
$07B30E	**CK3WAX[7].Chan[1].AdcAmp[0].a**
$07B316	**CK3WAX[7].Chan[2].AdcAmp[0].a**
$07B31E	**CK3WAX[7].Chan[3].AdcAmp[0].a**

Notes:

1. **CK3WAX[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **CK3WAX[*i*]**.

Turbo PMAC MACRO to Power PMAC MACRO

Turbo PMAC MACRO Current Feedback to Power PMAC

Turbo PMAC **I*xx*82**	Power UMAC **Motor[*x*].pAdc** for ACC-5E Node 1 & 2 Feedback	Power PMAC **Motor[*x*].pAdc** for ACC-5E3 Node 1 & 2 Feedback
$078422	**Acc5E[0].Macro[0][1].a**	**Acc5E3[0].MacroInA[0][1].a**
$078426	**Acc5E[0].Macro[1][1].a**	**Acc5E3[0].MacroInA[1][1].a**
$07842A	**Acc5E[0].Macro[4][1].a**	**Acc5E3[0].MacroInA[4][1].a**
$07842E	**Acc5E[0].Macro[5][1].a**	**Acc5E3[0].MacroInA[5][1].a**
$078432	**Acc5E[0].Macro[8][1].a**	**Acc5E3[0].MacroInA[8][1].a**
$078436	**Acc5E[0].Macro[9][1].a**	**Acc5E3[0].MacroInA[9][1].a**
$07843A	**Acc5E[0].Macro[12][1].a**	**Acc5E3[0].MacroInA[12][1].a**
$07843E	**Acc5E[0].Macro[13][1].a**	**Acc5E3[0].MacroInA[13][1].a**
$079422	**Acc5E[1].Macro[0][1].a**	**Acc5E3[0].MacroInB[0][1].a**
$079426	**Acc5E[1].Macro[1][1].a**	**Acc5E3[0].MacroInB[1][1].a**
$07942A	**Acc5E[1].Macro[4][1].a**	**Acc5E3[0].MacroInB[4][1].a**
$07942E	**Acc5E[1].Macro[5][1].a**	**Acc5E3[0].MacroInB[5][1].a**
$079432	**Acc5E[1].Macro[8][1].a**	**Acc5E3[0].MacroInB[8][1].a**
$079436	**Acc5E[1].Macro[9][1].a**	**Acc5E3[0].MacroInB[9][1].a**
$07943A	**Acc5E[1].Macro[12][1].a**	**Acc5E3[0].MacroInB[12][1].a**
$07943E	**Acc5E[1].Macro[13][1].a**	**Acc5E3[0].MacroInB[13][1].a**
$07A422	**Acc5E[2].Macro[0][1].a**	**Acc5E3[1].MacroInA[0][1].a**
$07A426	**Acc5E[2].Macro[1][1].a**	**Acc5E3[1].MacroInA[1][1].a**
$07A42A	**Acc5E[2].Macro[4][1].a**	**Acc5E3[1].MacroInA[4][1].a**
$07A42E	**Acc5E[2].Macro[5][1].a**	**Acc5E3[1].MacroInA[5][1].a**
$07A432	**Acc5E[2].Macro[8][1].a**	**Acc5E3[1].MacroInA[8][1].a**
$07A436	**Acc5E[2].Macro[9][1].a**	**Acc5E3[1].MacroInA[9][1].a**
$07A43A	**Acc5E[2].Macro[12][1].a**	**Acc5E3[1].MacroInA[12][1].a**
$07A43E	**Acc5E[2].Macro[13][1].a**	**Acc5E3[1].MacroInA[13][1].a**
$07B422	**Acc5E[3].Macro[0][1].a**	**Acc5E3[1].MacroInB[0][1].a**
$07B426	**Acc5E[3].Macro[1][1].a**	**Acc5E3[1].MacroInB[1][1].a**
$07B42A	**Acc5E[3].Macro[4][1].a**	**Acc5E3[1].MacroInB[4][1].a**
$07B42E	**Acc5E[3].Macro[5][1].a**	**Acc5E3[1].MacroInB[5][1].a**
$07B432	**Acc5E[3].Macro[8][1].a**	**Acc5E3[1].MacroInB[8][1].a**
$07B436	**Acc5E[3].Macro[9][1].a**	**Acc5E3[1].MacroInB[9][1].a**
$07B43A	**Acc5E[3].Macro[12][1].a**	**Acc5E3[1].MacroInB[12][1].a**
$07B43E	**Acc5E[3].Macro[13][1].a**	**Acc5E3[1].MacroInB[13][1].a**

Notes:

1. **Acc5E[*i*]** data structure can also be entered as **Gate2[*i*]**, but will report back as **Acc5E[*i*]**.
2. **Acc5E3[*i*]** data structure can also be entered as **Gate3[*i*]**, but will report back as **Acc5E3[*i*]**.

Current Feedback ADC Mask

- For a given value of **I*xx*84**, add two zeros to the end of the hexadecimal representation of **I*xx*84** to get the value of **Motor[*x*].AdcMask** (e.g. if **I*xx*84** = $FFF000, set **Motor[*x*].AdcMask** to $FFF00000).

Current Loop Gain Terms

- Set **Motor[*x*].IiGain** equal to (8 * **I*xx*61**) for the same integral gain.
- Set **Motor[*x*].IpfGain** equal to (4 * **I*xx*62**) for the same forward-path proportional gain.
- Set **Motor[*x*].IpbGain** equal to (4 * **I*xx*76**) for the same back-path proportional gain.
- **I*xx*66** to **Motor[*x*].PwmSf**:
    - If using an ACC-24E2 UMAC axis board with a DSPGATE1 ASIC in the Power PMAC system, set **Motor[*x*].PwmSf** equal to **I*xx*66** for the same output scaling gain.
    - If using a Power Clipper, Power Brick, or ACC-24E3 UMAC axis board (which use the DSPGATE3 ASIC), set **Motor[*x*].PwmSf** equal to (**I7*m*00** * 16,384 / **I*xx*66**) for the same output scaling gain.
- Set bit 1 (value 2) of **Motor[*x*].PhaseMode** equal to bit 0 (value 1) of **I*xx*96**. These bits should only be set to 1 for direct-PWM control of DC brush motors and voice-coil motors.

Motor Servo Loop Parameters

Servo Algorithm Selection

- **I*xx*59** to **Motor[*x*].Ctrl**:
    - If **I*xx*59** bit 0 (value 1) is set to its default value of 0, set **Motor[*x*].Ctrl** to it default value of **Sys.ServoCtrl** to select the standard servo algorithm, which closely matches the Turbo PMAC standard algorithm.

      (Note: If **I*yy*00** for the motor in Turbo PMAC is set to 1 [not the default] to select the Extended Servo Algorithm, the user should use the Advanced Tuning controls in the IDE to tune the servo loop in Power PMAC to equivalent performance. This is very rare.)
    - If **I*xx*59** bit 0 (value 1) is set to 1 to select a user servo algorithm, the user will need to write a custom Power PMAC algorithm in C and specify its use for the motor through the IDE.

Servo Period Extension

- Set **Motor[*x*].Stime** equal to **I*xx*60** to specify the same number of cycles skipped between loop closures.

Servo Gain Terms

There is a straightforward conversion from the Turbo PMAC servo gain terms to the Power PMAC servo gain terms for the default algorith when the servo update frequency is kept the same. The gain-by-gain conversion is explained below.

(Note: If using the "direct microstepping" PWM control of stepper motors, as with the Brick LV or Clipper Stack amplifier, the technique is substantially different in Power PMAC, so the instruction below should not be used.. Refer to the User's Manual chapter *Setting Up Commutation and/or Current Loop* for step-by-step instructions.)

- Set **Motor[*x*].Servo.Kp** equal to (**I*xx*30** * **I*xx*08** / 2^{19}) for the same proportional gain.
- If **I*xx*96** bit 1 (value 2) is set to 0 (default) to have the velocity terms act after the integrator:
    - Set **Motor[*x*].Servo.Kvfb** equal to (**I*xx*31** * **I*xx*30** * **I*xx*09** / 2^{26}) for the same velocity feedback (derivative) gain.
    - Set **Motor[*x*].Servo.Kvff** equal to (**I*xx*31** * **I*xx*30** * **I*xx*08** / 2^{26}) for the same velocity feedforward gain.
    - Set **Motor[*x*].Servo.Kvifb** and **Motor[*x*].Servo.Kvifb** to 0.0.
- But if **I*xx*96** bit 1 (value 2) is set to 1 to have the velocity terms act before the integrator:
    - Set **Motor[*x*].Servo.Kvifb** equal to (**I*xx*31** * **I*xx*30** * **I*xx*09** / 2^{26}) for the same velocity feedback (derivative) gain.

- Set **Motor[*x*].Servo.Kviff** equal to (**I*xx*31** * **I*xx*30** * **I*xx*08** / 2^{26}) for the same velocity feedforward gain.
- Set **Motor[*x*].Servo.Kvfb** and **Motor[*x*].Servo.Kvfb** to 0.0.
- Set **Motor[*x*].Servo.Ki** equal to (**I*xx*33** / 2^{23}) for the same integral gain.
- Set **Motor[*x*].Servo.Kaff** equal to (**I*xx*35** * **I*xx*30** / 2^{26}) for the same acceleration feedforward gain.
- Set **Motor[*x*].Servo.Kfff** equal to **I*xx*68** for the same friction feedforward gain.

Mode-Switch Terms

- Set **Motor[*x*].Servo.SwZvInt** equal to **I*xx*34** for the same integration mode.
- Set **Motor[*x*].SwFffInt** equal to **I*xx*96** bit 1 (value 2) for the same friction feedforward mode.
- Set **Motor[*x*].SwPoly7** to 0 to limit polynomial filters like the "notch" to 2^{nd}-order.

Notch Polynomial Terms

- Set **Motor[*x*].Servo.Kc1** equal to **I*xx*36** for the same notch filter operation.
- Set **Motor[*x*].Servo.Kc2** equal to **I*xx*37** for the same notch filter operation.
- Set **Motor[*x*].Servo.Kd1** equal to **I*xx*38** for the same notch filter operation.
- Set **Motor[*x*].Servo.Kd2** equal to **I*xx*39** for the same notch filter operation.

Limiting Terms

- Set **Motor[*x*].Servo.MaxPosErr** equal to (**I*xx*67** / 16) for the same position error input limit.
- Set **Motor[*x*].Servo.MaxInt** equal to (**I*xx*63** * **I*xx*08** / 2^{23}) for the same integration limit.
- Set **Motor[*x*].MaxDac** equal to **I*xx*69** for the same output command limit.

Deadband Terms

- Set **Motor[*x*].Servo.BreakPosErr** equal to (**I*xx*65** / 16) for the same deadband size.
- Set **Motor[*x*].Servo.Kbreak** equal to ([**I*xx*64** + 16] / 16) for the same gain in the deadband zone.

Offset Terms

- If **I*xx*01** = 0 (no PMAC commutation), set **Motor[*x*].DacBias** equal to **I*xx*29** for the same servo command output offset.

Motor Safety Limits

- Set **Motor[*x*].FatalFeLimit** equal to (**I*xx*11** / 16) for the same fatal following error limit.
- Set **Motor[*x*].WarnFeLimit** equal to (**I*xx*12** / 16) for the same fatal following error limit.
- Set **Motor[*x*].MaxPos** equal to **I*xx*13** for the same positive software overtravel limit.
- Set **Motor[*x*].MinPos** equal to **I*xx*14** for the same negative software overtravel limit.
- Set **Motor[*x*].AbortTa** equal to (-1.0 / **I*xx*15**) for the same abort deceleration rate.
- Set **Motor[*x*].AbortTs** equal to 0.0 if you desire no S-curve in the abort deceleration (Turbo PMAC has none). Set **Motor[*x*].AbortTs** less than 0.0 to add S-curve to the abort deceleration.
- Set **Motor[*x*].MaxSpeed** equal to **I*xx*16** for the same maximum speed magnitude.
- Set **Motor[*x*].InvAmax** and **Motor[*x*].InvDmax** equal to (1.0 / **I*xx*17**) for the same acceleration limit.
- Set **Motor[*x*].I2tSet** equal to **I*xx*57** for the same continuous current limit.
- Set **Motor[*x*].I2tTrip** equal to (**I*xx*58** * 2^{30} * **Sys.ServoPeriod** /1000) for the same integrated current limit.
- Set **Motor[*x*].SoftLimitStopDis** equal to bit 14 ($4000) of **I*xx*24** for the same operation on hitting software position limits.
- Set **Motor[*x*].SoftLimitOffset** equal to **I*xx*41** for the same offset between calculation-time limits and execution-time limits.
- If **I*xx*24** bit 22 (value $400000) is 0, set bit 0 (value 1) of **Motor[*x*].FaultMode** to 1. If **I*xx*24** bit 22 is 1, set bit 0 of **Motor[*x*].FaultMode** to 0.
- If **I*xx*24** bit 8 (value $100) is 0, set bit 2 (value 4) of **Motor[*x*].FaultMode** to 1. If **I*xx*24** bit 8 is 1 set bit 2 of **Motor[*x*].FaultMode** to 1.

Motor Move Acceleration Parameters

If **I*xx*19** is set low enough to limit acceleration for jogging and/or homing search moves, which is true under the following conditions:

1. **I*xx*19** < **I*xx*22** / (**I*xx*20** + **I*xx*21**) for jog moves if **I*xx*20** > (2 * **I*xx*21**)
2. **I*xx*19** < **I*xx*22** / (2 * **I*xx*21**) for jog moves if **I*xx*20** <= (2 * **I*xx*21**)
3. **I*xx*19** < |**I*xx*23**| / (**I*xx*20** + **I*xx*21**) for homing search moves if **I*xx*20** > (2 * **I*xx*21**)
4. **I*xx*19** < |**I*xx*23**| / (2 * **I*xx*21**) for homing search moves if **I*xx*20** <= (2 * **I*xx*21**)

then:

- Set **Motor[*x*].JogTa** equal to (-1.0 / **I*xx*19**) for equivalent jogging and homing acceleration rates.
- Set **Motor[*x*].JogTs** equal to (-**I*xx*21** * **I*xx*19**) for equivalent jogging and homing jerk rates.

But if **I*xx*19** is *not* set low enough to limit acceleration for jogging and/or homing search moves (i.e. the above conditions are false):

then if **I*xx*20** > 2 * **I*xx*21**

- Set **Motor[*x*].JogTa** equal to (**I*xx*20** – **I*xx*21**) for the same acceleration time,
- Set **Motor[*x*].JogTs** equal to **I*xx*21** for the same S-curve time.

But if **I*xx*20** <= 2 * **I*xx*21**

- Set **Motor[*x*].JogTa** equal to (2 * **I*xx*21**) for the same acceleration time,
- Set **Motor[*x*].JogTs** equal to **I*xx*21** for the same S-curve time.

Motor Move Speed Parameters

- Set **Motor[*x*].JogSpeed** equal to **I*xx*22** for the same jogging speed magnitude.
- Set **Motor[*x*].HomeVel** equal to **I*xx*23** for the same homing search speed and direction.
- Set **Motor[*x*].RapidSpeedSel** equal to **I*xx*90** for the same choice of maximum or jog speed for rapid mode moves.

Motor Move Capture and Post-Trigger Parameters

- Set **Motor[*x*].HomeOffset** equal to (**I*xx*26** / 16) for the same offset between trigger and home positions.
- If **I*xx*97** is 0 or 1, set **Motor[*x*].CaptureMode** equal to **I*xx*97** for equivalent capture functionality.

- If **I*xx*97** is 2 or 3, set **Motor[*x*].CaptureMode** to 2 for equivalent capture functionality.

Motor Backlash Parameters

- Set **Motor[*x*].BlSize** equal to (**I*xx*86** / 16) for the same backlash magnitude.
- Set **Motor[*x*].BlSlewRate** equal to about (**I*xx*85** / 16 / *N*), where *N* is the best estimate of real-time interrupt cycles per Turbo PMAC background cycles (*N* = 2 is often a good estimate) for the same backlash takeup rate.
- Set **Motor[*x*].BlHysteresis** equal to (**I*xx*87** / 16) for the same backlash hysteresis.

Motor In-Position Function Parameters

- Set **Motor[*x*].InPosBand** equal to (**I*xx*28** / 16) for the same in-position band magnitude.
- If using the background in-position check in Turbo PMAC, set **Motor[*x*].InPosTime** equal to (**I*xx*88** * *N*), where *N* is the best estimate of servo cycles per Turbo PMAC background cycle (*N* = 5 is often a good estimate), for the same in-position settling time.
- If using the foreground in-position check in Turbo PMAC, set **Motor[*x*].InPosTime** to 0 for the same in-position settling time.

Motor Trajectory Filter Parameters

- If **I*xx*40** = 0.0, set **Motor[*x*].PreFilterEna** to 0 to disable the trajectory pre-filter.
- If **I*xx*40** > 0.0, set **Motor[*x*].PreFilterEna** to 1 to enable the trajectory pre-filter, updating every servo cycle.
    - Set **Motor[*x*].Pn0** equal to **I*xx*40** and **Motor[*x*].Pd1** equal to –**I*xx*40** for the same filter time constant.
    - Keep **Motor[*x*].Pn1**, **Pn2**, **Pn3**, **Pn4**, **Pd2**, **Pd3**, and **Pd4** at their default values of 0.0.

Coordinate System I-Variable Equivalents

In Turbo PMAC documentation, the coordinate-system number specification in these I-variables is given as ***sx***, as in **I*sx*13**, where ***sx*** is the coordinate system number plus 50. (For example, I5213 affects Coordinate System 2.) In Power PMAC documentation, the coordinate-system number specification is given as ***x***, as in **Coord[*x*].SegMoveTime**.

Turbo PMAC coordinate system numbers start with 1 (and go to 32). Power PMAC coordinate system numbers start with 0 (and go to 127), but Coordinate System 0 is rarely used to command a real group of axes in motion programs. This note explains the conversion from the setup of a Turbo PMAC coordinate system to the Power PMAC coordinate system of the same number.

In Power PMAC, coordinate systems numbered from 0 to (**Sys.MaxCoords** – 1) can be activated and configured. At the default value for **Sys.MaxCoords** of 16, C.S. 0 – 15 can be used. If C.S. 16 was used in the Turbo PMAC application, **Sys.MaxCoords** must be increased to a value of 17 or greater for compatibility.

Move Segmentation Parameters

- Set **Coord[*x*].SegMoveTime** equal to **I*sx*13** for the same segmentation time (or none).

- Set **Coord[*x*].SegOverride** equal to (**I*sx*15** + 1.0) for the same override percentage. Note that **Coord[*x*].SegOverride** is not a saved setup element in Power PMAC. At power-on/reset, it is always set to 1.0 ("real time").

- Set **Coord[*x*].SegOverrideSlew** equal to **I*sx*16** for the same override change rate.

Time Base Parameters

- **I*sx*93** to **Coord[*x*].pDesTimeBase**:

    - If **I*sx*93** is set to the default value of $002*y*00, where $y = x - 1$, set **Coord[*x*].pDesTimeBase** to the default value of **Coord[*x*].DesTimeBase.a** for the time base to be controlled by software commands.

    - If **I*sx*93** is set to a value of $0035*yy* to use an external time base frequency computed by the Encoder Conversion Table, set **Coord[*x*].pDesTimeBase** to **EncTable[*n*].DeltaPos.a** for the entry ***n*** that is computing this external frequency.

- Set **Coord[*x*].TimeBaseSlew** equal to (**I*sx*94** / 2^{23}) for the same rate of change of the time base override.

- Set **Coord[*x*].FeedHoldSlew** equal to (**I*sx*95** / 2^{23}) for the same rate of change of the override into and out of the feed hold state.

