



ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

# AN-408 APPLICATION NOTE

## AC Motor Control Using the ADMC200 Coprocessor

by Aengus Murray and Paul Kettle

### INTRODUCTION

This document describes the design of an ac motor control system using the ADSP-2115 digital signal processor (DSP) and the ADMC200 motion coprocessor. The architecture illustrated in this application note can be utilized for a number of systems, and was chosen for illustration purposes only. The purpose of the document is to demonstrate the use of the ADMC200 in the digital implementation of a high speed motor control system.

The document starts with a system hardware description that illustrates the hardware simplification when using the ADMC200 and ADSP-2115. A typical control scheme is described to demonstrate the ADMC200 features. A control algorithm is produced to match the described control scheme. The DSP control software is presented in both pseudo code and in DSP assembly code. This code demonstrates how the ADMC200 functions can be closely integrated into the control algorithm. Timing information is also presented which shows how the shaft torque functions could be implemented in less than 20  $\mu$ s.

### SYSTEM HARDWARE

The complete system consists of a permanent magnet ac servo motor with a shaft mounted resolver, a three phase power inverter, and the motor control circuit. The primary ICs in the control circuit are the ADSP-2115, the ADMC200, and the AD2S90 resolver-to-digital converter (RDC). The DSP is the shaft control processor and carries out all the motion control and torque current loop functions. The ADMC200 is the interface between the DSP and the inverter, and in addition provides the vector

transformation functions required for ac motor control. The interface to the host controller can be either via the DSP data and address bus or via the serial port.

The motion control software is stored on an external 8-bit EPROM and is automatically loaded into the DSP's 1K words of internal program RAM on power-up. Each of the 24-bit program words is stored on the EPROM in a 4-byte segment. The DSP boot firmware copies the program from the EPROM to the internal RAM in the correct order to rebuild the 24-bit wide program memory. This arrangement limits the external EPROM requirement to just a single slow memory device.

It is possible to switch between eight pages of program memory RAM stored in a 32K EPROM. For example, the first boot page could contain the programs which will initialize all data variables (look up tables, etc.), configure the ADMC200 registers (setup PWM registers, etc.), and perform self-diagnostic functions. While the second page can contain the motion control algorithms which are loaded at the end of the initialization phase.

The ADMC200 device can be connected directly to the DSP data and address busses, as described in detail in Appendix C. The internal registers can be written to in the same way as data RAM placed in the low memory address space. The shaft control algorithm can be timed through the ADMC200 CONVST pin or via the interrupt pin on the DSP.

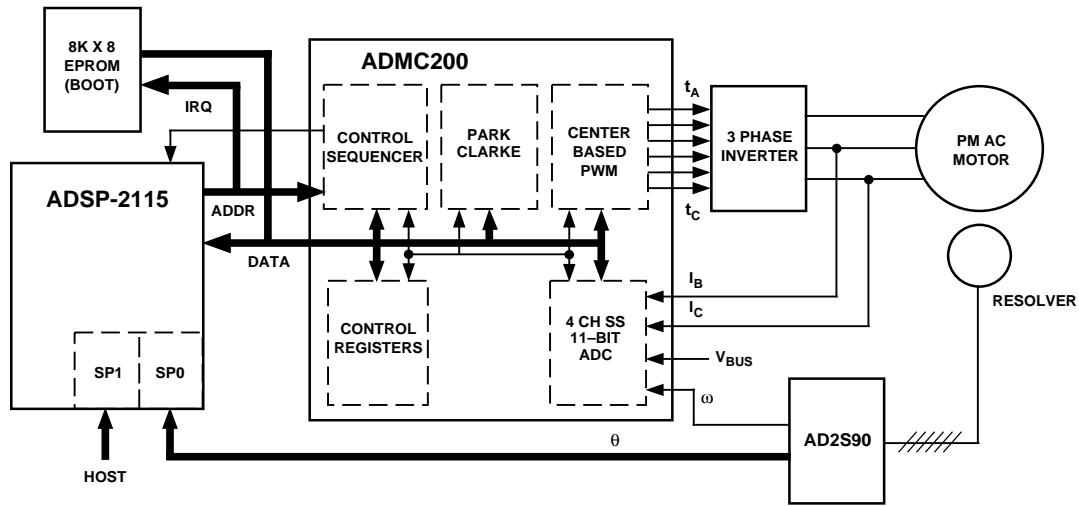


Figure 1. Motion Control Hardware

### THE CONTROL SYSTEM

Figure 2 below illustrates a field oriented control scheme for a permanent magnet synchronous motor. The outer position and velocity loop calculates the torque demand which is the input  $I_q^*$  reference for the current loops. At speeds less than the base speed, the  $I_d^*$  reference current will be zero. If an extended constant power speed range is required, the field control scheme can introduce some field weakening by setting a negative  $I_d^*$  value as a function of the motor speed.

\*Reference input to controller.

The analog-to-digital converter (ADC) block within the ADCMC200 samples the motor currents; the vector transformation block performs a reverse Clarke and Park vector transformation on these ac current waveforms, mapping them into equivalent direct and quadrature current components within a rotating reference frame ( $I_d$   $I_q$ ). A current loop control algorithm implemented on the DSP calculates desired  $V_d$  and  $V_q$  voltages for the motor. Finally the ADCMC200 forward vector transformation block performs a forward Park and Clarke transformation, mapping these direct and quadrature motor voltages into ac voltages within the stator reference frame. The DSP scales and then write these results to the PWM block of the ADCMC200.

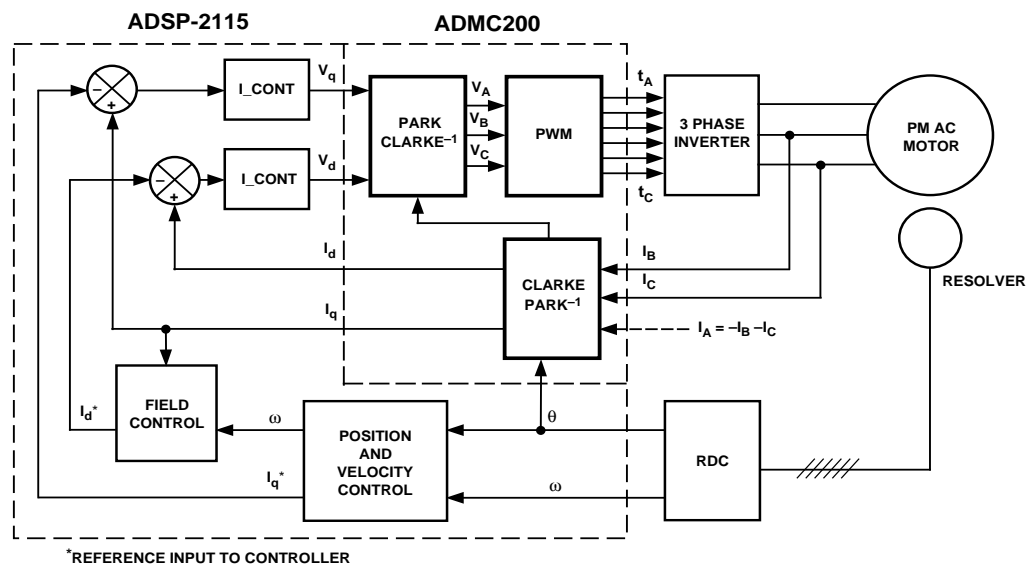


Figure 2. Motion Control System

### CONTROL ALGORITHM

An outline of the control algorithm, based on the scheme in Figure 2, is given in Table I. The functions in bold are implemented on the ADCM200 coprocessor. The scheduling of the control algorithm is synchronized with the ADC interrupt service routine. In this application the ADC will generate an interrupt at a rate of 10 kHz. The torque control loop is realized at this frequency, while the motion loop is scheduled every fourth

ADC sample. The motion control loop can be segregated into a *posterior* and a *priori* measurement portions. The *a priori* portion is executed during the first three time slices while the *posterior* portion is executed in the fourth interval. The basic torque loop functions can be carried out in less than 20  $\mu$ s; this leaves the remaining 80% of the time for the motion control loop and other functions. The code required takes up less than 10% of the available 1K of internal program RAM.