Kinematics Parameters

In Turbo PMAC, it is necessary to set **I*sx*50** to 1 to enable the use of the kinematic subroutines for the coordinate system. In Power PMAC, if the kinematic subroutines are present, they will automatically be used, so there is no comparable saved setup element.

Circle Move Parameters

- Set **Coord[*x*].MaxCirAccel** equal to **I*sx*78** for the same limit on circle-mode centripetal acceleration.

- **I*sx*97** to **Coord[*x*].MinArcLen**;

    - If **I*sx*97** = 0.0, set **Coord[*x*].MinArcLen** to 3.0 x 10^{-6} (3E-6 or 0.000003) for the same minimum arc length angle.

    - If **I*sx*97** > 0.0, set **Coord[*x*].MinArcLen** to (3.14 * **I*sx*97**) for the same minimum arc length angle.

- Set **Coord[*x*].RadiusErrorLimit** equal to **I*sx*96** for the same maximum permitted difference between start and end radius.

Blending and Cornering Control Parameters

- Set **Coord[*x*].InPosTimeOut** equal to **I*sx*81** for the same time limit (or none) on the in-position wait when blending is disabled between moves.

- Set **Coord[*x*].NoBlend** equal to **I*sx*92** for the same blending enable/disable control.

- Set **Coord[*x*].CornerBlendBp** equal to **I*sx*83** for the same corner angle threshold for blending.

- Set **Coord[*x*].CornerDwellBp** equal to **I*sx*85** for the same corner angle threshold for an automatic added dwell.

- Set **Coord[*x*].AddedDwellTime** equal to **I*sx*82** for the same dwell time inserted between sharp corner stops.

- Set **Coord[*x*].CCAddedArcBp** equal to **I*sx*99** for the same corner angle threshold for adding an arc move on a compensated outside corner.

- Set bit 0 (value 1) of **Coord[*x*].CCCtrl** equal to **I*sx*84** for the same stopping behavior on compensated outside corners with an added arc.

- Set **Coord[*x*].StepMode** equal to **I*sx*53** for the same single-step operation.

Velocity Control Parameters

- Set **Coord[*x*].FeedTime** equal to **I*sx*90** for the same time units for feedrate specification.

- Set **Coord[*x*].AltFeedRate** equal to **I*sx*86** for the same speed for non-feedrate axes.

- Set **Coord[*x*].MaxFeedRate** equal to **I*sx*99** for the same maximum speed for feedrate axes.

- Set **Coord[*x*].Tm** equal to (–**I*sx*89**) for the same power-on default feedrate. This will typically be overridden by an **F** or **tm** command in the motion program

- Set **Coord[*x*].RapidVelCtrl** equal to **I*x*79** for the same multi-axis rapid move coordination.

Acceleration Time Parameters

- If **I*x*87** > 2 * **I*x*88**:
    - Set **Coord[*x*].Ta** and **Coord[*x*].Td** equal to (**I*x*87** – **I*x*88**) for the same time at constant acceleration and deceleration in linear and circle mode moves.
    - Set **Coord[*x*].Ts** equal to **I*x*88** for the same S-curve time in these moves.
- If **I*x*87** <= 2 * **I*x*88**:
    - Set **Coord[*x*].Ta** and **Coord[*x*].Td** equal to **I*x*88** (any value from 0 to **I*x*88** will have the same effect).
    - Set **Coord[*x*].Ts** equal to **I*x*88** for the same S-curve time and no constant acceleration or deceleration time in linear and circle mode moves.

Lookahead Parameters

- Set **Coord[*x*].LHDistance** equal to **I*x*20** for the same number of segments to look ahead.
- The command settings of **I*x*21** are replaced by the **`lh>`**, **`lh<`**, and **`lh\`** program direct commands.

Countdown Timers

- Set **Sys.CdTimer[*i*]** equal to (**I*x*11** / [**I10** / 8388608]) or (**I*x*12** / [**I10** / 8388608]) for the same time delay in counting down to 0.

Internal Pointer Variables

Most PMAC applications access internal (memory) registers in user code. The methods for doing this can be different in Turbo PMAC and Power PMAC.

Methods of Accessing Internal Registers

Power PMAC provides more flexibility in accessing these internal registers than Turbo PMAC does. This means that the user must decide the best way to perform this access in Power PMAC.

Turbo PMAC Access

In Turbo PMAC, internal registers that are not saved setup variables are accessed through pointer (M) variables. Unlike the pre-defined I-variables for the saved elements, the user is free to define M-variables in his own manner.

However, the Turbo PMAC Software Reference Manual provides an extensive list of "suggested" M-variables, and most users utilize this list, at least as a starting point.

Power PMAC Access

In Power PMAC, all of these internal registers have pre-defined English-language element names. There is no need to define an M-variable to access these registers; they can be accessed directly by using the element names.

The Power PMAC Software Reference Manual has a chapter – *Power PMAC Turbo Suggested M-variable Equivalents* – detailing the Power PMAC element names that correspond to the Turbo PMAC suggested M-variables for various configurations. That chapter should be used for the detailed conversion.

Notes on the Conversion

It is most efficient in Power PMAC to use the data structure elements directly. It is possible to assign an M-variable to the data structure element, but this adds some computational overhead to the access of the register.

Still, it may be desirable to assign an M-variable to the element with a defined or declared name, in order to match the Turbo PMAC name better than the fixed name of the Power PMAC element. This is especially true if the desired names are in a language other than English.

For example, in Turbo PMAC:

```
M140->Y:$0000C0,0,1                ; Motor 1 in-position status bit
#define XaxisInPosition M140
```

To match this in Power PMAC:

```
ptr XaxisInPosition->Motor[1].InPos // Motor 1 in-position status bit
```

or:

```
#define XaxisInPosition Motor[1].InPos    // Motor 1 in-pos status bit
```

Internal Motor Position Registers

Many applications will access internal motor position registers directly. Care must be taken in converting these applications from Turbo PMAC to Power PMAC.

Scaling of Motor Position Registers

The suggested motor M-variables in Turbo PMAC that contain position and velocity information are fixed-point registers, but because the resolution provided by fractional-count information is important for high-performance control, they are in units that provide this resolution, usually 1/[I*xx*08*32] counts, sometimes in 1/16 counts, of the motor.

The motor position and velocity elements in Power PMAC are floating-point registers. As such they can be scaled in whole counts (or other scaled motor units), with the floating-point format providing the needed fractional resolution. So when converting from Turbo PMAC to Power PMAC, the scaling will be different. Usually this just means eliminating the scaling terms that were required in Turbo PMAC to get to units of counts.

Motor Reference Position

In Turbo PMAC, the internal motor position registers are referenced to the motor zero ("home") position. While the position at power-on/reset is automatically the zero position to start, in virtually all applications the motor zero position is then redefined by a homing search move or an absolute sensor read.

In Power PMAC, the internal motor position registers are always referenced to the power-on/reset position, even after the motor zero position has been redefined by a homing search move or absolute sensor read. The distance from power-on/reset position found at the homing search trigger or the absolute sensor zero is automatically stored in **Motor[*x*].HomePos**.

Example

Using the motor instantaneous desired position register as an example, to get the position in counts relative to motor zero, we have in Turbo PMAC:

```
M161->D:$000088                    ; #1 des pos (1/[I108*32] cts)

P161 = M161 / (I108 * 32)          ; Convert to whole counts
```

The equivalent in Power PMAC (presuming motor units are counts) is:

```
P161 = Motor[1].DesPos - Motor[1].HomePos // Relative to motor zero pos
```

Range and Rollover Issues

Turbo PMAC's fixed-point motor position registers have the potential to roll over when they reach the maximum positive or negative value (~+/-45 billion counts at the default I*xx*08 value). For directly commanded motor moves, such as jogging moves, the motor can travel indefinitely.

However, when the motor is commanded from an axis in a motion program, the floating-point axis position values cannot be made to handle the rollover of the corresponding motor, so the rollover point must be considered the limit of commanded motion.

Power PMAC's floating-point motor position registers cannot roll over, even for directly commanded motor moves. However, the double-precision representation provides enough resolution and range for all conceivable applications. Motor registers maintain fractional resolution equivalent to Turbo PMAC's default resolution over a range of over +/-1 trillion counts, and full-count resolution over a range of +/-4 quadrillion counts.

Input/Output Pointer Variables

In PMAC controllers, general-purpose inputs and outputs are usually accessed through "M" pointer variables. In Turbo PMAC, these pointers are defined using numerical addresses of the I/O registers. In Power PMAC, these pointers are usually defined using the data structure names for the registers. This section shows how to convert Turbo PMAC definitions to Power PMAC definitions.

Clipper General-Purpose Digital I/O

The Clipper boards have two general-purpose digital I/O ports, each with 16 5-volt I/O points, direction-selectable in sets of 8.

Clipper JTHW Port

The JTHW port on the Turbo Clipper and Power Clipper boards can be used for 16 individual digital I/O points, or as a single collective 8-in/8-out port to interface to multiplexed I/O accessories such as the ACC-34*x* family.

The following table shows the I/O points for this port, and how they can be accessed in Turbo Clipper and Power Clipper. The I/O points in the Power Clipper can also be defined using the "generic" element **Gate3[0].GpioData[0]** (where it is documented in the Software Reference Manual), but the definitions will report back using the **Clipper[0]** alias for the element.

Pin(s)	Suggested Turbo M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
SEL0	**M40**	Y:$78402,8	**Clipper[0].GpioData[0].0**
SEL1	**M41**	Y:$78402,9	**Clipper[0].GpioData[0].1**
SEL2	**M42**	Y:$78402,10	**Clipper[0].GpioData[0].2**
SEL3	**M43**	Y:$78402,11	**Clipper[0].GpioData[0].3**
SEL4	**M44**	Y:$78402,12	**Clipper[0].GpioData[0].4**
SEL5	**M45**	Y:$78402,13	**Clipper[0].GpioData[0].5**
SEL6	**M46**	Y:$78402,14	**Clipper[0].GpioData[0].6**
SEL7	**M47**	Y:$78402,15	**Clipper[0].GpioData[0].7**
SEL0-7	**M48**	Y:$78402,8,8	**Clipper[0].GpioData[0].0.8**
DAT0	**M50**	Y:$78402,0	**Clipper[0].GpioData[0].8**
DAT1	**M51**	Y:$78402,1	**Clipper[0].GpioData[0].9**
DAT2	**M52**	Y:$78402,2	**Clipper[0].GpioData[0].10**
DAT3	**M53s**	Y:$78402,3	**Clipper[0].GpioData[0].11**
DAT4	**M54**	Y:$78402,4	**Clipper[0].GpioData[0].12**
DAT5	**M55**	Y:$78402,5	**Clipper[0].GpioData[0].13**
DAT6	**M56**	Y:$78402,6	**Clipper[0].GpioData[0].14**
DAT7	**M57**	Y:$78402,7	**Clipper[0].GpioData[0].15**
DAT0-7	**M58**	Y:$78402,0,8	**Clipper[0].GpioData[0].8.8**

Buffer Directions

While the input/output direction can be set individually for each point in the ASIC, both Turbo Clipper and Power Clipper boards have byte-wide buffer ICs whose direction is set collectively with a jumper.

Jumper E15 sets the direction of the buffer for the SEL0 – SEL7 I/O points (ON is input, OFF is output) on both boards. This jumper must be OFF to use multiplexed I/O accessories.

Jumper E14 set the direction of the buffer for the DAT0 – DAT7 I/O points (ON is input, OFF is output) on both boards. This jumper must be ON to use multiplexed I/O accessories.

ASIC I/O Point Setup

Turbo Clipper Setup

In Turbo Clipper, each of the 16 I/O points on the JTHW port must be configured as to use (general-purpose I/O or servo use), direction (input or output), and inversion (high-true or low-true). At power-on/reset, Turbo Clipper automatically sets all 16 I/O points on the JTHW port to general-purpose I/O, non-inverting, with the DAT0 – DAT7 points set to inputs and the SEL0 – SEL7 points set to outputs.

If you want to use the JTHW port in this manner (and to use multiplexed I/O accessories, you must), you do not need to perform any setup for the port. If you want any change from this configuration, you must specify the change in your application after every power-on/reset. This is usually done in a "power-on" PLC program that executes once and disables itself.

To do this, there will be M-variables defined to some or all of the configuration registers, something like:

```
M30->Y:$78406,0,16      ; Use-control bits for all 16 I/O points
M31->X:$78400,0,8       ; Direction control bits for DAT0 – DAT7
M32->X:$78400,8,8       ; Direction control bits for SEL0 – SEL
M33->X:$78406,0,16      ; Inversion control for all 16 I/O points
```

Then the power-on PLC could look something like:

```
OPEN PLC 1 CLEAR
M30=$FFFF               ; All 16 points general-purpose I/O
M31=$FF                 ; MO1 – MO8 outputs
M32=$00                 ; MI1 – MI8 inputs
M33=$FFFF               ; All 16 points inverting
DISABLE PLC 1           ; So only runs once
CLOSE
```

Power Clipper Setup

In Power Clipper, all 16 of the I/O points on the JTHW port are dedicated I/O points. Each point must be configured as to direction (input or output) and inversion (high-true or low-true). This is done with the individual bits of saved setup elements **Clipper[0].GpioDir[0]** and **Clipper[0].GpioPol[0]**, respectively. Once the values of these elements are set and saved, the port will automatically be set up properly at every power-on/reset.

(Note: These setup elements are write-protected, so **Sys.WpKey** must be set to $AAAAAAAA to permit you to change their values.)

Each bit of **Clipper[0].GpioDir[0]** specifies the direction of the matching bit of **Clipper[0].GpioData[0]**. If the direction bit is 0, the I/O point is set up as an input. If the direction bit is 1, the I/O point is set up as an output.

To set up the JTHW port in the Power Clipper to match the default configuration of the Turbo Clipper, set **Clipper[0].GpioDir[0]** to $*xxxx*00FF, making the DAT0 – DAT7 lines inputs and the SEL0 – SEL7 lines outputs. The "*xxxx*" hex digits control the directions of the I/O points on the JIO port, covered below.

To set up the JTHW port in the Power Clipper to match a different Turbo Clipper configuration, set bits 0 – 7 (the last two hex digits) of **Clipper[0].GpioDir[0]** equal to bits 8 – 15 (the previous two hex digits) of X:$78402 in Turbo Clipper, and bits 8 – 15 of **Clipper[0].GpioDir[0]** equal to bits 0 – 7 of X:$78402.

Each bit of **Clipper[0].GpioPol[0]** specifies the inversion polarity of the matching bit of **Clipper[0].GpioData[0]**. If the polarity bit is 0, the I/O point is set up as an high-true (5V ⇔ 1). If the polarity bit is 1, the I/O point is set up as low-true (0V ⇔ 1).

To set up the JTHW port in the Power Clipper to match the default configuration of the Turbo Clipper, set **Clipper[0].GpioPol[0]** to $*xxxx*0000, making all lines non-inverting. The "*xxxx*" hex digits control the directions of the I/O points on the JIO port, covered below.

To set up the JTHW port in the Power Clipper to match a different Turbo Clipper configuration, set bits 0 – 7 (the last two hex digits) of **Clipper[0].GpioPol[0]** equal to bits 8 – 15 (the previous two hex digits) of Y:$78406 in Turbo Clipper, and bits 8 – 15 of **Clipper[0].GpioDir[0]** equal to bits 0 – 7 of Y:$78406.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M40->Y:$78402,8                // SEL0 signal
#define LeftClampOn     M40
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr LeftClampOn->Clipper[0].GpioData[0].0
```

Or it can be done with a simple text substitution:

```
#define LeftClampOn     Clipper[0].GpioData[0].0
```

Use of JTHW for Multiplexed I/O

With the SEL0 – SEL7 lines all configured for outputs in hardware and software, and the DAT0 – DAT7 lines all configured for inputs, the JTHW port can interface to large numbers of multiplexed I/O on ACC-34*x* and compatible accessory boards.

Turbo PMAC Multiplexed I/O Support

On Turbo Clipper, if **I29** is set to its default value of 0, the JTHW port will be used for any multiplexed I/O.

On Turbo UMAC, if **I29** is set to its default value of 0, the JTHW port on the first ACC-5E will be used for any multiplexed I/O.

On any Turbo PMAC, if **I27** is set to its default value of 0, input values are expected on the DAT0 line. If **I27** is set to 1, input values are expected on the DAT7 line (which requires a changed jumper setting on the accessory board).

The data on a 32-bit port of an ACC-34*x* board is accessed using a special form of M-variable. The definition of this M-variable is of the form:

```
M{constant}->TWS:{port address}
```

Each ACC-34*x* board has a base port address, set by its DIP-switch block, that is a multiple of 8. The address of Port A on the board, usually the 32-bit input port, is (base address + 1). The address of Port B on the board, usually the 32-bit output port, is (base address + 6).

The I/O on the ACC-34*x* boards is accessed on an as-needed basis. When an M-variable assigned to an input port is read in a command, the input data from that port is copied into Turbo PMAC. It is usually copied into a 32-bit integer holding variable.

When an M-variable assigned to an output port is written to in a command, the output data for that power is copied from Turbo PMAC. It is usually copied from a 32-bit integer holding variable.

Power Clipper Multiplexed I/O Support

On Power Clipper, the **MuxIo** data structure elements control how the multiplexed I/O is configured.

- Set **MuxIo.pOut** to **Clipper[0].GpioData[0].a** and set **MuxIo.OutBit** to 0 to use the SEL0 – SEL7 lines on the Power Clipper JTHW port for multiplexed inputs.

- If **I27** is 0, set **MuxIo.pIn** to **Clipper[0].GpioData[0].a** and set **MuxIo.InBit** to 8 to use the DAT0 line on the Power Clipper JTHW port for multiplexed inputs.

- If **I27** is 1, set **MuxIo.pIn** to **Clipper[0].GpioData[0].a** and set **MuxIo.InBit** to 15 to use the DAT7 line on the Power Clipper JTHW port for multiplexed inputs. (This requires a changed jumper setting on the accessory board.)

- Set **MuxIo.Enable** to 1 to permit multiplexed data transfers over this port.

- Set **MuxIo.ClockPeriod** to the default value of 0 if you want the serial data transfers over this port to go at the fastest possible frequency. This could lead to bit rates of over 1 MHz, which may only be supported for 1 or 2 accessories at short distances.

- Set **MuxIo.ClockPeriod** to a non-zero value to specify the minimum period in microseconds for each bit to be transferred. Larger values may be needed for long distances and noisy environments. (Note: When using the JTHW port on the Power Clipper or ACC-5E3, only one input bit can be read each phase cycle [default 110 microseconds]. This requires a setting for **MuxIo.ClockPeriod** greater than the phase cycle period.)

Each ACC-34*x* board on the port has an index ***n*** (***n*** = 0 to 31) determined by the DIP switch settings on the board, with ***n*** = *{Turbo Base Address}* / 8.

- To enable Port A on board ***n*** for inputs, set **MuxIo.PortA[*n*].Enable** to 1 and **MuxIo.PortA[*n*].Dir** to 0.

- To enable Port B on board ***n*** for outputs, set **MuxIo.PortB[*n*].Enable** to 1 and **MuxIo.PortB[*n*].Dir** to 1.

The multiplexed data transfers over this port can either be done on an "on-demand" basis as in Turbo PMAC, or at a regular frequency with automatic transfers at that frequency.

- Set **MuxIo.UpdatePeriod** to the default value of 0 if you want to keep the "on-demand" nature of multiplexed data transfers of Turbo PMAC. In this case, set **MuxIo.PortA[*n*].Enable** or **MuxIo.PortB[*n*].Enable** to 1 each time you want to start a data transfer with that accessory port. Power PMAC will automatically reset the port's **Enable** element to 0 when this transfer is complete.

- Set **MuxIo.UpdatePeriod** to a non-zero value if you want automatic regular data transfers with each enabled accessory port. The value specifies the period in milliseconds. While this mechanism is different from Turbo PMAC, it may allow the application software to be more like Turbo PMAC, as this software simply accesses the holding registers in memory in the same manner as it would the TWS-format M-variable in Turbo PMAC.

To access data for multiplexed I/O transfers, the application will use 32-bit unsigned integer element **MuxIo.PortA[*n*].Data** for an accessory input port, and 32-bit unsigned integer element **MuxIo.PortB[*n*].Data** for an accessory output port.

If **MuxIo.UpdatePeriod** is 0 for on-demand transfers:

- Set **MuxIo.PortA[*n*].Enable** to 1 when you want to read an input port. Wait for this **Enable** element to be set to 0, then read the values in the bits of **MuxIo.PortA[*n*].Data** to access the port's input data.

- Write the values you want into the bits of **MuxIo.PortB[*n*].Data**, then set **MuxIo.PortB[*n*].Enable** to 1 to start the transfer to the accessory's output port.

If **MuxIo.UpdatePeriod** is greater than 0 for automatic repeated transfers:

- Simply read the values in the bits of **MuxIo.PortA[*n*].Data** to access the port's input data from the most recent cycle's transfer.

- Simply write the values you want into the bits of **MuxIo.PortB[*n*].Data**. These will then automatically be transferred to the accessory's output port on the next cycle.

Clipper JIO Port

The JIO port on the Turbo Clipper and Power Clipper boards provides 16 individual digital I/O points. The following table shows the I/O points for this port, and how they can be accessed in Turbo Clipper and Power Clipper. The I/O points in the Power Clipper can also be defined using the "generic" element **Gate3[0].GpioData[0]** (where it is documented in the Software Reference Manual), but the definitions will report back using the **Clipper[0]** alias for the element.

Pin(s)	Suggested Turbo M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
MO1	**M0**	Y:$78400,0	**Clipper[0].GpioData[0].16**
MO2	**M1**	Y:$78400,1	**Clipper[0].GpioData[0].17**
MO3	**M2**	Y:$78400,2	**Clipper[0].GpioData[0].18**
MO4	**M3**	Y:$78400,3	**Clipper[0].GpioData[0].19**
MO5	**M4**	Y:$78400,4	**Clipper[0].GpioData[0].20**
MO6	**M5**	Y:$78400,5	**Clipper[0].GpioData[0].21**
MO7	**M6**	Y:$78400,6	**Clipper[0].GpioData[0].22**
MO8	**M7**	Y:$78400,7	**Clipper[0].GpioData[0].23**
MI1	**M8**	Y:$78400,8	**Clipper[0].GpioData[0].24**
MI2	**M9**	Y:$78400,9	**Clipper[0].GpioData[0].25**
MI3	**M10**	Y:$78400,10	**Clipper[0].GpioData[0].26**
MI4	**M11**	Y:$78400,11	**Clipper[0].GpioData[0].27**
MI5	**M12**	Y:$78400,12	**Clipper[0].GpioData[0].28**
MI6	**M13**	Y:$78400,13	**Clipper[0].GpioData[0].29**
MI7	**M14**	Y:$78400,14	**Clipper[0].GpioData[0].30**
MI8	**M15**	Y:$78400,15	**Clipper[0].GpioData[0].31**

Buffer Directions

While the input/output direction can be set individually for each point in the ASIC, both Turbo Clipper and Power Clipper boards have byte-wide buffer ICs whose direction is set collectively with a jumper.

Jumper E16 sets the direction of the buffer for the MO1 – MO8 I/O points (ON is input, OFF is output) on both boards.

Jumper E17 sets the direction of the buffer for the MI1 – MI8 I/O points (ON is input, OFF is output) on both boards.

ASIC I/O Point Setup

Turbo Clipper Setup

In Turbo Clipper, each of the 16 I/O points on the JIO port must be configured as to use (general-purpose I/O or servo use), direction (input or output), and inversion (high-true or low-true). At power-on/reset, Turbo Clipper automatically sets all 16 I/O points on the JIO port to general-purpose I/O, non-inverting, and inputs.

If you want to use the JIO port in this manner, you do not need to perform any setup for the port. If you want any change from this configuration, you must specify the change in your application after every power-on/reset. This is usually done in a "power-on" PLC program that executes once and disables itself.

To do this, there will be M-variables defined to some or all of the configuration registers, something like:

```
M34->Y:$78404,0,16      ; Use control bits for all 16 I/O points
M35->X:$78400,0,8       ; Direction control bits for MO1 – MO8
M36->X:$78400,8,8       ; Direction control bits for MI1 – MI8
M37->X:$78404,0,16      ; Inversion control for all 16 I/O points
```

Then the power-on PLC could look something like:

```
OPEN PLC 1 CLEAR
M34=$FFFF               ; All 16 points general-purpose I/O
M35=$FF                 ; MO1 – MO8 outputs
M36=$00                 ; MI1 – MI8 inputs
M37=$FFFF               ; All 16 points inverting
DISABLE PLC 1           ; So only runs once
CLOSE
```

Power Clipper Setup

In Power Clipper, all 16 of the I/O points on the JIO port are dedicated I/O points. Each point must be configured as to direction (input or output) and inversion (high-true or low-true). This is done with the individual bits of saved setup elements **Clipper[0].GpioDir[0]** and **Clipper[0].GpioPol[0]**, respectively. Once the values of these elements are set and saved, the port will automatically be set up properly at every power-on/reset.

(Note: These setup elements are write-protected, so **Sys.WpKey** must be set to $AAAAAAAA to permit you to change their values.)

Each bit of **Clipper[0].GpioDir[0]** specifies the direction of the matching bit of **Clipper[0].GpioData[0]**. If the direction bit is 0, the I/O point is set up as an input. If the direction bit is 1, the I/O point is set up as an output.

To set up the JIO port in the Power Clipper to match the default configuration of the Turbo Clipper, set **Clipper[0].GpioDir[0]** to $0000*xxxx*, making all lines inputs. The "*xxxx*" hex digits control the directions of the I/O points on the JTHW port, covered above.

To set up the JIO port in the Power Clipper to match a different Turbo Clipper configuration, set bits 16 – 31 (the first four hex digits) of **Clipper[0].GpioDir[0]** equal to bits 0 – 15 (the last four hex digits) of X:$78400 in Turbo Clipper.

Each bit of **Clipper[0].GpioPol[0]** specifies the inversion polarity of the matching bit of **Clipper[0].GpioData[0]**. If the polarity bit is 0, the I/O point is set up as high-true (5V ⇔ 1). If the polarity bit is 1, the I/O point is set up as low-true (0V ⇔ 1).

To set up the JIO port in the Power Clipper to match the default configuration of the Turbo Clipper, set **Clipper[0].GpioPol[0]** to $0000*xxxx*, making all lines non-inverting. The "*xxxx*" hex digits control the directions of the I/O points on the JTHW port, covered above.

To set up the JIO port in the Power Clipper to match a different Turbo Clipper configuration, set bits 0 – 7 (the last two hex digits) of **Clipper[0].GpioPol[0]** equal to bits 8 – 15 (the previous two hex digits) of Y:$78406 in Turbo Clipper, and bits 8 – 15 of **Clipper[0].GpioDir[0]** equal to bits 0 – 7 of Y:$78406.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M1->Y:$78400,8                // M01 signal
#define LeftClampOn     M1
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr LeftClampOn->Clipper[0].GpioData[0].16
```

Or it can be done with a simple text substitution:

```
#define LeftClampOn     Clipper[0].GpioData[0].16
```

Brick General-Purpose Digital I/O

The PMAC Brick products (Brick AC, Brick LV, and Brick Controller) come standard with a 16-in/8-out general-purpose digital I/O port. Optionally, an identical second port can be provided.

Brick Standard GPIO Port (J6)

The J6 port on the Turbo Brick and Power Brick boards provides 16 digital input points and 8 digital output points. The following table shows the I/O points for this port, and how they can be accessed in Turbo Brick and Power Brick. The I/O points in the Power Brick can also be defined using the "generic" element **Gate3[0].GpioData[0]** (where it is documented in the Software Reference Manual), but the definitions will report back using the **PowerBrick[0]** alias for the element.

Pin	Function	Suggested Turbo M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
GPI1	Input 01	**M1**	Y:$78800,0	**PowerBrick[0].GpioData[0].0**
GPI2	Input 02	**M2**	Y:$78800,1	**PowerBrick[0].GpioData[0].1**
GPI3	Input 03	**M3**	Y:$78800,2	**PowerBrick[0].GpioData[0].2**
GPI4	Input 04	**M4**	Y:$78800,3	**PowerBrick[0].GpioData[0].3**
GPI5	Input 05	**M5**	Y:$78800,4	**PowerBrick[0].GpioData[0].4**
GPI6	Input 06	**M6**	Y:$78800,5	**PowerBrick[0].GpioData[0].5**
GPI7	Input 07	**M7**	Y:$78800,6	**PowerBrick[0].GpioData[0].6**
GPI8	Input 08	**M8**	Y:$78800,7	**PowerBrick[0].GpioData[0].7**
GPI9	Input 09	**M9**	Y:$78801,0	**PowerBrick[0].GpioData[0].8**
GPI10	Input 10	**M10**	Y:$78801,1	**PowerBrick[0].GpioData[0].9**
GPI11	Input 11	**M11**	Y:$78801,2	**PowerBrick[0].GpioData[0].10**
GPI12	Input 12	**M12**	Y:$78801,3	**PowerBrick[0].GpioData[0].11**
GPI13	Input 13	**M13**	Y:$78801,4	**PowerBrick[0].GpioData[0].12**
GPI14	Input 14	**M14**	Y:$78801,5	**PowerBrick[0].GpioData[0].13**
GPI15	Input 15	**M15**	Y:$78801,6	**PowerBrick[0].GpioData[0].14**
GPI16	Input 16	**M16**	Y:$78801,7	**PowerBrick[0].GpioData[0].15**
GPO1	Output 01	**M33**	Y:$78802,0	**PowerBrick[0].GpioData[0].16**
GPO2	Output 02	**M34**	Y:$78802,1	**PowerBrick[0].GpioData[0].17**
GPO3	Output 03	**M35**	Y:$78802,2	**PowerBrick[0].GpioData[0].18**
GPO4	Output 04	**M36**	Y:$78802,3	**PowerBrick[0].GpioData[0].19**
GPO5	Output 05	**M37**	Y:$78802,4	**PowerBrick[0].GpioData[0].20**
GPO6	Output 06	**M38**	Y:$78802,5	**PowerBrick[0].GpioData[0].21**
GPO7	Output 07	**M39**	Y:$78802,6	**PowerBrick[0].GpioData[0].22**
GPO8	Output 08	**M40**	Y:$78802,7	**PowerBrick[0].GpioData[0].23**

The I/O circuitry on this port is fixed in both Turbo Brick and Power Brick. The input pins can only be used as inputs, and the output pins can only be used as outputs. The selection of sinking or sourcing for both inputs and outputs is made by wiring choices.

ASIC I/O Point Setup

Turbo Brick Setup

All aspects of the digital I/O control in the Turbo Brick are fixed: use, direction, and inversion. There is nothing to set up, and nothing can be changed from the default.

Power Brick Setup

In Power Brick, two saved setup elements must be configured properly to use the general-purpose I/O. The factory default values for these elements set up the port to work properly; these should not be changed from the default.

(Note: These setup elements are write-protected, so **Sys.WpKey** must be set to $AAAAAAAA to permit you to change their values.)

Each bit of **PowerBrick[0].GpioDir[0]** specifies the direction of the matching bit of **PowerBrick[0].GpioData[0]**. If the direction bit is 0, the I/O point is set up as an input. If the direction bit is 1, the I/O point is set up as an output. To use the I/O port on the Power Brick, **PowerBrick[0].GpioDir[0]** *must* be set to the default of $00FF0000.

Each bit of **PowerBrick[0].GpioPol[0]** specifies the inversion polarity of the matching bit of **Clipper[0].GpioData[0]**. If the polarity bit is 0, the output point is set up as high-true (5V ⇔ 1) from the ASIC, which turns the output transistor on. The input point is set up as high-true (5V ⇔ 1) into the ASIC, which corresponds to a non-conducting input.

If the polarity bit is 1, the output point is set up as low-true (0V ⇔ 1) from the ASIC, which turns the output transistor off (many find this confusing). The input point is set up as low-true (0V ⇔ 1) into the ASIC, which corresponds to a non-conducting input.

To set up the port in the Power Brick to match the configuration of the Turbo Brick, set **PowerBrick[0].GpioPol[0]** to the default of $00000000, making all lines non-inverting.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M33->Y:$78802,0                // GPO1 signal
#define LeftClampOn     M33
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr LeftClampOn->PowerBrick[0].GpioData[0].16
```

Or it can be done with a simple text substitution:

```
#define LeftClampOn     PowerBrick[0].GpioData[0].16
```

Brick Optional GPIO Port (J7)

The J7 port on the Turbo Brick and Power Brick boards provides an additional 16 digital input points and 8 digital output points. The following table shows the I/O points for this port, and how they can be accessed in Turbo Brick and Power Brick. The I/O points in the Power Brick can also be defined using the "generic" element **Gate3[1].GpioData[0]** (where it is documented in the Software Reference Manual), but the definitions will report back using the **PowerBrick[1]** alias for the element.

Pin	Function	Suggested Turbo M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
GPI17	Input 17	**M17**	Y:$78803,0	**PowerBrick[1].GpioData[0].0**
GPI18	Input 18	**M18**	Y:$78803,1	**PowerBrick[1].GpioData[0].1**
GPI19	Input 19	**M19**	Y:$78803,2	**PowerBrick[1].GpioData[0].2**
GPI20	Input 20	**M20**	Y:$78803,3	**PowerBrick[1].GpioData[0].3**
GPI21	Input 21	**M21**	Y:$78803,4	**PowerBrick[1].GpioData[0].4**
GPI22	Input 22	**M22**	Y:$78803,5	**PowerBrick[1].GpioData[0].5**
GPI23	Input 23	**M23**	Y:$78803,6	**PowerBrick[1].GpioData[0].6**
GPI24	Input 24	**M24**	Y:$78803,7	**PowerBrick[1].GpioData[0].7**
GPI25	Input 25	**M25**	Y:$78804,0	**PowerBrick[1].GpioData[0].8**
GPI26	Input 26	**M26**	Y:$78804,1	**PowerBrick[1].GpioData[0].9**
GPI27	Input 27	**M27**	Y:$78804,2	**PowerBrick[1].GpioData[0].10**
GPI28	Input 28	**M28**	Y:$78804,3	**PowerBrick[1].GpioData[0].11**
GPI29	Input 29	**M29**	Y:$78804,4	**PowerBrick[1].GpioData[0].12**
GPI30	Input 30	**M30**	Y:$78804,5	**PowerBrick[1].GpioData[0].13**
GPI31	Input 31	**M31**	Y:$78804,6	**PowerBrick[1].GpioData[0].14**
GPI32	Input 32	**M32**	Y:$78804,7	**PowerBrick[1].GpioData[0].15**
GPO9	Output 09	**M41**	Y:$78805,0	**PowerBrick[1].GpioData[0].16**
GPO10	Output 10	**M42**	Y:$78805,1	**PowerBrick[1].GpioData[0].17**
GPO11	Output 11	**M43**	Y:$78805,2	**PowerBrick[1].GpioData[0].18**
GPO12	Output 12	**M44**	Y:$78805,3	**PowerBrick[1].GpioData[0].19**
GPO13	Output 13	**M45**	Y:$78805,4	**PowerBrick[1].GpioData[0].20**
GPO14	Output 14	**M46**	Y:$78805,5	**PowerBrick[1].GpioData[0].21**
GPO15	Output 15	**M47**	Y:$78805,6	**PowerBrick[1].GpioData[0].22**
GPO16	Output 16	**M48**	Y:$78805,7	**PowerBrick[1].GpioData[0].23**

The I/O circuitry on this port is fixed in both Turbo Brick and Power Brick. The input pins can only be used as inputs, and the output pins can only be used as outputs. The selection of sinking or sourcing for both inputs and outputs is made by wiring choices.

ASIC I/O Point Setup

Turbo Brick Setup

All aspects of the digital I/O control in the Turbo Brick are fixed: use, direction, and inversion. There is nothing to set up, and nothing can be changed from the default.

Power Brick Setup

In Power Brick, two saved setup elements must be configured properly to use the general-purpose I/O. The factory default values for these elements set up the port to work properly; these should not be changed from the default.

(Note: These setup elements are write-protected, so **Sys.WpKey** must be set to $AAAAAAAA to permit you to change their values.)

Each bit of **PowerBrick[1].GpioDir[0]** specifies the direction of the matching bit of **PowerBrick[1].GpioData[0]**. If the direction bit is 0, the I/O point is set up as an input. If the direction bit is 1, the I/O point is set up as an output. To use the I/O port on the Power Brick, **PowerBrick[1].GpioDir[0]** *must* be set to the default of $00FF0000.

Each bit of **PowerBrick[1].GpioPol[0]** specifies the inversion polarity of the matching bit of **Clipper[1].GpioData[0]**. If the polarity bit is 0, the output point is set up as high-true (5V ⇔ 1) from the ASIC, which turns the output transistor on. The input point is set up as high-true (5V ⇔ 1) into the ASIC, which corresponds to a non-conducting input.

If the polarity bit is 1, the output point is set up as low-true (0V ⇔ 1) from the ASIC, which turns the output transistor off (most find this confusing). The input point is set up as low-true (0V ⇔ 1) into the ASIC, which corresponds to a non-conducting input.

To set up the port in the Power Brick to match the configuration of the Turbo Brick, set **PowerBrick[1].GpioPol[0]** to the default of $00000000, making all lines non-inverting.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M41->Y:$78805,0                // GPO9 signal
#define LeftClampOn     M41
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr LeftClampOn->PowerBrick[1].GpioData[0].16
```

Or it can be done with a simple text substitution:

```
#define LeftClampOn     PowerBrick[1].GpioData[0].16
```

UMAC General Purpose Digital I/O

UMAC systems can have boards for general-purpose digital I/O. These include the ACC-14E, ACC-65E, ACC-66E, ACC-67E, and ACC-68E. While each of these boards has its own hardware configuration for the inputs and/or outputs, all use the same I/O ASIC and have the same software mapping to the CPU.

All of these boards have 48 I/O points, organized in hardware into two 24-bit ports, with the Port A connectors on the bottom and the Port B connectors on the top. They are organized in software into six 8-bit registers, with the first three for Port A, and the second three for Port B.

UMAC I/O Board Port A

The following table shows the Turbo PMAC and Power PMAC addressing for the Port A I/O points on the board at the first I/O address (all SW1 switches ON).