Table I. Control Algorithm Outline

inputs	ADMC200 write register	<b>Motion_Control</b> functions	out	ADMC200 read register
		Read_RDC	$\omega$ $\rho$	ADCAUX
$\omega$		position_velocity_loop	$I_q^*$	
$\omega$		Field_control ( $I_d^* = 0$ for $\omega < \omega_{base}$ )	$I_d^*$	

inputs	ADMC200 write register	<b>Torque_control</b> functions	out	ADMC200 register
		<b>Sample_phase_currents</b>	$I_b$ $I_c$	ADCV ADCW
$I_b$ $I_c$ $\rho$	PHIP2/VQ PHIP3 RHO	<b>Clarke_Park<sup>-1</sup></b>  ( $I_a = -I_b - I_c$ )	$I_d$ $I_q$ $I_a$	ID/PHV1/VX IQ/PHV2 IX/PHV3
$I_q$ $I_q^*$ $\omega$		$I\_control\_q$	$V_q$	
$I_d$ $I_d^*$ $\omega$		$I\_control\_d$	$V_d$	
$V_q$ $V_d$ $\rho$	PHIP1/VD PHIP2/VQ RHOP	<b>Park _ Clarke<sup>-1</sup></b>	$V_a$ $V_b$ $V_c$	ID/PHV1 IQ/PHV2 IX/PHV3
$V_a$ $V_b$ $V_c$		PWM_scale	$T_a$ $T_b$ $T_c$	
$T_a$ $T_b$ $T_c$	PWMCHA PWMCHB PWMCHC	<b>PWM_out</b>		

## TORQUE\_LOOP FUNCTIONS

A more detailed explanation of the torque control loop is given in Table II. In this example the ADC sample rate is tied to the PWM frequency by connecting the PWMSYNC pin to the CONVST pin. All four channels are converted in sequence, and the completion of the conversion process will be signaled by an interrupt to the DSP, which initiates the Torque\_loop. The control inputs to this loop are the reference currents  $I_d^*$  ( $I_{d\_ref}$ ),  $I_q^*$  ( $I_{q\_ref}$ ) derived from the motion control loops. The measured inputs are two motor phase currents ( $I_B$ ,  $I_C$ ) and the shaft angle  $\rho$  ( $\rho$ ). The outputs are six deadtime compensated PWM signals for the inverter.

Only two current values are measured by the ADC since the third phase is derived by the ADMC200 vector transformation block. The reverse vector transformation produces  $I_d$  and  $I_q$  currents in the rotor reference frame. The reverse Clarke and Park vector transformation block executes in 37 clock cycles; the block interrupts the processor on completion. While the vector rotation has been performed, the computation resources of the DSP are thus free to perform other tasks such as overcurrent detection or velocity signal conditioning. The current loops are based on a machine back emf model with a PI error loop, as outlined in Appendix B.

The winding resistance drops ( $IR_q = I_q^* \cdot R_{stator}$ ) are precalculated by the motion control loop. The current loop outputs, the  $V_d$  and  $V_q$  voltages, are fed to the PARK block. When the shaft angle is loaded into the forward angle register (RHOP), the forward PARK transformation produces the three motor phase voltages.

The phase voltages can be scaled as a function of the bus voltage and the PWM period. Deadtime compensation is implemented by adding/subtracting the deadtime to/from the PWM zero offset time depending on the phase current polarity. The DSP writes the three PWM on time duty cycles to the PWM input register. The PWM output registers are latched with the new duty cycle information at the beginning of the next PWM period. The updated PWM duty cycle data is only latched if all three of the input registers have been updated.

Sample code for some of these functions is given in Appendix A. Two sets of circular arrays are used for the phase currents and phase voltages. The ADMC200 registers are memory mapped to the data memory address bus and so are read as data variables. The wait states for the ADMC200 access are defined in the ADSP-2115 data memory wait state control register. The use of indirect addressing in the PWM\_scale routine allows the use of the simultaneous multiply and memory read function.

Table II. Torque Loop Algorithm

Start_torque_loop	wait for ADC interrupt	elapsed time
Read_currents:	ADC_int: read I <sub>ph</sub> (2) from ADCM200: ADCV read I <sub>ph</sub> (3) from ADCM200: ADCW	1.1 μs
load PARK registers for stator to rotor transformation	write I <sub>ph</sub> (2) to ADCM200: PHIP2 write I <sub>ph</sub> (3) to ADCM200: PHIP3 write ρ to ADCM200: RHO	
<b>Clarke_Park</b> <sup>-1</sup>	<b>meanwhile</b> check for over current wait for RPARK interrupt	5.1 μs
read PARK registers	RPARK_int: read I <sub>d</sub> from ADCM200: ID read I <sub>q</sub> from ADCM200: IQ read I <sub>ph</sub> (1) from ADCM200: IX	5.9 μs
I_control_d error driven PI loop (save DV <sub>q_n</sub> and DI <sub>q_n</sub> values) +machine equations	D <sub>Id_n1</sub> = I <sub>d_ref</sub> - I <sub>d</sub> DV <sub>d_n1</sub> = DV <sub>d_n</sub> + K <sub>Pd</sub> *(D <sub>Id_n1</sub> -K <sub>Id</sub> *D <sub>Id_n</sub> ) DV <sub>d_n</sub> = DV <sub>d_n1</sub> D <sub>Id_n</sub> = D <sub>Id_n1</sub> V <sub>d</sub> = DV <sub>d_n1</sub> + (L <sub>s</sub> .I <sub>q</sub> )*velocity + [I <sub>Rd</sub> =I <sub>d</sub> *.R <sub>S</sub> ]	7.8 μs
I_control_q: error driven PI loop (save DV <sub>q_n</sub> and DI <sub>q_n</sub> values) +machine equations	D <sub>Iq_n1</sub> = I <sub>q_ref</sub> - I <sub>q</sub> DV <sub>q_n1</sub> = DV <sub>q_n</sub> + K <sub>Pq</sub> *(D <sub>Iq_n1</sub> -K <sub>Iq</sub> *D <sub>Iq_n</sub> ) DV <sub>q_n</sub> = DV <sub>q_n1</sub> D <sub>Iq_n</sub> = D <sub>Iq_n1</sub> V <sub>q</sub> = DV <sub>q_n1</sub> + (K <sub>E</sub> +L <sub>s</sub> .I <sub>d</sub> )*velocity + [I <sub>Rq</sub> =I <sub>q</sub> *.R <sub>S</sub> ]	9.6 μs
load PARK registers for rotor to stator transformation	write V <sub>q</sub> to ADCM200 VQ; write V <sub>d</sub> to ADCM200: VD; write ρ to ADCM200: RHOP;	9.9 μs
<b>Park_Clarke</b> <sup>-1</sup>	<b>meanwhile</b> do some velocity filtering wait for FPARK interrupt	13.9 μs
read PARK registers	FPARK_int: read V <sub>ph</sub> (1) from ADCM200: PHV1 read V <sub>ph</sub> (2) from ADCM200: PHV2 read V <sub>ph</sub> (3) from ADCM200: PHV3	14.8 μs
<b>PWM_out</b> deadtime adjustment reverse for -ve I <sub>ph</sub> calculate PWM times write to registers	for I = 1,3 T <sub>0</sub> = TPWM/2 + TPD IF I <sub>ph</sub> (I) LT 0 T <sub>0</sub> = TPWM/2 - TPD T(I) = T <sub>0</sub> + VSCALE*V <sub>ph</sub> (I) write T(I) to PWMCH(I)_ADMC200 end_for_loop	18.0 μs