Board I/O Point #	Port Function	Suggested M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
1	I/O 01	**M7001**	Y:$78C00,0	**GateIo[0].DataReg[0].0**
2	I/O 02	**M7002**	Y:$78C00,1	**GateIo[0].DataReg[0].1**
3	I/O 03	**M7003**	Y:$78C00,2	**GateIo[0].DataReg[0].2**
4	I/O 04	**M7004**	Y:$78C00,3	**GateIo[0].DataReg[0].3**
5	I/O 05	**M7005**	Y:$78C00,4	**GateIo[0].DataReg[0].4**
6	I/O 06	**M7006**	Y:$78C00,5	**GateIo[0].DataReg[0].5**
7	I/O 07	**M7007**	Y:$78C00,6	**GateIo[0].DataReg[0].6**
8	I/O 08	**M7008**	Y:$78C00,7	**GateIo[0].DataReg[0].7**
9	I/O 09	**M7009**	Y:$78C01,0	**GateIo[0].DataReg[1].0**
10	I/O 10	**M7010**	Y:$78C01,1	**GateIo[0].DataReg[1].1**
11	I/O 11	**M7011**	Y:$78C01,2	**GateIo[0].DataReg[1].2**
12	I/O 12	**M7012**	Y:$78C01,3	**GateIo[0].DataReg[1].3**
13	I/O 13	**M7013**	Y:$78C01,4	**GateIo[0].DataReg[1].4**
14	I/O 14	**M7014**	Y:$78C01,5	**GateIo[0].DataReg[1].5**
15	I/O 15	**M7015**	Y:$78C01,6	**GateIo[0].DataReg[1].6**
16	I/O 16	**M7016**	Y:$78C01,7	**GateIo[0].DataReg[1].7**
17	I/O 17	**M7017**	Y:$78C02,0	**GateIo[0].DataReg[2].0**
18	I/O 18	**M7018**	Y:$78C02,1	**GateIo[0].DataReg[2].1**
19	I/O 19	**M7019**	Y:$78C02,2	**GateIo[0].DataReg[2].2**
20	I/O 20	**M7020**	Y:$78C02,3	**GateIo[0].DataReg[2].3**
21	I/O 21	**M7021**	Y:$78C02,4	**GateIo[0].DataReg[2].4**
22	I/O 22	**M7022**	Y:$78C02,5	**GateIo[0].DataReg[2].5**
23	I/O 23	**M7023**	Y:$78C02,6	**GateIo[0].DataReg[2].6**
24	I/O 24	**M7024**	Y:$78C02,7	**GateIo[0].DataReg[2].7**

This table shows the "generic" Power PMAC data structure name **GateIo[0]**. The board-specific name (e.g. **Acc65E[0]**) could also be used, and Power PMAC will report back the definition using the board-specific name.

More generally, if the SW1 switches are configured to provide a Turbo PMAC base address of $7*xy*00, where *x* is a hex digit that can take the values 8, 9, A (10), or B (11), and *y* is a hex digit that can take the values C (12), D (13), or E (14), then the index ***i*** of **GateIo[*i*]** can be computed as:

$$i = 4 * (x - 8) + (y - 12)$$

UMAC I/O Board Port B

The following table shows the Turbo PMAC and Power PMAC addressing for the Port B I/O points on the board at the first I/O address (all SW1 switches ON).

Board I/O Point #	Port Function	Suggested M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
25	I/O 01	**M7025**	Y:$78C03,0	**GateIo[0].DataReg[3].0**
26	I/O 02	**M7026**	Y:$78C03,1	**GateIo[0].DataReg[3].1**
27	I/O 03	**M7027**	Y:$78C03,2	**GateIo[0].DataReg[3].2**
28	I/O 04	**M7028**	Y:$78C03,3	**GateIo[0].DataReg[3].3**
29	I/O 05	**M7029**	Y:$78C03,4	**GateIo[0].DataReg[3].4**
30	I/O 06	**M7030**	Y:$78C03,5	**GateIo[0].DataReg[3].5**
31	I/O 07	**M7031**	Y:$78C03,6	**GateIo[0].DataReg[3].6**
32	I/O 08	**M7032**	Y:$78C03,7	**GateIo[0].DataReg[3].7**
33	I/O 09	**M7033**	Y:$78C04,0	**GateIo[0].DataReg[4].0**
34	I/O 10	**M7034**	Y:$78C04,1	**GateIo[0].DataReg[4].1**
35	I/O 11	**M7035**	Y:$78C04,2	**GateIo[0].DataReg[4].2**
36	I/O 12	**M7036**	Y:$78C04,3	**GateIo[0].DataReg[4].3**
37	I/O 13	**M7037**	Y:$78C04,4	**GateIo[0].DataReg[4].4**
38	I/O 14	**M7038**	Y:$78C04,5	**GateIo[0].DataReg[4].5**
39	I/O 15	**M7039**	Y:$78C04,6	**GateIo[0].DataReg[4].6**
40	I/O 16	**M7040**	Y:$78C04,7	**GateIo[0].DataReg[4].7**
41	I/O 17	**M7041**	Y:$78C05,0	**GateIo[0].DataReg[5].0**
42	I/O 18	**M7042**	Y:$78C05,1	**GateIo[0].DataReg[5].1**
43	I/O 19	**M7043**	Y:$78C05,2	**GateIo[0].DataReg[5].2**
44	I/O 20	**M7044**	Y:$78C05,3	**GateIo[0].DataReg[5].3**
45	I/O 21	**M7045**	Y:$78C05,4	**GateIo[0].DataReg[5].4**
46	I/O 22	**M7046**	Y:$78C05,5	**GateIo[0].DataReg[5].5**
47	I/O 23	**M7047**	Y:$78C05,6	**GateIo[0].DataReg[5].6**
48	I/O 24	**M7048**	Y:$78C05,7	**GateIo[0].DataReg[5].7**

This table shows the "generic" Power PMAC data structure name **GateIo[0]**. The board-specific name (e.g. **Acc65E[0]**) could also be used, and Power PMAC will report back the definition using the board-specific name.

More generally, if the SW1 switches are configured to provide a Turbo PMAC base address of $7*xy*00, where *x* is a hex digit that can take the values 8, 9, A (10), or B (11), and *y* is a hex digit that can take the values C (12), D (13), or E (14), then the index ***i*** of **GateIo[*i*]** can be computed as:

$$i = 4 * (x - 8) + (y - 12)$$

Configuring the Power UMAC I/O Board

At re-initialization, the Power UMAC auto-identifies all of the I/O boards and sets up the control registers to use them properly. These configuration values will be saved to flash memory and then used every subsequent power-on/reset. Very few users will need to override these default settings.

The ACC-14E board is configured by default for all inputs. If a different configuration is desired, consult the manual for instructions in custom configuration, including directions and special latching of inputs.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M7001->Y:$78C00,0              // I/O01 signal
#define PartInserted    M7001
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr PartInserted->GateIo[0].DataReg[0].0
```

Or it can be done with a simple text substitution:

```
#define PartInserted    GateIo[0].DataReg[0].0
```

Converting Turbo UMAC Digital I/O to CK3E Digital I/O

For those users converting from the Turbo UMAC to the CK3E configuration of the Power PMAC, the change in digital I/O use is a little different. Each CK3W-AX*nnnn* axis interface module in a CK3M rack comes with 16 digital input points and 16 digital output points. Also, each CK3W-MD*xxxx* dedicated digital-I/O module comes with 16 digital input points and 16 digital output points.

These I/O points on the CK3W modules map into the DSPGATE3 ASIC that use the **Gate3[*i*]** data structure (or its **CK3WAX[*i*]** or **CK3WMD[*i*]** alias), not the IOGATE ASIC of the UMAC I/O boards. The following table shows the Power PMAC element name for each I/O point in a module.

Input Point	Power PMAC Element	Output Point	Power PMAC Element
IN00	**CK3W*nn*[*i*].GpioData[0].0**	OUT00	**CK3W*nn*[*i*].GpioData[0].16**
IN01	**CK3W*nn*[*i*].GpioData[0].1**	OUT01	**CK3W*nn*[*i*].GpioData[0].17**
IN02	**CK3W*nn*[*i*].GpioData[0].2**	OUT02	**CK3W*nn*[*i*].GpioData[0].18**
IN03	**CK3W*nn*[*i*].GpioData[0].3**	OUT03	**CK3W*nn*[*i*].GpioData[0].19**
IN04	**CK3W*nn*[*i*].GpioData[0].4**	OUT04	**CK3W*nn*[*i*].GpioData[0].20**
IN05	**CK3W*nn*[*i*].GpioData[0].5**	OUT05	**CK3W*nn*[*i*].GpioData[0].21**
IN06	**CK3W*nn*[*i*].GpioData[0].6**	OUT06	**CK3W*nn*[*i*].GpioData[0].22**
IN07	**CK3W*nn*[*i*].GpioData[0].7**	OUT07	**CK3W*nn*[*i*].GpioData[0].23**
IN08	**CK3W*nn*[*i*].GpioData[0].8**	OUT08	**CK3W*nn*[*i*].GpioData[0].24**
IN09	**CK3W*nn*[*i*].GpioData[0].9**	OUT09	**CK3W*nn*[*i*].GpioData[0].25**
IN10	**CK3W*nn*[*i*].GpioData[0].10**	OUT10	**CK3W*nn*[*i*].GpioData[0].26**
IN11	**CK3W*nn*[*i*].GpioData[0].11**	OUT11	**CK3W*nn*[*i*].GpioData[0].27**
IN12	**CK3W*nn*[*i*].GpioData[0].12**	OUT12	**CK3W*nn*[*i*].GpioData[0].28**
IN13	**CK3W*nn*[*i*].GpioData[0].13**	OUT13	**CK3W*nn*[*i*].GpioData[0].29**
IN14	**CK3W*nn*[*i*].GpioData[0].14**	OUT14	**CK3W*nn*[*i*].GpioData[0].30**
IN15	**CK3W*nn*[*i*].GpioData[0].15**	OUT15	**CK3W*nn*[*i*].GpioData[0].31**

Notes:

1. "*nn*" is "AX" for a CK3W-AX*xxxx* module, or "MD" for a CK3W-MD*xxxx* module
2. The index "*i*" for the module is set by the rotary switch on the front of the module.

Assigning User Names to I/O Points

In Turbo PMAC, to assign a user variable name to an I/O point, a two-step process is used. First, an M-variable is assigned to the point. Then a text substitution for the M-variable is defined. For example:

```
M7001->Y:$78C00,0              // I/O01 signal
#define PartInserted    M7001
```

In Power PMAC, it is a one-step process that can be done in either of two ways. In a matching example, it can be done with a declared pointer variable:

```
ptr PartInserted->CK3WAX[0].GpioData[0].0
```

Or it can be done with a simple text substitution:

```
#define PartInserted    CK3WAX[0].GpioData[0].0
```

UMAC ACC-5E Digital I/O

The ACC-5E has a 16-point digital I/O port (JTHW), direction-selectable by byte, for direct I/O or with multiplexed I/O from the ACC-34*x* family. It also has a 32-point digital I/O port (JIO), direction-selectable by byte, for direct I/O.

ACC-5E JTHW Port

The following table shows the I/O points for this port, and how they can be accessed in Turbo UMAC and Power UMAC. The I/O points in the Power UMAC can also be defined using the "generic" element **Gate2[0].MuxData** (where it is documented in the Software Reference Manual), but the definitions will report back using the **Acc5E[0]** alias for the element.

Pin(s)	Suggested M-Variable	Turbo PMAC Address Definition	Power PMAC Element Definition
SEL0	**M40**	Y:$78402,8	**Acc5E[0].MuxData.8**
SEL1	**M41**	Y:$78402,9	**Acc5E[0].MuxData.9**
SEL2	**M42**	Y:$78402,10	**Acc5E[0].MuxData.10**
SEL3	**M43**	Y:$78402,11	**Acc5E[0].MuxData.11**
SEL4	**M44**	Y:$78402,12	**Acc5E[0].MuxData.12**
SEL5	**M45**	Y:$78402,13	**Acc5E[0].MuxData.13**
SEL6	**M46**	Y:$78402,14	**Acc5E[0].MuxData.14**
SEL7	**M47**	Y:$78402,15	**Acc5E[0].MuxData.15**
SEL0-7	**M48**	Y:$78402,8,8	**Acc5E[0].MuxData.8.8**
DAT0	**M50**	Y:$78402,0	**Acc5E[0].MuxData.0**
DAT1	**M51**	Y:$78402,1	**Acc5E[0].MuxData.1**
DAT2	**M52**	Y:$78402,2	**Acc5E[0].MuxData.2**
DAT3	**M53s**	Y:$78402,3	**Acc5E[0].MuxData.3**
DAT4	**M54**	Y:$78402,4	**Acc5E[0].MuxData.4**
DAT5	**M55**	Y:$78402,5	**Acc5E[0].MuxData.5**
DAT6	**M56**	Y:$78402,6	**Acc5E[0].MuxData.6**
DAT7	**M57**	Y:$78402,7	**Acc5E[0].MuxData.7**
DAT0-7	**M58**	Y:$78402,0,8	**Acc5E[0].MuxData.0.8**

I/O Point Use

Set saved setup element **Acc5E[0].MuxMode** to $FFFF to use all 16 of these pins for general-purpose I/O.

I/O Point Directions

Set saved setup element **Acc5E[0].MuxDir** equal to the value of X:$78402 for the same direction control for the ASIC I/O points. To use SEL0 – SEL7 as outputs and DAT0 – DAT7 as inputs, this value is $FF00.

Buffer Directions

The direction of the byte-wide buffers are set this card by non-saved elements **Cid[4].PartData[*k*]**. Set bit 3 (value 8) of **Cid[4].PartData[0]** to 0 to configure the buffer for the DAT0-7 lines as inputs. Set bit 0 (value 1) of **Cid[4].PartData[1]** to 1 to configure the buffer for the SEL0-7 lines as outputs.

Multiplexed I/O Support

On a Power UMAC with an ACC-5E:

- **MuxIo.pOut** to **Acc5E[0].MuxData.a** and set **MuxIo.OutBit** to 8 to use the SEL0 – SEL7 lines on the ACC-5E3 JTHW port for multiplexed inputs.
- If **I27** is 0, set **MuxIo.pIn** to **Acc5E[0].MuxData.a** and set **MuxIo.InBit** to 0 to use the DAT0 line on the ACC-5E3 JTHW port for multiplexed inputs.
- If **I27** is 1, set **MuxIo.pIn** to **Acc5E[0].MuxData.a** and set **MuxIo.InBit** to 7 to use the DAT7 line on the ACC-5E3 JTHW port for multiplexed inputs. (This requires a changed jumper setting on the accessory board.)
- Set **MuxIo.Enable** to 1 to permit multiplexed data transfers over this port.
- Set **MuxIo.ClockPeriod** to the default value of 0 if you want the serial data transfers over this port to go at the fastest possible frequency. This could lead to bit rates of over 1 MHz, which may only be supported for 1 or 2 accessories at short distances.
- Set **MuxIo.ClockPeriod** to a non-zero value to specify the minimum period in microseconds for each bit to be transferred. Larger values may be needed for long distances and noisy environments.

Each ACC-34*x* board on the port has an index ***n*** (***n*** = 0 to 31) determined by the DIP switch settings on the board, with ***n*** = *{Turbo Base Address}* / 8.

- To enable Port A on board ***n*** for inputs, set **MuxIo.PortA[*n*].Enable** to 1 and **MuxIo.PortA[*n*].Dir** to 0.
- To enable Port B on board ***n*** for outputs, set **MuxIo.PortB[*n*].Enable** to 1 and **MuxIo.PortB[*n*].Dir** to 1.

The multiplexed data transfers over this port can either be done on an "on-demand" basis as in Turbo PMAC, or at a regular frequency with automatic transfers at that frequency.

- Set **MuxIo.UpdatePeriod** to the default value of 0 if you want to keep the "on-demand" nature of multiplexed data transfers of Turbo PMAC. In this case, set **MuxIo.PortA[*n*].Enable** or **MuxIo.PortB[*n*].Enable** to 1 each time you want to start a data transfer with that accessory port. Power PMAC will automatically reset the port's **Enable** element to 0 when this transfer is complete.
- Set **MuxIo.UpdatePeriod** to a non-zero value if you want automatic regular data transfers with each enabled accessory port. The value specifies the period in milliseconds. While this mechanism is different from Turbo PMAC, it may allow the application software to be more like Turbo PMAC, as this software simply accesses the holding registers in memory in the same manner as it would the TWS-format M-variable in Turbo PMAC.

To access data for multiplexed I/O transfers, the application will use 32-bit unsigned integer element **MuxIo.PortA[*n*].Data** for an accessory input port, and 32-bit unsigned integer element **MuxIo.PortB[*n*].Data** for an accessory output port.

If **MuxIo.UpdatePeriod** is 0 for on-demand transfers:

- Set **MuxIo.PortA[*n*].Enable** to 1 when you want to read an input port. Wait for this **Enable** element to be set to 0, then read the values in the bits of **MuxIo.PortA[*n*].Data** to access the port's input data.

- Write the values you want into the bits of **MuxIo.PortB[*n*].Data**, then set **MuxIo.PortB[*n*].Enable** to 1 to start the transfer to the accessory's output port.

If **MuxIo.UpdatePeriod** is greater than 0 for automatic repeated transfers:

- Simply read the values in the bits of **MuxIo.PortA[*n*].Data** to access the port's input data from the most recent cycle's transfer.

- Simply write the values you want into the bits of **MuxIo.PortB[*n*].Data**. These will then automatically be transferred to the accessory's output port on the next cycle.

Motor Compensation Tables

Turbo PMAC has four separate structures for motor compensation tables: 1D position compensation tables, 2D position compensation tables, backlash compensation tables, and torque compensation tables, each differently defined. It is possible to have 32 position compensation tables, 32 backlash compensation tables, and 32 torque compensation tables in a single Turbo PMAC.

Power PMAC has one common structure for motor compensation tables, adaptable to each of these functions by parameterization. It is possible to have 256 total compensation tables in a single Power PMAC.

The suggested settings in this note assume that Power PMAC motors are scaled in units of feedback counts or LSBs, as Turbo PMAC motors must be. Other scaling is possible.

Entering Position Compensation Tables (1D)

The most common compensation tables are one-dimensional (1D) position compensation tables, often used as "leadscrew compensation" tables.

Defining Table Structure

Before the individual points of the table can be entered, the structure of the table must be defined.

Turbo PMAC Definition

In Turbo PMAC, the one-dimensional position-compensation table structure is defined through a motor-specific on-line command of the form:

```
DEFINE COMP n, #x[D], #y, z
```

The table is assigned to the addressed motor for "bookkeeping" purposes, although it does not need to use that motor as a source or target motor. Each motor can have only one position compensation table assigned to it.

In this command:

- ***n*** is the number of zones in the table;
- ***x*** is the number of the source motor (defaults to addressed motor if not specified);
- ***y*** is the number of the target motor (defaults to addressed motor if not specified);
- ***z*** is the span of the table in counts of the source motor;
- If the letter **D** is used after the source motor number, the table correction is a function on the desired position of the source motor; if it is not used, the table correction is based on the actual position of the source motor.

Power PMAC Definition

In Power PMAC, the one-dimensional position-compensation table structure is defined by setting the values of saved setup data structure elements. Each compensation table (of any type) in Power PMAC must have a unique table index value ***m***.

To specify an equivalent definition to the above Turbo PMAC command, the following settings would be used:

- **CompTable[*m*].Nx[0]** = ***n*** // *n* zones in the first dimension
- **CompTable[*m*].Nx[1]** = 0 // No second dimension
- **CompTable[*m*].Nx[2]** = 0 // No third dimension
- **CompTable[*m*].Source[0]** = ***x*** // Source motor *x*

If the source motor was declared with a "D":

- **CompTable[*m*].SourceCtrl** = 0 // For desired position source

If the source motor was declared without a "D":

- **CompTable[*m*].SourceCtrl** = 1 // For actual position source

To apply the correction to the target motor net desired position (the only option in Turbo PMAC):

- **CompTable[*m*].Target[0]** = **Motor[*y*].CompDesPos.a**
- **CompTable[*m*].Sf[0]** = 1/16 // If keeping table entries same magnitude

(The Turbo PMAC position compensation tables write to a target register with units of 1/16 of a count. The Power PMAC position compensation tables write to a target register with units of whole counts. To keep the magnitude of the entries the same, the output scale factor of the table could be set to 1/16. Alternatively, the table entries themselves could be rescaled by the same factor.)

To apply the correction to the target motor inner and outer loop net actual positions (which provides smoother operation when the table is correcting for measurement errors):

- **CompTable[*m*].Target[0]** = **Motor[*y*].CompPos.a**
- **CompTable[*m*].Sf[0]** = -1/16 // If keeping table entries same magnitude
- **CompTable[*m*].Target[1]** = **Motor[*y*].CompPos2.a**
- **CompTable[*m*].Sf[1]** = -1/16 // If keeping table entries same magnitude

(When the same corrections are added to actual position registers instead of desired position registers, the correction direction is the opposite. The easiest way to compensate for this is to

make the scale factor negative. Alternatively, the signs of all of the table entries could be changed.)

- **CompTable[*m*].X0[0]** = 0 // Turbo tables always start at position zero of source
- **CompTable[*m*].Dx[0]** = ***z*** // Span of table in counts of source motor

To specify interpolation and rollover like Turbo PMAC tables:

- **CompTable[*m*].Ctrl** = $0 // 1st-order interpolation, rollover enabled

To enable smoother cubic interpolation (which should not hurt backward compatibility):

- **CompTable[*m*].Ctrl** = $3 // 3rd-order interpolation, rollover enabled (default)

To enable cubic interpolation and disable rollover (maintaining the last correction):

- **CompTable[*m*].Ctrl** = $57 // 3rd-order interpolation, rollover disabled

Entering Table Points

After the table structure is defined, the actual points in the table can be entered.

Turbo PMAC Point Entry

After the command to define the table structure, the next ***n*** constants entered are used as table data points. These constants can be separated by spaces, commas, or carriage returns. The entry will be of the form:

d1, d2, d3, …, dn

where ***di*** is the "***i***th" data point, the correction at (***i * z / n***) counts of the source motor.

If **I30** is 0, the final point ***dn*** must be 0 for smooth rollover of the table action beyond the defined span, because the correction at source position 0 is always 0.

If **I30** is 1, the final point ***dn*** is used at both ends of the table, making it possible to have a non-zero correction at the zero point, and guaranteeing smooth rollover of the table action.

Power PMAC Point Entry

In Power PMAC, the table data points must be directly assigned to data structure elements **CompTable[*m*].Data[*i*]**. This can be done one table point per command – for example:

CompTable[*m*].Data[2] = ***d2***

However, multiple table points can be entered into consecutive elements in a single command, with the values separated by commas – for example:

CompTable[*m*].Data[0] = ***d0***, ***d1***, ***d2***, ***d3***, …, ***dn***

In Power PMAC, the correction at the start of the table (***d0*** here) must be explicitly entered, even if it is 0. There is no automatic wraparound function comparable to the **I30** = 1 mode of Turbo PMAC to copy the last value into the first.

The points can be entered on multiple lines. For example, if the first command line entered the data points ***d0*** through ***d9***, the next command line could enter points ***d10*** through ***d19*** as follows:

CompTable[*m*].Data[10] = ***d10***, ***d11***, ***d12***, ***d13***, …, ***d19***

Entering Position Compensation Tables (2D)

Two-dimensional (planar) compensation tables are commonly used with Cartesian tables in high-precision applications.

Defining Table Structure

Before the individual points of the table can be entered, the structure of the table must be defined.

Turbo PMAC Definition

In Turbo PMAC, the two-dimensional position-compensation table structure is defined through a motor-specific on-line command of the form:

DEFINE COMP *r.c, #R[D], #C[D], #y, RS, CS*

The table is assigned to the addressed motor for "bookkeeping" purposes, although it does not need to use that motor as a source or target motor. Each motor can have only one position compensation table assigned to it.

In this command:

- ***r*** is the number of rows in the table;
- ***c*** is the number of columns in the table;
- ***R*** is the number of the row source motor;
- ***C*** is the number of the column source motor (defaults to addressed motor if not specified);
- ***y*** is the number of the target motor (defaults to addressed motor if not specified);
- ***RS*** is the span of the table in counts of the row source motor;
- ***CS*** is the span of the table in counts of the column source motor;
- If the letter **D** is used after a source motor number, the table correction is a function on the desired position of this source motor; if it is not used, the table correction is based on the actual position of this source motor.

Power PMAC Definition

In Power PMAC, the two-dimensional position-compensation table structure is defined by setting the values of saved setup data structure elements. To specify an equivalent definition to the above Turbo PMAC command, the following settings would be used:

- **CompTable[*m*].Nx[0]** = ***r*** // *r* zones in the first dimension
- **CompTable[*m*].Nx[1]** = ***c*** // *c* zones in the second dimension
- **CompTable[*m*].Nx[2]** = 0 // No third dimension
- **CompTable[*m*].Source[0]** = ***R*** // Row source motor *R*
- **CompTable[*m*].Source[1]** = ***C*** // Column source motor *C*

If the source motors were declared with a "**D**":

- **CompTable[*m*].SourceCtrl** = 0 // For desired position sources

If the source motors were declared without a "**D**":

- **CompTable[*m*].SourceCtrl** = 3 // For actual position sources

To apply the correction to the target motor net desired position (the only option in Turbo PMAC):

- **CompTable[*m*].Target[0]** = **Motor[*y*].CompDesPos.a**
- **CompTable[*m*].Sf[0]** = 1/16 // If keeping table entries same magnitude

(The Turbo PMAC position compensation tables write to a target register with units of 1/16 of a count. The Power PMAC position compensation tables write to a target register with units of whole counts. To keep the magnitude of the entries the same, the output scale factor of the table could be set to 1/16. Alternatively, the table entries themselves could be rescaled by the same factor.)

To apply the correction to the target motor inner and outer loop net actual positions (which provides smoother operation when the table is correcting for measurement errors):

- **CompTable[*m*].Target[0]** = **Motor[*y*].CompPos.a**
- **CompTable[*m*].Sf[0]** = -1/16 // If keeping table entries same magnitude
- **CompTable[*m*].Target[1]** = **Motor[*y*].CompPos2.a**
- **CompTable[*m*].Sf[1]** = -1/16 // If keeping table entries same magnitude

(When the same corrections are added to actual position registers instead of desired position registers, the correction direction is the opposite. The easiest way to compensate for this is to make the scale factor negative. Alternatively, the signs of all of the table entries could be changed.)

- **CompTable[*m*].X0[0]** = 0 // Turbo tables always start at position zero of source
- **CompTable[*m*].Dx[0]** = ***RS*** // Span of table in counts of row source motor
- **CompTable[*m*].X0[1]** = 0 // Turbo tables always start at position zero of source
- **CompTable[*m*].Dx[1]** = ***CS*** // Span of table in counts of column source motor

To specify interpolation and rollover like Turbo PMAC tables:

- **CompTable[*m*].Ctrl** = $0 // 1st-order interpolation, rollover enabled

To enable smoother cubic interpolation (which should not hurt backward compatibility):

- **CompTable[*m*].Ctrl** = $3 // 3rd-order interpolation, rollover enabled (default)

To enable cubic interpolation and disable rollover (maintaining the last correction):

- **CompTable[*m*].Ctrl** = $57 // 3rd-order interpolation, rollover disabled

Entering Table Points

After the table structure is defined, the actual points in the table can be entered.

Turbo PMAC Point Entry

After the command to define the table structure, the next [(***r***+1)*(***c***+1)-1] constants entered are used as table data points. These constants can be separated by spaces, commas, or carriage returns. The entry will be of the form:

```
      x1y0, x2y0, …, xcy0
x0y1, x1y1, x2y1, …, xcy1
x0y2, x1y2, x2y2, …, xcy2
…
x0yr, x1yr, x2yr, …, xcyr
```

where ***xiyj*** is the data point for the "***i***th" row and "***j***th" column (which do not need to represent X and Y axes), the correction at (***i * RS / c***) counts of the first source motor, at (***j * CS / r***) of the second source motor.

Power PMAC Point Entry

In Power PMAC, the table data points must be directly assigned to data structure elements **CompTable[*m*].Data[*j*][*i*]**. This can be done one table point per command – for example:

CompTable[*m*].Data[5][2] = ***x2y5***

However, multiple table points can be entered into consecutive row elements (with the *final* index incrementing in a single command), with the values separated by commas – for example:

CompTable[*m*].Data[0][0] = ***x0y0***, ***x1y0***, ***x2y0***, …, ***xcy0***

Note the different ordering of the data-structure indices from the way points are usually referenced. The point value ***x1y0*** is assigned to **CompTable[*m*].[0][1]**.

In Power PMAC, the correction at the start of the table (***x0y0*** here) must be explicitly entered, even if it is 0. There is no automatic wraparound function comparable to the **I30** = 1 mode of Turbo PMAC to copy the last value into the first.

The points can be entered on multiple lines, and separate rows of the table (with different first index values) should be entered on separate lines. For example, if the first command line entered the data points ***d0*** through ***d9***, the next command line could enter points ***d10*** through ***d19*** as follows:

CompTable[*m*].Data[1][0] = ***x0y1***, ***x1y1***, ***x2y1***, …, ***xcy1***

Entering Backlash Compensation Tables

Backlash compensation tables are most often used in combination with 1D position compensation tables to create position compensation that is different in the two directions of motion.

Defining Table Structure

Before the individual points of the table can be entered, the structure of the table must be defined.

Turbo PMAC Definition

In Turbo PMAC, the backlash-compensation table structure is defined through a motor-specific on-line command of the form:

```
#x DEFINE BLCOMP n, z
```

The table always uses the position of the addressed motor as its source data, and the backlash compensation register of the addressed motor as its target register. Backlash compensation tables in Turbo PMAC are always 1D.

In this command:

- ***n*** is the number of zones in the table;
- ***z*** is the span of the table in counts of the source motor;

Power PMAC Definition

In Power PMAC, the backlash-compensation table structure is defined by setting the values of saved setup data structure elements. To specify an equivalent definition to the above Turbo PMAC command, the following settings would be used:

- **CompTable[*m*].Nx[0]** = ***n*** // *n* zones in the first dimension
- **CompTable[*m*].Nx[1]** = 0 // No second dimension
- **CompTable[*m*].Nx[2]** = 0 // No third dimension

- **CompTable[*m*].Source[0]** = ***x*** // Source motor *x*
- **CompTable[*m*].SourceCtrl** = 1 // for actual position source

To apply the correction to the backlash compensation register:

- **CompTable[*m*].Target[0]** = **Motor[*x*].BlCompSize.a**
- **CompTable[*m*].Sf[0]** = 1/16 // If keeping table entries same magnitude

(The Turbo PMAC backlash compensation tables write to a target register with units of 1/16 of a count. The Power PMAC backlash compensation tables write to a target register with units of whole counts. To keep the magnitude of the entries the same, the output scale factor of the table could be set to 1/16. Alternatively, the table entries themselves could be rescaled by the same factor.)

- **CompTable[*m*].X0[0]** = 0 // Turbo tables always start at position zero of source
- **CompTable[*m*].Dx[0]** = ***z*** // Span of table in counts of source motor

To specify interpolation and rollover like Turbo PMAC tables:

- **CompTable[*m*].Ctrl** = $0 // 1st-order interpolation, rollover enabled

To enable smoother cubic interpolation (which should not hurt backward compatibility):

- **CompTable[*m*].Ctrl** = $3 // 3rd-order interpolation, rollover enabled

To enable cubic interpolation and disable rollover (maintaining the last correction):

- **CompTable[*m*].Ctrl** = $57 // 3rd-order interpolation, rollover disabled

Entering Table Points

After the table structure is defined, the actual points in the table can be entered.

Turbo PMAC Point Entry

After the command to define the table structure, the next ***n*** constants entered are used as table data points. These constants can be separated by spaces, commas, or carriage returns. The entry will be of the form:

d1, d2, d3, …, dn

where ***di*** is the "***i***th" data point, the correction at (***i * z / n***) counts of the source motor.

If **I30** is 0, the final point ***dn*** must be 0 for smooth rollover of the table action beyond the defined span, because the correction at source position 0 is always 0.

If **I30** is 1, the final point ***dn*** is used at both ends of the table, making it possible to have a non-zero correction at the zero point, and guaranteeing smooth rollover of the table action.

Power PMAC Point Entry

In Power PMAC, the table data points must be directly assigned to data structure elements **CompTable[*m*].Data[*i*]**. This can be done one table point per command – for example:

CompTable[*m*].Data[2] = ***d2***

However, multiple table points can be entered into consecutive elements in a single command, with the values separated by commas – for example:

CompTable[*m*].Data[0] = ***d0***, ***d1***, ***d2***, ***d3***, …, ***dn***

In Power PMAC, the correction at the start of the table (***d0*** here) must be explicitly entered, even if it is 0. There is no automatic wraparound function comparable to the **I30** = 1 mode of Turbo PMAC to copy the last value into the first.

The points can be entered on multiple lines. For example, if the first command line entered the data points ***d0*** through ***d9***, the next command line could enter points ***d10*** through ***d19*** as follows:

CompTable[*m*].Data[10] = ***d10***, ***d11***, ***d12***, ***d13***, …, ***d19***

Entering Torque Compensation Tables

Torque compensation tables are used to combat effects such as cogging (reluctance) torque in motors, providing torque offsets as a function of position.

Defining Table Structure

Before the individual points of the table can be entered, the structure of the table must be defined.

Turbo PMAC Definition

In Turbo PMAC, the torque-compensation table structure is defined through a motor-specific on-line command of the form:

#*x* DEFINE TCOMP *n*, *z*

The table always uses the position of the addressed motor as its source data, and the torque offset register of the addressed motor as its target register. Torque compensation tables in Turbo PMAC are always 1D.

In this command:

- ***n*** is the number of zones in the table;
- ***z*** is the span of the table in counts of the source motor;

Power PMAC Definition

In Power PMAC, the torque-compensation table structure is defined by setting the values of saved setup data structure elements. To specify an equivalent definition to the above Turbo PMAC command, the following settings would be used:

- **CompTable[*m*].Nx[0]** = ***n*** // *n* zones in the first dimension
- **CompTable[*m*].Nx[1]** = 0 // No second dimension
- **CompTable[*m*].Nx[2]** = 0 // No third dimension
- **CompTable[*m*].Source[0]** = ***x*** // Source motor *x*
- **CompTable[*m*].SourceCtrl** = 1 // For actual position source

To apply the correction to the torque offset register:

- **CompTable[*m*].Target[0]** = **Motor[*x*].CompDac.a**
- **CompTable[*m*].Sf[0]** = 1/256 // If keeping data points same magnitude

(The Turbo PMAC torque compensation tables treat the target register as a signed 24-bit value having a range of $+/-2^{23}$. The Power PMAC torque compensation tables treat the target register as a signed 16-bit value having a range of $+/-2^{15}$. To keep the magnitude of the entries the same, the output scale factor of the table could be set to $1 / 2^{8}$. Alternatively, the table entries themselves could be rescaled by the same factor.)

- **CompTable[*m*].X0[0]** = 0 // Turbo tables always start at position zero of source
- **CompTable[*m*].Dx[0]** = ***z*** // Span of table in counts of source motor

To specify interpolation and rollover like Turbo PMAC tables:

- **CompTable[*m*].Ctrl** = $0 // 1st-order interpolation, rollover enabled

To enable smoother cubic interpolation (which should not hurt backward compatibility):

- **CompTable[*m*].Ctrl** = $3 // 3rd-order interpolation, rollover enabled (default)

To enable cubic interpolation and disable rollover (maintaining the last correction):

- **CompTable[*m*].Ctrl** = $57 // 3rd-order interpolation, rollover disabled

Entering Table Points

After the table structure is defined, the actual points in the table can be entered.

Turbo PMAC Point Entry

After the command to define the table structure, the next ***n*** constants entered are used as table data points. These constants can be separated by spaces, commas, or carriage returns. The entry will be of the form:

```
d1, d2, d3, …, dn
```

where ***di*** is the "***i***th" data point, the correction at (***i * z / n***) counts of the source motor.

If **I30** is 0, the final point ***dn*** must be 0 for smooth rollover of the table action beyond the defined span, because the correction at source position 0 is always 0.

If **I30** is 1, the final point ***dn*** is used at both ends of the table, making it possible to have a non-zero correction at the zero point, and guaranteeing smooth rollover of the table action.

Power PMAC Point Entry

In Power PMAC, the table data points must be directly assigned to data structure elements **CompTable[*m*].Data[*i*]**. This can be done one table point per command – for example:

CompTable[*m*].Data[2] = ***d2***

However, multiple table points can be entered into consecutive elements in a single command, with the values separated by commas – for example:

CompTable[*m*].Data[0] = ***d0***, ***d1***, ***d2***, ***d3***, …, ***dn***

In Power PMAC, the correction at the start of the table (***d0*** here) must be explicitly entered, even if it is 0. There is no automatic wraparound function comparable to the **I30** = 1 mode of Turbo PMAC to copy the last value into the first.

The points can be entered on multiple lines. For example, if the first command line entered the data points ***d0*** through ***d9***, the next command line could enter points ***d10*** through ***d19*** as follows:

CompTable[*m*].Data[10] = ***d10***, ***d11***, ***d12***, ***d13***, …, ***d19***

Executing Compensation Tables

The methods for controlling the execution of compensation tables is somewhat different between Turbo PMAC and Power PMAC.

Turbo PMAC Enabling Control

In Turbo PMAC, the execution of compensation tables is controlled by the value of Boolean variable **I51**. If **I51** is 0, no compensation tables of any type are enabled. If **I51** is 1, all compensation tables of all types are enabled.

The value of **I51** can be saved to flash memory and automatically restored to active memory on power-on/reset, so the tables can be enabled immediately on power-on/reset.

Power PMAC Enabling Control

In Power PMAC, the execution of compensation tables is controlled by the value of integer variable **Sys.CompEnable**. The compensation tables with table index values ***m*** from 0 to (**Sys.CompEnable** – 1) are enabled.

The value of **Sys.CompEnable** *cannot* be saved to flash memory. On power-on/reset, **Sys.CompEnable** is automatically set to 0, disabling all tables. The application software must subsequently set it to a value greater than 0 to enable tables. Usually, the tables are not enabled until after position references have been established for all motors.

Coordinate System Axis Definitions

In both Turbo PMAC and Power PMAC, motors can be assigned to axes in coordinate systems so they can be commanded in motion programs. In both controller families, these assignments can be made either through axis definition statements or kinematic subroutines.

"Undefined" Motors

In both Turbo PMAC and Power PMAC, after re-initialization or a global **UNDEFINE ALL** command, axis definitions for all motors are cleared. After a coordinate-system-specific **UNDEFINE** command, axis definitions for all motors in that coordinate system are cleared.

In Turbo PMAC, an "undefined" motor does not belong to any coordinate system for fault-sharing purposes, but it will use Coordinate System 1's time base for motor moves such as jogs.

In Power PMAC, an "undefined" motor belongs to Coordinate System 0 for fault-sharing purposes and for time base in motor moves. Technically, it has the "null" definition (**#*x*->0**) in C.S. 0. It is not common to assign motors to an axis in C. S. 0, or to run motion programs in it.

In Power PMAC, it is possible to clear the axis definition of a single motor with the coordinate-system-specific command **#*x*->0**. This leaves it in the coordinate system for fault-sharing purposes, but it has no relationship to any axes. Once it has the null definition in a coordinate system, it can be assigned to another coordinate system.

Standard Axis Definitions

In both Turbo PMAC and Power PMAC, motors are assigned to axes through the use of on-line "axis definition" commands acting on the addressed coordinate system.

Axis definition commands used in Turbo PMAC can be used unchanged in Power PMAC. (Power PMAC also provides more flexibility in these commands.)

Power PMAC supports all 9 of the axis names (A, B, C, U, V, W, X, Y, Z) of Turbo PMAC. All definitions can be used with scaling and offset in both controller families (e.g. **#1->2000X-500**).

Power PMAC supports the combination of X, Y, and Z axes, or U, V, and W axes in a single axis definition command (e.g. **#1->1000X-1000Y**), just as Turbo PMAC does.

Power PMAC supports the definition of "inverse kinematic" axes through the definition **#*x*->I**, just as Turbo PMAC does. With this definition the motor positions are calculated from axis positions using the inverse kinematic subroutine for the coordinate system.

Kinematic Subroutines

Both Turbo PMAC and Power PMAC can accept forward and inverse-kinematic subroutines in each coordinate system to define relationships between motors and axes that are too complex for axis definition statements. While the concept is the same in both controller families, there are significant differences in implementation.

Turbo PMAC Kinematic Subroutine Variables

In Turbo PMAC kinematic subroutines, motor positions for each Motor *x* use global variable **P*x***. These are the inputs for forward-kinematic subroutines, and the outputs for inverse-kinematic subroutines.

In Turbo PMAC kinematic subroutines, axis positions for the A, B, C, U, V, W, X, Y, and Z-axes of the coordinate system use variables **Q1** – **Q9**, respectively, for the coordinate system. These are the outputs for forward-kinematic subroutines, and the inputs for inverse-kinematic subroutines.

Power PMAC Kinematic Subroutine Variables

In Power PMAC kinematic subroutines, motor positions for each Motor *x* use local variable **L*x*** for the coordinate system. These are the inputs for forward-kinematic subroutines, and the outputs for inverse-kinematic subroutines.

In Power PMAC kinematic subroutines, axis positions for the A, B, C, U, V, W, X, Y, and Z-axes of the coordinate system use local variables **C0** – **C8**, respectively, for the coordinate system. These are the outputs for forward-kinematic subroutines, and the inputs for inverse-kinematic subroutines.

The following example shows the difference in syntax for a very simple example in which the kinematic subroutines implement a coordinate system equivalent to [**#1->1000X #2->1000Y**] for a coordinate system. No defined or declared variable names are used in this example.

Turbo PMAC Syntax	Power PMAC Syntax

`OPEN FORWARD CLEAR`
`Q7 = P1 / 1000`
`Q8 = P2 / 1000`
`CLOSE`
 	
`OPEN INVERSE CLEAR`
`P1 = 1000 * Q7`
`P2 = 1000 * Q8`
`CLOSE`
	`open forward`
	`C6 = L1 / 1000;`
	`C7 = L2 / 1000;`
	`close`
	
	`open inverse`
	`L1 = 1000 * C6;`
	`L2 = 1000 * C7;`
	`close`

The Power PMAC IDE provides a set of pre-defined text substitutions for the motor and axis variables used. **KinPosMotor*x*** can be used for the **L*x*** motor position value, and **KinPosAxis*α*** can be used for **C*n*** axis position value (***α*** = A, B, C, U, V, W, X, Y, Z for **C0** – **C8**, respectively). The IDE will automatically substitute the variable name on downloading to the Power PMAC.

The above example now becomes:

Turbo PMAC Syntax	Power PMAC Syntax

`OPEN FORWARD CLEAR`
`Q7 = P1 / 1000`
`Q8 = P2 / 1000`
`CLOSE`
 	
`OPEN INVERSE CLEAR`
`P1 = 1000 * Q7`
`P2 = 1000 * Q8`
`CLOSE`
	`open forward`
	`KinPosAxisX = KinPosMotor1 / 1000;`
	`KinPosAxisY = KinPosMotor2 / 1000;`
	`close`
	
	`open inverse`
	`KinPosMotor1 = 1000 * KinPosAxisX;`
	`KinPosMotor2 = 1000 * KinPosAxisY;`
	`close`

Problem Detection and Reaction

Robust kinematic subroutines will detect problems in the coordinate system geometry and prevent or abort motion if valid solutions cannot be found. In forward-kinematic subroutines, the most important potential problem is usually whether all of the motors have been properly position-referenced or not. In inverse-kinematic subroutines, the most important potential problem is usually whether the commanded axis position is physically achievable or not.

Forward-Kinematic Problem Detection

In Turbo PMAC, the forward-kinematic subroutine typically checks the "home complete" status bits for each motor in the coordinate system (suggested variable **M*x*45**). All of these bits must be true to allow continuation of program execution.

In Power PMAC, the forward-kinematic subroutine can check the single coordinate-system "home complete" status bit **Coord[*x*].HomeComplete**, which Power PMAC has already computed as the logic AND of the individual motor status bits.

Alternatively, Power PMAC users can set saved setup element **Coord[*x*].HomeRequired** to 1. In this setting, Power PMAC will check that all of the motors are properly referenced before it even calls the forward-kinematic subroutine from a motion program, preventing execution if the referencing is not complete.

Inverse-Kinematic Problem Detection

The inverse-kinematic check for whether the commanded position can be physically achieved or not is done in fundamentally the same way in both Turbo PMAC and Power PMAC. It is specific to the mechanism geometry, and implemented with user math and logic.

Aborting on Problem Detection

In Turbo PMAC, the most common method of stopping motion program execution when a problem is detected in either the forward or inverse-kinematic subroutine is to set the coordinate system "run-time error" bit (suggested variable **M*sx*82**) to 1. This bit is usually set by Turbo PMAC firmware and treated by the user as a status bit, but it is possible to use it as a control bit here.

In Power PMAC, the most common method of stopping motion program execution when a problem is detected in either the forward or inverse-kinematic subroutine is to set the element **Coord[*x*].ErrorStatus** to 255. Several other values of this element are set automatically by

Power PMAC firmware on other problems, but the value of 255 is reserved for user-detected error conditions.

In either Turbo PMAC or Power PMAC, this setting effectively causes an "abort" command to be issued by the firmware. If the setting is made in the forward-kinematic subroutine at the beginning of motion program execution, it will prevent the motion program from starting. If the setting is made in the inverse-kinematic subroutine in the middle of program execution, program execution is halted, and all motors are decelerated to a stop according to their individual abort deceleration parameters.

Axis Position Reporting

Many users want the PMAC to be able to report the present position (and sometimes related quantities) in the axis (tool-tip) coordinates. This reporting requires that the forward-kinematic mathematical transformation from motor to axis coordinates be performed.

In Turbo PMAC, the forward-kinematic subroutine used for motion program execution cannot also be used for position reporting purposes. Turbo PMAC users who want this position reporting capability have written PLC programs that effectively duplicate the calculations of the forward-kinematic subroutine, but with different input and output variables.

In Power PMAC, the forward-kinematic subroutine itself can be used for position reporting purposes as well as for motion program execution. With the on-line **p** or buffered **pread** command, it uses the present motor actual position values as inputs. With the on-line **d** or buffered **dread** command, it uses the present motor actual position values as inputs.

It is also possible to use the forward-kinematic subroutine for axis velocity reporting (with the on-line **v** or buffered **vread** command) or for axis position-error reporting (with the on-line **f** or buffered **fread** command). However, to implement this functionality, it is necessary to use the subroutine in a "double-pass" mode. Refer to the Power PMAC User's Manual chapter *Setting Up Coordinate Systems* to see how to structure the subroutine to provide this capability.

Coordinate Transformation Matrices

Both Turbo PMAC and Power PMAC support coordinate-system axis-transformation matrices that permit the user to perform operations such as offsets, scaling, and rotations on axes in the coordinate system. The Power PMAC transformation capabilities are a superset of the Turbo PMAC capabilities, which are limited to the X, Y, and Z axes. This note covers only those Power PMAC transformation capabilities that match Turbo PMAC.

Reserving Memory for Transformation Matrices

In Turbo PMAC, the user must reserve memory for ***n*** transformation matrices with the **`define tbuf n`** command. The matrices created with this command are numbered 1 to ***n***.

In Power PMAC, memory is always reserved for 256 transformation matrices (**Tdata[0]** to **Tdata[255]**). It is not possible to reserve memory for any additional matrices.

Selecting the Active Transformation Matrix

In Turbo PMAC, the buffered motion program command **`TSELECTi`** (short form **`TSELi`**) is used to select the transformation matrix to be used in the motion program calculations. ***i*** must be a constant in the range 1 to ***n***.

To deselect all transformation matrices, the buffered motion-program command **`TSELECT0`** (short form **`TSEL0`**) is used.

In Power PMAC, the buffered motion program command **`tseli`** is used to select the transformation matrix to be used in the motion program calculations. ***i*** can be a constant, or a mathematical expression in parentheses; it must be in the range 0 to 255. Note that **`tselecti`** is not valid syntax in Power PMAC.

To deselect all transformation matrices, the buffered motion-program command **`tsel-1`** is used.

Initializing the Active Transformation Matrix

In Turbo PMAC, the selected transformation matrix is initialized to the identity matrix with the buffered motion-program command **`TINIT`**. There is no matrix number specified in the command; it always operates on the already-selected matrix.

In Power PMAC, the specified transformation matrix is initialized to the identity matrix with the matrix function **`tinit(i)`**, where ***i*** is the matrix number. It is the user's responsibility to match the specified matrix number with the selected matrix number.

This function provides a Boolean return value indicating the success of the operation. It can either be assigned to a user variable, as in:

```
MySuccess = tinit(5);
```

or it can be ignored with the "no-op" command, as in:

```
nop(tinit(1));
```

Absolute Specification of Transformation Matrix

Turbo PMAC Absolute Specification

In Turbo PMAC, the values of the selected transformation matrix must first be assigned to a set of consecutively numbered Q-variables for the coordinate system, then copied to the selected matrix with a buffered program command.

The values of the 3x3 "rotation" portion of the transformation matrix are set with motion-program code such as:

```
Q(n) = R11              Q(n+1) = R12            Q(n+2) = R13
Q(n+3) = R21            Q(n+4) = R22            Q(n+5) = R23
Q(n+6) = R31            Q(n+7) = R32            Q(n+8) = R33
AROTn
```

The **AROT*n*** command causes these 9 values to be copied into the rotation portion of the selected transformation matrix, overwriting the values that had been there.

The values of the 1x3 "displacement" portion of the transformation matrix are set with motion-program code such as:

```
Q(m) = D1
Q(m+1) = D2
Q(m+2) = D3
ADISm
```

The **ADIS*m*** command causes these 3 values to be copied into the displacement portion of the selected transformation matrix, overwriting the values that had been there.

Power PMAC Absolute Specification

In Power PMAC, the values of the specified transformation matrix are written directly to elements of the matrix, without any intermediate holding variables. It is the user's responsibility to match the specified matrix number with the selected matrix number.

The values of the 3x3 XYZ "rotation" portion of the transformation matrix are set directly with code such as:

```
Tdata[i].Diag[6] = R11;     Tdata[i].XYZ[0] = R12;      Tdata[i].XYZ[1] = R13;
Tdata[i].XYZ[2] = R21;      Tdata[i].Diag[7] = R22;     Tdata[i].XYZ[3] = R23;
Tdata[i].XYZ[4] = R31;      Tdata[i].XYZ[5] = R32;      Tdata[i].Diag[8] = R33;
```

The values of the 1x3 "displacement" portion of the transformation matrix are set directly with code such as:

```
Tdata[i].Bias[6] = D1;
Tdata[i].Bias[7] = D2;
Tdata[i].Bias[8] = D3;
```

Incremental Specification of Transformation Matrix

Turbo PMAC Incremental Specification

In Turbo PMAC, the values used to act on the selected transformation matrix must first be assigned to a set of consecutively number Q-variables for the coordinate system, then used to operate on the values already in the selected transformation matrix with a buffered program command.

The values to operate on the 3x3 "rotation" portion of the transformation matrix are set with motion-program code such as:

```
Q(n) = R11              Q(n+1) = R12            Q(n+2) = R13
Q(n+3) = R21            Q(n+4) = R22            Q(n+5) = R23
Q(n+6) = R31            Q(n+7) = R32            Q(n+8) = R33
IROTn
```

The **IROT*n*** command causes these 9 values to be used in a 3x3 matrix multiplication operation with the rotation portion of the selected transformation matrix, with the results overwriting the values that had been there.

The values to operate on the 1x3 "displacement" portion of the transformation matrix are set with code such as:

```
Q(m) = D1
Q(m+1) = D2
Q(m+2) = D3
IDISm
```

The **IDIS*m*** command causes these 3 values to be used in a 1x3 vector addition operation with the displacement portion of the selected transformation matrix, with the results overwriting the values that had been there.

Power PMAC Incremental Specification

In Power PMAC, the values used to act on the selected transformation matrix must first be assigned to a separate transformation matrix (often called an "operator" matrix), then used to operate on the values already in the selected transformation matrix with a special function.

The values to operate on the 3x3 "rotation" portion, and the 1x3 "displacement" portion, of the selected transformation code can be set with code such as:

```
Tdata[j].Diag[6] = R11;     Tdata[j].XYZ[0] = R12;      Tdata[j].XYZ[1] = R13;
Tdata[j].XYZ[2] = R21;      Tdata[j].Diag[7] = R22;     Tdata[j].XYZ[3] = R23;
Tdata[j].XYZ[4] = R31;      Tdata[j].XYZ[5] = R32;      Tdata[j].Diag[8] = R33;
Tdata[j].Bias[6] = D1;
Tdata[j].Bias[7] = D2;
Tdata[j].Bias[8] = D3;
```

The matrix index ***j*** for the operator matrix is different from the matrix index ***i*** for the selected matrix.

This operator transformation can then be "propagated" into the active selected transformation with the **tprop** transformation mathematical function. This function takes three arguments:

1. The index of the resulting transformation matrix
2. The index of the starting transformation matrix
3. The index of the operator transformation matrix

In this case, we are using the selected transformation matrix "***i***" as both the starting and resulting transformation matrix, and our "incremental" transformation matrix "***j***" as the operator matrix, so the function would be used as **tprop(*i*,*i*,*j*)**.

The **tprop** function provides a return value of the determinant of the resulting matrix. It can either be assigned to a user variable, as in:

```
TransformDet = tprop(i,i,j);
```

or it can be ignored with the "no-op" command, as in:

```
nop(tprop(i,i,j));
```

Note that this matrix propagation function in Power PMAC combines what were separate rotation and displacement functions in Turbo PMAC. Many Turbo PMAC transformations, such as scaling and rotating about an arbitrary center point, required consecutive **IDIS**, **IROT**, and **IDIS** incremental transformations. These can be implemented in Power PMAC with a single **tprop** function call.

Axis Programming Offsets

It is common for users to want to offset their axis frames of reference dynamically from the "base" machine origin. This is done differently in Turbo PMAC and Power PMAC.

Turbo PMAC Axis Offsets

In Turbo PMAC, any of the nine axes of a coordinate system can be offset using the buffered motion program **PSET*{axis}{data}*** command, or the equivalent on-line command ***{axis}={constant}***.

In either command, the specified numerical value becomes the new effective value for the present commanded position for the axis. No motion is caused by this command. The difference between this value and the value for the axis relative to the base machine origin is the offset.

No transformation matrix needs to be defined or selected for this offset method. The motor origin, which is the reference for software overtravel limits and compensation tables, is not changed by these offsets.

The X, Y, and Z-axes in a coordinate system can also be offset through the displacement vector portion of the selected transformation matrix using the **ADIS*n*** or **IDIS*n*** buffered motion program commands. The resulting values in the three displacement elements are the offsets of the axes from the base origins. These offsets do not change the motor origin.

Power PMAC Axis Offsets

In Power PMAC, the only technique for offsetting the axes without changing the underlying motor origins is through the use of the "Bias" terms of the selected transformation matrix. All axes can be offset this way, not just X, Y, and Z.

The offsets can be set directly by writing to the **Tdata[*i*].Bias[*k*]** elements for the selected matrix ***i***, where ***k*** = 0 to 8 for the A, B, C, U, V, W, X, Y, and Z axes, respectively. This is equivalent to using the **ADIS*n*** command in Turbo PMAC.

The offsets can be changed incrementally by writing to the **Tdata[*j*].Bias[*k*]** elements for an "operator" matrix ***j***, then using the **tprop(*i,i,j*)** function to add those offsets to the existing ones.

Note that the **pset*{axis}{data}*** command in Power PMAC *does* change the underlying base motor and axis origins. It is most commonly used when homing to a non-zero position, as with "distance-coded reference marks".

Buffered Lookahead

Both Turbo PMAC and Power PMAC provide special lookahead buffers for real-time checking of the computed trajectories against motor position, velocity, and acceleration limits. In both controllers, the buffer must be created after each power-on/reset with a "define" command.

In both controllers, memory is reserved based on the number of motors assigned to positioning axes in the coordinate system at the time the "define" command is issued. Sufficient memory must be reserved for all motion segments for these motors between calculation time and execution time, and for all synchronous variable assignments between calculation time and execution time.

Turbo PMAC Lookahead Buffer Definition

In Turbo PMAC, memory is reserved for a coordinate system's lookahead buffer with the coordinate-system-specific on-line command:

`define lookahead` ***`{constant},{constant}`***

In this command, the first constant specifies the number of segments that can be stored for each motor assigned to an axis in the coordinate system, and the second constant specifies the number of synchronous variable assignments that can be stored.

This command is most commonly issued from within a "power-on" PLC program (e.g. **`cmd"&1define lookahead 1000,50"`**) that executes once, then disables itself. Note that the order in which various buffers must be defined (and deleted) is very specific in Turbo PMAC.

Buffered Synchronous Assignments

In Turbo PMAC, the buffering of synchronous assignments with lookahead is a two-stage process. In the first stage, the assignments are held in this buffer. In the second stage, they are transferred to the smaller standard synchronous assignment buffer, which only holds them for the standard 2 or 3 moves ahead required for blending and possibly cutter-radius compensation.

The number of assignments per move that can be held for a coordinate system is dependent on the value of **I68**. At the default value of 15, a maximum of 3 assignments per move can be buffered without cutter-radius compensation, or 2 assignments per move with cutter-radius compensation active. Values for **I68** of 7 or less permit more assignments to be buffered per active coordinate system, but for fewer coordinate systems.

Changing the Number of Motors

If it is desired to add or subtract a motor as a positioning axis for the coordinate system (usually to convert a motor between a rotary positioning axis and a velocity-controlled spindle), the existing lookahead buffer must be cleared with a **`delete lookahead`** command, then the buffer redefined with the new number of motors assigned to positioning axes.

Power PMAC Lookahead Buffer Definition

In Power PMAC, memory is reserved for a coordinate system's lookahead buffer with the coordinate-system-specific on-line command:

`define lookahead` ***`{constant}`***

In this command, the single constant specifies the number of segments that can be stored. The minimum number of segments that can be specified is 2048.

In Power PMAC, the order in which buffers are defined is much more flexible than in Turbo PMAC, which requires a very specific order.

The buffer can be defined at power-on/reset from within a "power-on" PLC program (e.g. **`cmd"&1define lookahead 5000"`**), as in Turbo PMAC. The on-line command can also be placed in a project file, such as `pp_startup.txt`, whose on-line commands are directly executed at power-on/reset.

Buffered Synchronous Assignments

The number of synchronous variable assignments that can be stored for the coordinate system in Power PMAC is determined by the value of saved setup element **Coord[*x*].SyncOps**. The default value of 8192 is sufficient for virtually all applications, especially those ported from Turbo PMAC. Synchronous assignments are buffered in a single stage in Power PMAC, with or without lookahead.

Changing the Number of Motors

In Power PMAC, it is possible to assign a motor to a "spindle" (**`S`**) axis in a coordinate system. A motor assigned this way has a column reserved for it in the lookahead buffer, but it is not involved in the lookahead calculations (or any motion program calculations).

This functionality permits a motor to be assigned to a rotary positioning axis at some times, and a velocity-controlled spindle at other times, without the need to delete and redefine the lookahead buffer. (However, moving a motor completely out of the coordinate system or into it from a different coordinate system does requiring deletion and redefinition of the lookahead buffer.)

Note thtat if a Power PMAC motor is always used as a spindle, and never as a positioning axis, it does not need to be declared as an "**`S`**" axis in the coordinate system with the positioning axes. It can be assigned to a different coordinate system, as is common in Turbo PMAC, or as a "null" axis in the same coordinate system.

2D Cutter Radius Compensation

Both Turbo PMAC and Power PMAC support two-dimensional cutter (tool) radius compensation on any plane in XYZ space. There are only a few differences in the use of this functionality between Turbo PMAC and Power PMAC.

Compensated Move Buffering

2D cutter radius compensation requires at least one additional move be buffered compared to no compensation, in order that the compensated intersection point with the next move can be calculated.

Turbo PMAC Move Buffering

If buffering this single added move is sufficient, no special buffer needs to be defined in the Turbo PMAC.

However, if there could be commanded moves in the sequence that have zero distance in the plane of compensation, additional buffering is required. These moves include moves perpendicular to the plane of compensation, moves of only non-XYZ axes, moves with no distance in any axis, delays, and dwells.

In this case, a **DEFINE CCBUF *n*** command is required. This command creates a buffer that can store ***n*** additional compensated moves for the coordinate system, permitting it to work ahead multiple moves while compensation is active.

Power PMAC Move Buffering

In Power PMAC, a buffer for compensated moves *must* be defined for the coordinate system, even if there will be no moves with zero distance in the plane of compensation. This buffer is defined by setting saved setup element **Coord[*x*].CCSize** to the number of moves to a value greater than 0. It must be set to a value of 2 or greater to permit 2D cutter radius compensation to function.

Coord[*x*].CCDistance tells Power PMAC how many in-plane moves to pre-calculate. If it is set less than 2, Power PMAC will pre-calculate 2 moves. This permits both the calculation of the next intersection and a check for interference and overcut with the ability to stop before overcut occurs. It is possible to pre-calculate more in-plane moves to detect rarer interference cases by setting **Coord[*x*].CCDistance** to a value greater than 2.

The buffer defined by **Coord[*x*].CCSize** should be large enough to store the number of in-plane moves desired for interference checking, plus the maximum number of "zero distance" moves that could occur in this sequence.

Compensation Enabling and Disabling

Enabling and disabling 2D compensation is similar in Turbo PMAC and Power PMAC. The first move after the enabling command is the "lead-in" move. The first move after the disabling command is the "lead-out" move.

Turbo PMAC Enabling and Disabling

In Turbo PMAC the following commands are used to enable and disable 2D compensation:

- **CC1** enables compensation to the left in the direction of motion (G41).
- **CC2** enables compensation to the right in the direction of motion (G42).
- **CC0** disables compensation (G40).

The "lead-in" and "lead-out" moves can be either linear or circle mode (circle mode is rarely used).

Power PMAC Enabling and Disabling

In Power PMAC the following commands are used to enable and disable 2D compensation:

- **ccmode1** enables compensation to the left in the direction of motion (G41).
- **ccmode2** enables compensation to the right in the direction of motion (G42).
- **ccmode0** disables compensation (G40).

The "lead-in" and "lead-out" moves must be linear mode. Attempting to use a circle mode move here will result in an error that stops program execution.

Single Stepping in Compensation

Single-step operation in 2D compensation is sometimes used to check and debug a compensated sequence. Because of the additional buffering required for a compensated move sequence, it is important to understand how this works in the controller.

Turbo PMAC Single Stepping

In Turbo PMAC, a single-step (**S**) command causes one move to be *calculated*. Depending on the state of compensation, zero, one, or more than one move may be executed. For example, if the lead-in move is calculated on the **S** command, no move will be executed. If the lead-out move is calculated on the **S** command, both the last of the fully compensated moves and the lead-out move will be executed.

Power PMAC Single Stepping

In Power PMAC, a single-step (**S**) command causes one move to be *executed*. Depending on the state of compensation, zero, one, or more than one move may be calculated. For example, if the lead-in move is executed on the **S** command, that move plus at least two additional moves will be calculated. If the lead-out move is executed on the **S** command, no moves will be calculated, as the lead-out move will have been pre-calculated.

3D Cutter Radius Compensation

Both Turbo PMAC and Power PMAC support three-dimensional cutter (tool) radius compensation in XYZ space. There are several differences in the use of this functionality between Turbo PMAC and Power PMAC.

Defining Tool Nose Geometry

The methods for defining the tool nose geometry are substantially different in Turbo PMAC and Power PMAC. Power PMAC's method is much more flexible than Turbo PMAC's, but it is easy to use it to specify a tool nose compatible with the Turbo PMAC specification.

Turbo PMAC Tool Nose Specification

Turbo PMAC provides two variables to specify the tool nose geometry. The value specified in the buffered motion-program command **`TR{data}`** defines the outer tool shaft radius. The value specified in the buffered motion-program command **`CCR{data}`** defines radius of the rounding of the tool nose end. If these two values are the same, a hemispherical tool nose is specified.

Power PMAC Tool Nose Specification

Power PMAC provides the capability to specify a complex tool-nose geometry as a set of multiple arc segments. To use this capability to specify a geometry compatible with a Turbo PMAC specification, make the following settings with on-line or buffered program Script commands:

Coord[*x*].CC3Data[0].ToolRadius = *{TRvalue}*
Coord[*x*].CC3Data[0].ToolOffset = *{TRvalue – CCRvalue}*
Coord[*x*].CC3Data[0].NdotT = 0.0
Coord[*x*].CC3Data[0].CutRadius = *{CCRvalue}*
Coord[*x*].CC3Data[1].NdotT = 1.0

Compensation Enabling and Disabling

Enabling and disabling 3D compensation is similar in Turbo PMAC and Power PMAC. The first move after the enabling command is the "lead-in" move. The first move after the disabling command is the "lead-out" move.

Turbo PMAC Enabling and Disabling

In Turbo PMAC the following commands are used to enable and disable 2D compensation:

- **`CC3`** enables 3D compensation.
- **`CC0`** disables 3D compensation.

The "lead-in" and "lead-out" moves can be either linear or circle mode (circle mode is rarely used).

Power PMAC Enabling and Disabling

In Power PMAC the following commands are used to enable and disable 3D compensation:

- **ccmode3** enables 3D compensation.
- **ccmode0** disables 3D compensation.

The "lead-in" and "lead-out" moves must be linear mode. Attempting to use a circle mode move here will result in an error that stops program execution.

Compensated Move Commands

Each move command in 3D compensation must specify three things:

1. Uncompensated destination positions or distances for all axes
2. Surface normal vector
3. Tool orientation vector

The destination positions/distances are specified the same on both Turbo and Power PMAC. These will typically include rotary axes as well as the base Cartesian axes.

Turbo PMAC Vector Specifications

In Turbo PMAC, the surface normal vector (perpendicular from the part directed into the tool) is specified using the following syntax:

NX*{data}* NY*{data}* NZ*{data}*

where the three values specify the relative vector components in the X, Y, and Z directions, respectively.

The tool orientation vector (either direction along the turning axis of the tool) is specified using the following syntax:

TX*{data}* TY*{data}* TZ*{data}*

where the three values specify the relative vector components in the X, Y, and Z directions, respectively.

This makes the full Turbo PMAC syntax for the move:

X*{data}* Y*{data}* Z*{data}* A*{data}* B*{data}* NX*{data}* NY*{data}* NZ*{data}* TX*{data}* TY*{data}* TZ*{data}*

Power PMAC Vector Specifications

In Power PMAC, the surface normal vector (perpendicular from the part directed into the tool) is specified using the following syntax:

nxyz I*{data}* J*{data}* K*{data}*

where the three values specify the relative vector components in the X, Y, and Z directions, respectively.

The tool orientation vector (either direction along the turning axis of the tool) is specified using the following syntax:

```
txyz I{data} J{data} K{data}
```

where the three values specify the relative vector components in the X, Y, and Z directions, respectively.

This makes the full Power PMAC syntax for the move:

```
X{data} Y{data} Z{data} A{data} B{data} nxyz I{data} J{data}
K{data} txyz I{data} J{data} K{data}
```

On-Line Command Syntax

The syntax for on-line commands is very similar between Turbo and Power PMAC. The few differences are detailed in this section.

Turbo PMAC Control Character Commands

In Turbo PMAC, there are many control-character commands, typically as "global" versions of motor-specific or coordinate-system-specific letter commands.

In Power PMAC, there are no control-character commands, because the operating system "traps" these characters so they cannot reach the Power PMAC application software. Power PMAC uses the non-modal **#*** ("address all motors") and **&*** ("address all coordinate systems") addressing to create global versions of these commands.

The following table shows the Turbo PMAC control-character commands and their Power PMAC equivalents.

Function	Turbo PMAC Command	Power PMAC Command	Notes
Abort all C.S.	**<CTRL-A>**	**&*a**	
Query all motor status	**<CTRL-B>**	**#1..8?** or **#*?**	Turbo uses **##*n*** to select set of 8 Response is encoded differently
Query all C.S. status	**<CTRL-C>**	**&1..16?**	Response is encoded differently
Disable all PLCs	**<CTRL-D>**	**disable plc 0..31** and **disable cplc 0..31**	
Enable all motors in all C.S.	**<CTRL-E>**	**&*enable** or **#*j/**	
Query all motor following errors	**<CTRL-F>**	**#1..8f** or **#*f**	Turbo uses **##*n*** to select set of 8
Disable all motors in all C.S.	**<CTRL-K>**	**&*disable** or **#*k**	
Feed hold all C.S.	**<CTRL-O>**	**&*h**	
Query all motor positions	**<CTRL-P>**	**#1..8p** or **#*p**	Turbo uses **##*n*** to select set of 8
Quit all C.S.	**<CTRL-Q>**	**&*q**	
Run all C.S.	**<CTRL-R>**	**&*r**	
Single-step all C.S.	**<CTRL-S>**	**&*s**	
Query all motor velocities	**<CTRL-V>**	**#1..8v** or **#*v**	Turbo uses **##*n*** to select set of 8

Axis Attribute Commands

In Turbo PMAC, it is possible (but very rarely used) to set various modal attributes for axes in a coordinate system from on-line commands. In Power PMAC, these attributes must be set from program commands (which is how it is most commonly done in Turbo PMAC as well).

To set these attributes from within an on-line command in Power PMAC, the "**cx**" (one-line, one-shot PLC program) or "**cpx**" (one-line, one-shot motion program) should be used.

Turbo PMAC Syntax	Power PMAC Syntax
`&1 ABS`	`&1 cpx abs`
`&1 ABS(X,Y)`	`&1 cpx abs(X,Y)`
`&1 INC`	`&1 cpx inc`
`&1 INC(Z)`	`&1 cpx inc(Z)`
`&1 FRAX(X,Y)`	`&1 cpx frax(X,Y)`
`&1 NORMAL J-1`	`&1 cpx normal J-1  // Cannot use cx`

Motor and Coordinate System Addressing

Both Turbo PMAC and Power PMAC use the **#*x*** command to modally address Motor *x*, and the **&*x*** command to modally address Coordinate System *x*. However, in Turbo PMAC, this modal addressing applies to all motor-specific or coordinate-system-specific commands on the same hardware communications port (bus or serial). Power PMAC permits separate modal addressing for each software communications thread on the communications port.

Turbo PMAC defaults to **#1** (Motor 1) and **&1** (C.S. 1) on power-on/reset. Power PMAC defaults to **#0** (Motor 0) and **&0** (C.S. 0) on power-on/reset. Note that Motor 0 and C.S. 0 are seldom used for actual motor and coordinate system control in Power PMAC.

Mathematical Features

Most of the mathematical features in the Power PMAC operate in the same manner as on the Turbo PMAC, so no changes will be necessary. This section explains what must be changed.

Internal Numerical Representations

The Turbo PMAC processors have a 24-bit short word, and 48-bit long word format. The Power PMAC processors have a 32-bit short word, and 64-bit long word format. Because of this basic architectural differences, the internal representations of numerical values differ between the controller families.

However, the larger word size of the Power PMAC provides greater range in numerical representations. This means that for all standard usages, converting from a Turbo PMAC representation to the large Power PMAC representation can be done without any need for code changes.

Floating-Point Representations

In Turbo PMAC, the floating-point representation uses a 48-bit long word. It has a 36-bit mantissa and 12-bit exponent. This is the representation used for general-purpose user (P and Q) variables, and it is the common intermediate representation for all user Script program calculations.

In Power PMAC, the main floating-point representation uses a 64-bit long word. It is the IEEE-754 standard "double-precision" floating-point representation, with a 52-bit mantissa and 12-bit exponent. This is the representation used for general-purpose user (P and Q) variables, and it is the common intermediate representation for all user Script program calculations. Due to the greater word length, virtually all uses of floating-point variables should transfer directly.

It is possible in Power PMAC to define 32-bit single-precision floating-point variables (24-bit mantissa, 8-bit exponent) in the user shared memory buffer. It is not recommended to use this variable type to replace Turbo PMAC 48-bit floating-point variables.

Fixed-Point Representations

In Turbo PMAC, the standard fixed-point representation uses a 24-bit short word. This format can be used in user Script programs through pointer (M) variables defined as signed or unsigned integers.

In Power PMAC, the standard fixed-point representation uses a 32-bit short word. This format can be used in user Script programs through pointer (M) variables defined as signed or unsigned integers. Due to the greater word length, virtually all uses of fixed-point variables should transfer directly. (However, if specific rollover or saturation behavior is expected, changes might be required.)

Trigonometric Functions

Turbo PMAC applications using trigonometric functions, especially in degrees, will likely need some changes in the Power PMAC implementation.

Degrees versus Radians

In Turbo PMAC, if **I15** is set to its default value of 0, the trigonometric functions **SIN**, **COS**, **TAN**, **ASIN**, **ACOS**, **ATAN**, and **ATAN2** use degrees. This is the most common configuration. If **I15** is set to 1, these functions use radians instead.

In Power PMAC, to use trigonometric functions with degrees, the functions **sind**, **cosd**, **tand**, **asind**, **acosd**, **atand**, and **atan2d** should be used. To use trigonometric functions with radians, the functions **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, and **atan2** should be used.

Two-Argument Arctangent Function

In Turbo PMAC, the second ("cosine") argument for the two-argument **ATAN2** function is not inside the parentheses. Instead, Turbo PMAC uses the value of **Q0** for this argument. (It uses the value inside the parentheses for the "sine" argument.) In a motion program, **Q0** for the coordinate system executing the motion program is always used. In a PLC program, **Q0** for the coordinate system selected by the **ADDRESS** command is used. In an on-line command, **Q0** for the port's addressed coordinate system is used.

In Power PMAC, both arguments for the two-argument **atan2** and **atan2d** functions are inside the parentheses, separated by commas, "sine" argument first.

Turbo PMAC Syntax	Power PMAC Syntax
Q0=0.866 **Q2=ATAN2(0.5)** **Q2** 30.0007	**Q2=atan2d(0.5,0.866)** **Q2** 30.0007

#define Variable Text Substitutions

In the Turbo PMAC Executive Program (PEWINPRO2), the user can create text substitutions for the underlying numbered variable names to permit meaningful names that make the programs more understandable. For example:

```
#define CycleCount      P50
#define CycleLimit      P51
#define XaxisPos        Q207
#define LaserOn         M35
```

Note that these defined variable names *are* case sensitive. This permits them to be used in case-sensitive C programs as well as Script programs.

PEWIN32PRO automatically substitutes the underlying variable name on downloading the file. It is the user's responsibility to make sure that each PMAC variable is only used once.

In the Power PMAC IDE, the user has this same capability for text substitutions for variable names. The above examples could be used unchanged in the Power PMAC. As with the Turbo PMAC, it is the user's responsibility to make sure that each PMAC variable is only used once.

Unlike the Turbo PMAC, the Power PMAC text substitutions are not just used in the file download. The user-defined names can also be used in other IDE functions, such as the terminal window and the watch window.

The Power PMAC IDE also provides the capability for "declared" variables, in which the IDE automatically assigns the underlying variable to the user name for the variable. For example:

```
global HoleNumber;          // P-variable
csglobal YaxisVel;          // Q-variable
ptr PowerOnLight            // M-variable
```

Note that these declared variable names *are* case sensitive. This permits them to be used in case-sensitive C programs as well as Script programs.

The IDE makes sure that no underlying variable name is used for more than one user-declared variable. However, if the user is also using #define text substitutions for variables, he must make sure that there is no conflict between declared and defined variables.

By default, the IDE assigns P-variables to declared "global" variables starting with **P8192** (Turbo PMAC only has **P0** – **P8191**). It assigns Q-variables to declared "csglobal" variables starting with **Q1024** (Turbo PMAC only has **Q0** – **Q1023** for each coordinate system if more than 4 active coordinate systems). It assigns M-variables to declared "ptr" (pointer) variables starting with **M8192** (Turbo PMAC only has **M0** – **M8191**).

If these starting points (which are part of the Power PMAC "project properties" in the IDE) are used, there should be no conflict between "defined" and "declared" variables. This means that the variable definitions from the Turbo PMAC application can be used unchanged if desired in the Power PMAC application, even if declared variables are added.

Modulo (Remainder) Operator

In both Turbo PMAC and Power PMAC, the **%** (modulo) operator can be used to calculate the remainder when the value before the operator is divided by the value after the operator. In most cases, this operator works in exactly the same manner in Turbo PMAC and Power PMAC. For the operation ***y % x***, where ***x*** > 0, the result ***r*** has the range 0 <= ***r*** < ***x*** in both controllers.

However, if the value after the operator is negative, the results are different between the two controllers.

In Turbo PMAC, for the operation ***y % -x***, the result has the range of **–*x*** to **+*x***. Some find this useful to compute differences when a quantity can roll over in either the positive or negative direction.

In Power PMAC, for the operation ***y % -x***, the result has the range of 0 to ***x***. To duplicate the action in Power PMAC of the Turbo PMAC operation ***y % -x***, use the **rem(*y*¸-*x*)** function.

Buffered Program Command Syntax

Note: The Script language syntax in both Turbo PMAC and Power PMAC is not case-sensitive. Turbo PMAC documentation shows the syntax in all upper-case letters (e.g. **WHILE**). Power PMAC documentation shows the syntax in all lower-case letters (e.g. **while**), except for single-letter commands (e.g. **P1=1**, **X10**) and double-letter axis commands (e.g. **XX10**). This note will use these conventions to help distinguish between Turbo PMAC and Power PMAC examples.

Opening a Program Buffer

In Turbo PMAC, when a fixed program buffer is opened for entry with the **OPEN** command, the existing contents of that program buffer (if any) are kept, and any subsequent program lines are added to the end of the buffer. For this reason, users wishing to overwrite the existing program use the **CLEAR** command immediately after the **open** command.

In Power PMAC, when a fixed program buffer is opened for entry with the **open** command, the existing contents of that program buffer (if any) are automatically erased, and any subsequent program lines are entered into the buffer starting from the beginning. There is no need for a **clear** command, and if one is sent, it will be rejected with an error.

Turbo PMAC Syntax	Power PMAC Syntax
`OPEN PROG 25 CLEAR`	`open prog 25`

In both Turbo PMAC and Power PMAC, when the rotary program buffer is opened for a coordinate system, the existing contents of the buffer are *not* automatically cleared. This permits additions to be made periodically to the buffer as program execution works through already loaded contents.

Program Comments

In Turbo PMAC, all characters on a command line after a semi-colon (**;**) are considered comments.

In Power PMAC, all characters on a command line after a double-slash (**//**) are considered comments. Power PMAC will try to interpret characters after a semi-colon as command syntax, so an unchanged Turbo PMAC comment could cause a syntax error.

Turbo PMAC Syntax	Power PMAC Syntax
`RAPID X0 Y0 ; Go to origin`	`rapid; X0 Y0; // Go to origin`

Semi-colons in Power PMAC syntax are not required. However, they can be used for stylistic reasons, and they help the IDE syntax checker find and highlight errors before downloading.

Conditional Comparators

Equality Conditional Comparator

In Turbo PMAC, the equality comparator is the single equals sign (**=**).

In Power PMAC, the equality comparator is the double equals sign (**==**). Power PMAC will reject the Turbo PMAC comparator as invalid.

Turbo PMAC Syntax	Power PMAC Syntax
`IF (P1 = 1)` …	`if (P1 == 1)` …

Greater-Than-Or-Equal-To Conditional Comparator

In Turbo PMAC, the "greater-than-or-equal-to" comparator is the "not-less-than" sign (**!<**).

In Power PMAC, it is the "greater-than-or-equal-to" sign (**>=**). Power PMAC will accept the Turbo PMAC comparator as valid, but will store it and report it as the Power PMAC syntax.

Turbo PMAC Syntax	Power PMAC Syntax
`IF (P1 !< 1)` …	`if (P1 >= 1)` …

Less-Than-Or-Equal-To Conditional Comparator

In Turbo PMAC, the "less-than-or-equal-to" comparator is the "not-greater-than" sign (**!>**).

In Power PMAC, it is the "less-than-or-equal-to" sign (**<=**). Power PMAC will accept the Turbo PMAC comparator as valid, but will store it and report it as the Power PMAC syntax.

Turbo PMAC Syntax	Power PMAC Syntax
`IF (P1 !> 1)` …	`if (P1 <= 1)` …

Compound Conditions

In Turbo PMAC, the logical operators **AND** and **OR** are used to combine simple conditions into compound conditions.

In Power PMAC, the logical operators **&&** and **||** are used to combine simple conditions into compound conditions.

Turbo PMAC Syntax	Power PMAC Syntax
`IF (P1 > -20 AND P1 < 20)` … `WHILE (P5 > 0 OR P10 = 1)` …	`if (P1 > -20 && P1 < 20)` … `while (P5 > 0 || P10 = 1)` …

In Turbo PMAC, conditions can be compounded over multiple program lines. In Power PMAC, conditions must be compounded within a single program line (but the length of a program line is almost unlimited).

Multi-Line While Loops

In Turbo PMAC, multi-line while loops are delimited by the **WHILE** condition and the **ENDWHILE** command.

In Power PMAC, multi-line while loops are delimited by the "open bracket" (**{**) after the condition and the "close bracket" (**}**) at the end. The open bracket can be on the same line as the condition, or at the start of the next line.

Turbo PMAC Syntax	Power PMAC Syntax

WHILE (P1 < 10)
  {command}
  {command}
ENDWHILE
	while (P1 <= 10) {
	  {command}
	  {command}
	}

Multi-Line If Branches

In Turbo PMAC, multi-line if branches are delimited by the **IF** condition and the **ELSE** or **ENDIF** command.

In Power PMAC, multi-line if branches are delimited by the "open bracket" (**{**) after the condition and the "close bracket" (**}**) at the end. The open bracket can be on the same line as the condition (as shown in the example below), or at the start of the next line.

Turbo PMAC Syntax	Power PMAC Syntax

IF (P1 < 10)
  {command}
  {command}
ENDIF
 	
IF (P2 > 20)
  {command}
  {command}
ELSE
  {command}
  {command}
ENDIF
	if (P1 < 10) {
	  {command}
	  {command}
	}
	
	if (P2 > 20) {
	  {command}
	  {command}
	}
	else {
	  {command}
	  {command}
	}

Blockstart/Blockstop

In Turbo PMAC, the blockstart/blockstop commands can use either the short form (**BSTART**, **BSTOP**) or the long form (**BLOCKSTART**, **BLOCKSTOP**).

In Power PMAC, these commands can only use the short form (**bstart**, **bstop**).

Turbo PMAC Syntax	Power PMAC Syntax

BLOCKSTART
  {command}
  {command}
BLOCKSTOP
	bstart
	  {command}
	  {command}
	bstop

Program Jump Labels

Program jump labels are used in **goto**, **gosub**, and **call** commands.

In Turbo PMAC, program jump line labels have the syntax **N*{constant}*** (without a colon).

In Power PMAC, program jump line labels have the syntax **N*{constant}*:** (with a colon). Power PMAC line labels without a colon are "synchronizing" labels that cannot be "jumped" to.

Turbo PMAC Syntax	Power PMAC Syntax
N50 X10 Y20	**N50: X10 Y20**

Call Commands

Call commands cause a jump to a separate program/subprogram, with a jump back on a return command.

Programs versus Subprograms

In Turbo PMAC, a **CALL** command causes a jump to the motion program (**PROG**) whose number matches the integer component of the value in the **CALL** command.

In Power PMAC, a **call** command causes a jump to the subprogram (**subprog**) whose number matches the integer component of the value in the **call** command.

Turbo PMAC Syntax	Power PMAC Syntax
CALL 500 ; Jump to PROG 500	**call 500 // Jump to subprog 500**

Line Label in Called Program or Subprogram

In Turbo PMAC, the fractional component of the value in a **CALL** command is multiplied by 100,000 to determine the number of the jump label in the called program to jump to.

In Power PMAC, the fractional component of the value in a **call** command is multiplied by 1,000,000 to determine the number of the jump label in the called subprogram to jump to.

Turbo PMAC Syntax	Power PMAC Syntax

`CALL 500.123`
`; Jump to PROG 500`
`; Line label N12300`
	`call 500.123`
	`// Jump to subprog 500`
	`// Line label N123000:`

In Turbo PMAC, if there is no fractional component of the value in a **CALL** command, the jump is *always* to the top of the called program.

In Power PMAC, if there is no fractional component of the value in a **call** command, the jump is to the top of the called subprogram if there is no **N0:** jump label in the subprogram, or to the **N0:** jump label if it exists. (It does not need to be at the top of the subprogram.)

In both Turbo PMAC and Power PMAC, if the fractional component of the value in a call command specifies a non-existent jump label, the jump will be to the top of the called program or subprogram. In Turbo PMAC, this will always be the same as if the **CALL** command value had no fractional component.

In Power PMAC, this can be different from the case where there is no fractional component to the value in the **call** command, because **N0:** does not need to be at the top of the subprogram.

Read Commands

In Turbo PMAC, when a **READ** command is used to pass values to a subroutine, the value with the *n*th letter of the alphabet is assigned to **Q(100+*n*)**, and bit (*n*-1) of **Q100** is set to 1.

In Power PMAC, when a read command is used to pass values to a subroutine, the value with the *n*th letter of the alphabet is assigned to **D(*n*)**, and bit (*n*-1) of both **D0** and **D54** is set to 1. (If a forward-kinematic subroutine is used for position-reporting queries, it is possible for **D0** to be overwritten.)

Turbo PMAC Syntax	Power PMAC Syntax

`CALL 500 D10 E20`
…
`OPEN PROG 500`
`READ(D,E)`
`; Sets Q104 to 10`
`; Sets Q105 to 20`
`; Sets Q100 to 24 (= 2^3 + 2^4)`
	`call 500 D10 E20`
	…
	`open subprog 500`
	`read(D,E);`
	`// Sets D4 to 10`
	`// Sets D5 to 20`
	`// Sets D0, D54 to 24 (= 2^3 + 2^4)`

G, M, T, and D-Code Calls

In Turbo PMAC, the action of G, M, T, and D-codes is fixed:

- **G*{data}*** is always a call to label **N(*{data}*1000*)** of **prog 1000**.
- **M*{data}*** is always a call to label **N(*{data}*1000*)** of **prog 1001**.
- **T*{data}*** is always a call to label **N(*{data}*1000*)** of **prog 1002**.
- **D*{data}*** is always a call to label **N(*{data}*1000*)** of **prog 1003**.

In Power PMAC, the action of G, M, T, and D codes is a little more flexible:

- **G*{data}*** is a call to label **N(*{data}**1000):** of the subprogram whose number is specified by **Coord[*x*].Gprog** (default **subprog 1000**)

- **M*{data}*** is a call to label **N(*{data}**1000):** of the subprogram whose number is specified by **Coord[*x*].Mprog** (default **subprog 1001**)

- **T*{data}*** is a call to label **N(*{data}**1000):** of the subprogram whose number is specified by **Coord[*x*].Tprog** (default **subprog 1002**)

- **D*{data}*** is a call to label **N(*{data}**1000):** of the subprogram whose number is specified by **Coord[*x*].Dprog** (default **subprog 1003**)

These elements must have their default values for the Power PMAC codes to use subprograms of the same number as the Turbo PMAC codes use programs.

S-Code Calls

In Turbo PMAC, an S-code (**S*{data}***) causes coordinate system variable **Q127** to be set to the value of ***{data}***. Note that **S*{data}*** that is used as a **READ** argument (e.g. **M03 S1800** with a **READ(S)** in the subroutine) causes coordinate system variable **Q119** to be set to the value of ***{data}***.

In Power PMAC, an S-code (**S*{data}***) causes coordinate system local variable **D53** to be set to the value of ***{data}***. Note that **S*{data}*** that is used as a **READ** argument (e.g. **M03 S1800** with a **READ(S)** in the subroutine) causes coordinate system local variable **D19** to be set to the value of ***{data}***.

Cutter Radius Compensation Enable/Disable

In Turbo PMAC, the commands **cc0**, **cc1**, **cc2**, and **cc3** are used to enable and disable cutter radius compensation.

In Power PMAC, the commands **ccmode0**, **ccmode1**, **ccmode2**, and **ccmode3** are used to enable and disable cutter radius compensation. (The commands **cc0**, **cc1**, **cc2**, and **cc3** are motion commands for the CC axis.)

PVT Move Mode

In Turbo PMAC, PVT move mode is enabled with the **PVT** program command (no arguments). The time for a PVT move is set by the **TA** or **TM** command, depending on the setting of **I42**.

In Power PMAC, PVT move mode is enabled with the **pvt*{data}*** command, with the value specifying the time for subsequent moves. If the time is to be changed, another **pvt*{data}*** command must be used.

Turbo PMAC Syntax	Power PMAC Syntax
PVT **TA100** **X10:50** **TA200** **X20:0**	**pvt100** **X10:50** **pvt200** **X20:0**

In Turbo PMAC, PVT moves are never segmented, even if I*sx*13 > 0 to enable segmentation in other modes.

In Power PMAC, PVT moves are segmented if **Coord[*x*].SegMoveTime** > 0 to enable segmentation in several move modes.

Spline Move Mode

In Turbo PMAC, spline move mode is enabled with the **SPLINE1** program command (uniform time) or the **SPLINE2** program command (non-uniform time). The time for a spline move is set by the **TA** or **TM** command, depending on the setting of **I42**.

In Power PMAC, spline move mode is enabled with the **spline*{data}*** command, with the value specifying the time for subsequent moves. If the time is to be changed, another **spline*{data}*** command must be used. For the non-uniform time splines to work in the same way as in Turbo PMAC, **Coord[*x*].SplineTimeRotate** must be set to 1.

Turbo PMAC Syntax	Power PMAC Syntax
SPLINE2 **TA100** **X10** **TA200** **X20**	**spline100** **X10** **spline200** **X20**

Circle Center Vector Mode

In Turbo PMAC, circle-mode moves are put in "incremental center vector" mode (in which the center position is specified by a vector from the move starting position) with the **INC(R)** command. This is the default at power-on/reset. Circle-mode moves are put in "absolute center vector" mode (in which the center position is specified by a vector from the active coordinate system origin) with the **ABS(R)** command.

In Power PMAC, circle-mode moves are put in "incremental center vector" mode with the **inc(I..K)** command. This is the default at power-on/reset. Circle-mode moves are put in "absolute center vector" mode with the **abs(I..K)** command.

Rapid Mode Altered Destination

In Turbo PMAC, a programmed rapid-mode move in process can be broken into and the destination changed (without a stop) using an on-line command starting with the "!" character.

In Power PMAC, this same functionality is achieved with a PLC program standard move command. To execute this from an on-line command, the "cx" (one-line, one-shot PLC) functionality can be used.

Turbo PMAC Syntax	Power PMAC Syntax
`&1 !X30 Y40`	`&1 cx X30 Y40`

Motor and C.S. Addressing from PLC Programs

PMAC PLC programs sometimes have a need to address a specific motor or coordinate system. This is done differently in Turbo PMAC and Power PMAC.

Turbo PMAC Addressing

In Turbo PMAC, a PLC program specifies which motor or coordinate system it addresses using the **ADDRESS** command.

For motors, this done with the **ADDRESS #*{constant}*** command, with the ***{constant}*** value directly specifying the motor number, or with the **ADDRESS #P*{constant}*** command, with the value of the specified P-variable specifying the motor number.

For coordinate systems, this done with the **ADDRESS &*{constant}*** command, with the ***{constant}*** value directly specifying the C.S. number, or with the **ADDRESS &P*{constant}*** command, with the value of the specified P-variable specifying the C.S. number.

For motors, this controls which motor a motor-specific command that does not directly specify a motor affects. For example, if Motor 3 is addressed with an **ADDRESS #3** command, **CMD"J+"** causes Motor 3 to jog in the positive direction.

For coordinate systems, this controls which C.S. a C.S.-specific command that does not directly specify a C.S. affects. For example, if C.S. 2 is addressed with an **ADDRESS &2** command, **CMD"A"** causes C.S. 2 to abort its motion program and stop all motors.

For coordinate systems, this also controls which system's Q-variables are used by the PLC program commands.

At power-on/reset, all Turbo PMAC PLC programs are addressing Motor 1 and C.S. 1.

Power PMAC Addressing

In Power PMAC, a PLC program specifies which motor it addresses by setting the value of data structure element **Ldata.motor**. For example, if **Ldata.motor** = 5, **CMD"J+"** or direct command **JOG+** causes Motor 5 to jog in the positive direction.

In Power PMAC, a PLC program specifies which coordinate system it addresses by setting the value of data structure element **Ldata.coord**. For example, if **Ldata.coord** = 3, **CMD"A"** or direct command **ABORT** causes C.S. 3 to abort its motion program and stop all motors..

At power-on/reset, all Power PMAC PLC programs are addressing Motor 0 and C.S. 0.

Issuing On-Line Commands from Programs

In many Turbo PMAC applications, on-line commands are issued from within PMAC programs, particularly from within PLC programs. This is done in Turbo PMAC with the **CMD"*{command}*"** syntax, with the on-line command inside the quotation marks of the program line. The longer-form **COMMAND"*{command}*"** syntax can also be used in Turbo PMAC.

In Power PMAC, on-line commands can be issued from within programs using the same **CMD"*{command}*"** syntax. It is *not* permitted to use the longer-form **COMMAND"*{command}*"** syntax.

However, it is also possible, and recommended, to use "program direct" commands instead. This is a faster and simpler method to execute these actions.

The program direct commands are generally the "full word" versions of the equivalent one- or two-letter on-line commands, with the specified motor or coordinate system number coming after the command.

For example, with a motor-specific command, **CMD"#3J+"** can be replaced by **jog+3**. Using modal addressing, **CMD"J/"** can be replaced by **jog/**. This affects the motor specified by **Ldata.motor** for the program.

For example, with a coordinate-system-specific command, **CMD"&2A"** can be replaced by **abort2**. Using modal addressing, **CMD"S"** can be replaced by **step**. This affects the C.S. specified by **Ldata.coord** for the program.

Compiled PLC Programs

In Turbo PMAC, compiled PLC programs are written in the same Script language as the interpreted PLC programs, but then compiled into executable code with a special Delta Tau compiler before downloading to the Turbo PMAC.

In Power PMAC, compiled PLC programs are written in C, and compiled into executable code with a standard C compiler that is built into the IDE before downloading to the Power PMAC.

If the user does not wish to rewrite the compiled Turbo PMAC PLC programs from PMAC Script language to C, he should first try to execute them as interpreted PLC programs in the Power PMAC. The higher underlying computational speed of the Power PMAC may well permit the program to execute quickly enough without requiring translation to C.

In Turbo PMAC, commands referencing the compiled (Script language) PLC programs use "**PLCC**" in their syntax. (**ENABLE PLCC*n***, **DISABLE PLCC*n***).

In Power PMAC, commands referencing the compiled (C language) PLC programs use "CPLC" in their syntax. (**enables cplc*n***, **disable cplc*n***).

Data Gathering

Power PMAC has data gathering capabilities very similar to (but expanded from) those of Turbo PMAC. In most applications, these capabilities are just used during the development stage of the application, but in some cases, they are used in the application itself.

Specifying What and How to Gather

Most users specify what and how to gather through the PC development software – the PMAC Executive program for Turbo PMAC, and the IDE for Power PMAC. Both programs have similar interactive controls for specifying the registers to gather, the period between gathering samples, and the length of the buffer to gather.

Some users specify these items directly. In Turbo PMAC, these are specified using I-variables I5000 – I5051, which can be saved to flash memory and then automatically loaded into active memory on power-on/reset.

In Power PMAC, these are specified using elements of the **Gather** data structure. The values of these elements are *not* saved to flash memory, so they must be set directly after power-on/reset.

Specifying What to Gather

In Turbo PMAC, **I5001** through **I5048** specify the addresses of the up to 48 registers that can be gathered using the numerical values of the addresses. For example: **I5001 = $800088** for Motor 1 desired position. (The first hex digit tells the Executive program how to interpret the value gathered from the address specified in the last 5 hex digits.)

In Power PMAC, the elements **Gather.Addr[*i*]** (***i*** = 0 to 127) specify the addresses of the registers that can be gathered, usually by specifying the element name for the register followed by the ".a" (address of) suffix. For example: **Gather.Addr[1]=Motor[1].DesPos.a**. The elements **Gather.Type[*i*]** specify how the uploading program `gather_csv` used by the IDE and user PC applications interpret the gathered data.

In Turbo PMAC, **I5050** and **I5051**, each a 24-bit value, form a bit mask specifying which of the 48 potential registers is gathered.

In Power PMAC, **Gather.Items** specifies the number of potential registers to be gathered. The registers specified by **Gather.Addr[0]** through **Gather.Addr[*Items* – *1*]** are gathered.

Specifying How Often to Gather

In Turbo PMAC, **I5049** specifies the gathering period in servo cycles.

In Power PMAC, **Gather.Period** specifies the gathering period in servo cycles.

Specifying the Data Gathering Buffer

In Turbo PMAC, the buffer to store the gathered data must be explicitly configured using the on-line **`DEFINE GATHER`** command. Without a value specified in the command, all of available memory is reserved for gathering. If a value is specified, that numer of long words of memory is reserved for gathering

In Power PMAC, there is no need to define a buffer for the gathered data; it is done automatically. The element **Gather.MaxSamples** specifies the effective size of this buffer.

Enabling and Disabling Gathering

In Turbo PMAC, the data gathering function is enabled using the on-line **GATHER** command. It is disabled using the on-line **ENDGATHER** command. From within a program, this is usually done with the **CMD"GATHER"** and **CMD"ENDGATHER"** syntax.

If bit 1 (value 2) of I5000 is set to 0, gathering will automatically stop if the end of the defined buffer is reached. If bit 1 of I5000 is set to 1, gathering storage will automatically roll over to the beginning of the buffer if the end of the defined buffer is reached, overwriting previously gathered data.

When gathering is stopped by command, storage will resume at the end of the existing data if nothing else is done. To reset storage to the beginning, first a **DELETE GATHER** command must be given, followed by another **DEFINE GATHER** command.

In Power PMAC, the data gathering function is enabled by setting **Gather.Enable=2** or **3**. It is disabled by setting **Gather.Enable** to 0 or 1. These settings can be made either with on-line commands or from within programs.

If **Gather.Enable** is set to 2, gathering will automatically stop if **Gather.MaxSamples** is reached. If **Gather.Enable** is set to 3, gathering storage will automatically roll over to the beginning of the buffer if **Gather.MaxSamples** is reached, overwriting previously gathered data.

If **Gather.Enable** is set to 0, the gathering storage pointer is automatically reset to the beginning of the buffer, so that when gathering resumes, it will start storage from the beginning. If **Gather.Enable** is set to 1, the gathering storage point is left where it was, so that when gathering resumes, it will add on to the end of the existing storage.

Uploading the Gathered Data

In Turbo PMAC, the data that has been stored in PMAC's internal gather buffer is uploaded to the host computer using the **LIST GATHER** command. The data is transferred as text in hexadecimal format, to be interpreted by the host computer algorithms.

In Power PMAC, the data that has been stored in PMAC's internal gather buffer is uploaded to the host computer using PMAC's `gather_csv` utility. The data is transferred as formatted text within a csv file.