**APPENDIX A: ASSEMBLY CODE SEGMENT FOR TORQUE LOOP**

<p>Read_ currents:  Read ADC registers and write to PARK input registers</p> <p><b>RPARK starts: Clarke_Park<sup>-1</sup></b></p> <p>Read PARK output registers including derived third phase current value.</p> <p>set up pointers for current loop call</p> <p><b>FPARK starts: Park_Clarke<sup>-1</sup></b></p>	<pre> ADC_int: i3=^I_ph;                {pointers for I_ph CIRC array} m1=1; l3=3; ax0=dm(ADCV);            {Read lph(2) from ADCV_ADMC200} dm(i3,m1)=ax0; dm(PHIP1_VD)=ax0;        {write lph(2) to PHIP1_ADMC200} ax1=dm(ADCW);            {Read lph(3) from ADCW_ADMC200} dm(i3,m1)=ax1; dm(PHIP2_VQ)=ax1; ay0=dm(theta)           {write theta to RHO_ADMC200;} dm(RHO)=ay0             {start RPARK by writing to ADMC200} i7=^error_int           {error if next call takes &gt; 37 cycles } call over_current       {check for over current during PARK} i7=^RPARK_int;          {interrupt vector set to RPARK } rti;  RPARK_int: ay0=dm(IX_PHV3);        {Read lph(1) from ADCW_ADMC200} dm(i3,m1)=ay0; i0=^lds;                {pointer for Id and Iq} l0=2; ay0=dm(ID_PHV1);        {Read Ids from ID_ADMC200} dm(i0,m1)=ay0; ax0=dm(IO_PHV2);        {Read Iqs from IO_ADMC200} dm(i0,m1)=ay0; i4=^ld_ref; m4=1; l4=0; i1=^Dld_n l1=2; call l_control          {current loop for Id return Vd in mr1} dm(PHIP1_VD)=mr1;      {write Vd to VD_ADMC200} i1=^DIq_n; i4=^Iq_ref; modify(i0,m1);          {i0 now points to Iq} call l_control          {current loop for Iq return Vq in mr1} dm(PHIP2_VQ)=mr1;      {write Vq to VQ_ADMC200} ax0=dm(theta) dm(RHOP)=ax0;          {write theta to RHO_ADMC200: start FPARK} i7=^error_int;         {error interrupt vector for &gt; 37 cycles} call ADC_filters; i7=^FPARK_int;         {store vector for park interrupt} rti; </pre>
<p>set up pointers for Vph</p> <p>Read PARK output registers</p>	<pre> FPARK_int: i2=^V_ph;                {pointers for V_ph} m2=1; l2=3; ax0=dm(PHV1_VD);        {Read Va from PHV1_ADMC200} dm(i3,m3)=ax0; ax0=dm(PHV2_VQ);        {Read Vb from PHV2_ADMC200} </pre>

<p>PWM_out including: deadtime adjustment</p>	<pre> dm(i3,m3)=ax0; ax0=dm(PHV3);           {Read Vc from PHV3_ADMC200} dm(i3,m3)=ax0; i1=^PWMTCCHA;           {pointers for pwmtcha_ADMC200} m1=1; l1=1; ay1=pwmttdt;             {dead time compensation} ax1=pwmtm_0;             {pwmtm/2} my0=dm(pwm_sc);         {pwm scale factor} mr0=0; cntr=3; do pwmout until ce ;     ar=ax1+ay1, ax0=dm(i3,m3);   {t0 incl +tdt adjust}     af=pass ax0;                 {check current polarity}     if lt ar = ax1-ay1;          {if -i then -tdt adjust}     mr1=ar;                       load adjusted t0}     mx0=dm(i2,m2);               {v_ph}     mr=mr+mx0*my0(ss);                                 {ton=t0+(+/-tdt)+v_ph*(t0/vbus) } pwmout:     dm(i1,m1)=mr1;               {write to pwmtchx_ADMC200 } call motion_control {run position and velocity in remaining time} rti; </pre>
---	---

<p>current_loop(13 ops) PI error driven loop.</p> <p>lq error</p> <p>integral part: proportional part: save old values: +machine equations add IR drop to Vq</p> <p>q axis flux q back emf total Vq</p>	<pre> { af = DIq_n1 = Iq_ref - Iq mr = int = DIq_n * KIq ar = sum = DIq_n1 + int mr = DVq_n1 = DVq_n+(sum)*KPq   PI part ar = Vq_sum = DVq_n1 + IRq mf = PHI = KEq+Id*Ls mr = Vq = Vq_sum + velocity*PHI   machine equations } I_control: ax0=dm(i0,m1), ay0=pm(i4,m4);   {ax0=Iq ay0=Iq_ref } af=ay0-ax0, mx0=dm(i1,m1), my0=pm(i4,m4); {mx0=DIq_n, my0=KIq} mr=mx0*my0(ss), my0=pm(i4,m4);   {mx0=KPq} ar = mr1+af, mr1=dm(i1,m1);       {mr1=DVq_n} mr=mr+ar*my0(ss), ay0= pm(i4,m4); {ay0=IRq} dm(i1,m1)=af;                     {DIq_n =af} dm(i1,m1)=mr1;                    {Dvq_n=mr1} ar =mr1+ay0, mr1= pm(i4,m4);       {mr1=KEq} mx0=dm(i0,m1), my0= pm(i4,m4);     {mx0=Id,my0=Lsq} mf=mr+mx0*my0(ss), mr1=ar; mx0=dm(velocity); mr=mr+mx0*mf(ss); rts; </pre>
---	--

## APPENDIX B: CURRENT LOOP EQUATIONS

The current loop equations are based on a PI driven error loop and the machine winding model.

The machine equations for the  $U_d$  and  $U_q$  voltages are:

$$\begin{aligned} U_d &= I_d \cdot R_s + \omega_r \cdot (L_s \cdot I_q) \\ U_q &= I_q \cdot R_s + \omega_r \cdot (L_s \cdot I_d + K_E) \end{aligned} \quad \text{Eq. (1)}$$

The PI loop is of the form:

$$G_{PI}(s) = K_P \cdot \left( 1 + \frac{K_I}{s} \right) \quad \text{Eq. (2)}$$

The discrete form of this can be obtained by substituting

$$s = \frac{T_s}{2} \cdot \left( \frac{z-1}{z+1} \right) \quad \text{Eq. (3)}$$

This gives an equation of the form:

$$\begin{aligned} G_{PI}(z) &= KP \cdot \left( \frac{z+KI}{z-1} \right) \\ \text{where, } KP &= K_P \cdot \left( 1 + \frac{K_I \cdot T_S}{2} \right), \quad KI = \left( \frac{1 - \frac{K_I \cdot T_S}{2}}{1 + \frac{K_I \cdot T_S}{2}} \right) \end{aligned} \quad \text{Eq. (4)}$$

The difference equation for this transfer function is:

$$DV_{K+1} = DV_K + KP \cdot DI_{K+1} + KP \cdot KI \cdot DI_K \quad \text{Eq. (5)}$$

The applied  $V_d$  and  $V_q$  voltages are the sum of the calculated machine winding voltages, Equation 1), and the error correcting term from the PI loop Equation 5.

$$\begin{aligned} V_d &= \Delta V_d + U_d \\ V_q &= \Delta V_q + U_q \end{aligned}$$

## APPENDIX C: ADCM200 INTERFACE

### ADMC200 Address Decoding

The ADCM200 address lines is connected directly to the lower four bits of the DSP address bus (A0 to A3). The chip select line ( $\overline{CS}$ ) may be decoded from the higher address lines. For example if  $\overline{CS}$  is derived from A4 and A5 using a dual input NAND the ADCM200 is placed in the memory address space 0030 to 003F. The DSP data memory select pin ( $\overline{DMS}$ ) is connected to the chip select pin on the ADCM200. The  $\overline{DMS}$  can be logically combined with some of the higher address lines if a number of devices are to reside on the bus. The 12-bit data lines should be connected to the lower 12 bits of the DSP data bus, thus scaling full scale positive output to  $+2^{11}$ . The DSP  $\overline{DMRD}$  is connected to the ADCM200  $\overline{OE}$  and is asserted in read operations. The DSP  $\overline{DMWR}$  line is connected to the  $\overline{WR}$  line to enable write operations to the ADCM200. The read and write operations will take only one CPU clock cycle, so the appropriate memory wait state registers can be set to zero.

The ADCM200 will produce an interrupt output on completion of an A/D conversion sequence or a PARK transformation. The interrupt output can be connected to the IRQ2 line of the DSP, so that the DSP reads the appropriate registers immediately on the completion of each ADCM200 function. The  $\overline{RST}$  should be enabled via a system reset line, so that the ADCM200 is reset on a DSP reset.