



T E C H N O S O F T

DMCode – L plug-in for DMCD-Pro **Object Code for Brushless and Induction Motor Speed Control**

Technosoft is a Third Party of Texas Instruments supporting the TMS320C24xx DSP controllers from the C2000 family.

To get your project started rapidly, Technosoft offers the **DMCode–L** plug-in for **DMCD-Pro**, a complete object library for digital motion control and demo application code for speed control of Brushless and Induction Motors. Please find the description of these examples on the following pages.

DMCD-Pro (Digital Motion Control Developer Pro)

Digital Motion Control Developer for DSP software development for the TMS320F24xx

- Incorporated Debugger Watch Windows
- Memory and I/O registers view/modify
- Integrated source code editor with powerful programming options
- Project Management System
- Tracing Module
- Plug Ins
- Reference Generator Module
- Application Sources (Optional)

Fully integrated DSP software development environment

- Windows environment with DSP-specific functions gets you started quickly

Incorporated Debugger

- Observe/edit global variables during the debugging process
- Breakpoints, single stepping, stopping and continuing the current program
- Observe/edit global variables during the debugging process
- You can view/edit both data and program memory contents of the DSP target board
- Disassembly window with disassembled instructions with symbolic information for effective debugging
- View/edit I/O and internal registers of the DSP processor

Integrated source code editor with powerful programming options

- Each file has its own window and you can edit many views of the same file
- Advance search and replace mechanism
- Syntax coloring for C and ASM (TI's assembly syntax is also supported)
- Bookmarks management

Project Management System

- The system provides an effective way of quickly visualizing, accessing, and manipulating all the project files and their dependencies
- The result is a concise, highly organized project management system that promotes a very efficient development process.

Tracing module

- The system provides an advanced graphical tool for the analysis and evaluation of motion control applications
- The program variables may be stored during the real-time execution of the motion, and then up-loaded and visualized in the graphical environment

Plug-ins

- This module allows to users using external module functions into their DSP applications. Basically, you may select one or more external modules from a list containing all available external modules.
- If the reference generator plug-in is included in your application, you may define the motion reference at a high level, in DMC Developer, download it and execute it automatically on the DSP board



T E C H N O S O F T

DMCode - L Motion Control Library Plug-In for DMCD-Pro

The Motion Control Library implements a set of functions used in the real-time control structure for motion systems. Based on these functions, the library allows the user to develop different types of Digital Motion Control applications.

Motion Library is structured on the following functional module levels:

1. **Level 0** modules which contain DSP hardware depending functions
2. **Level 1** modules which contain algorithm-type functions

As it was structured, a part of the library is hardware dependent, mainly the RTC Kernel, the measurement modules, PWM module. Another part of it contains only algorithmic functions.

The Motion Control Library functions can be classified, based on the following criteria:

- Initialization functions
- Interrupt functions (ISR)
- Fast-loop functions
- Slow-loop functions
- Auxiliary functions

The Plug-In Kit contains an object library file (*MCLib24xx.lib*), several header files associated to library modules and four demo applications that implement Digital Motion Control structures for: Brushless DC Motors (BLDC), Permanent Magnet Synchronous Motors (PMSM) and Induction Motors (IM).

The demos for the Brushless Motor implement speed control applications for both PMSM and BLDC operating modes. The demos for the Induction Motor implement the vector control (IMVC) and V/F control (IMVF) applications.

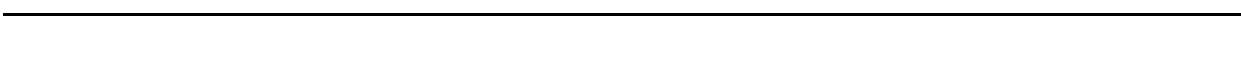
The demo applications are structured as **DMCD-Pro** projects, containing the source files of the application kernel, where from the library function calls are performed, and all the other necessary files (.h, .prf and .cmd files) needed to implement the motion control application on a TMS320F24xx DSP module as MCK24xx or ACPM750 from Technosoft.

The application kernel code of the demo applications contains function calls from the *MCLib24xx* library. This library is linked with the application demo files within the DMCD-Pro projects.

As a general remark, in order to get a maximum performance application implementation, the motion programs were implemented using C programming language and the library routines are implemented in both C and assembler: the time-critical parts, as controllers, coordinates transformations, etc.

The C code contains the main function and also the interrupt control routines associated to control loops. The main function contains the calls to all the initialization routines, from the library, as well as the main kernel of the application. This offers a maximum readability of the demo software structure.

Versions of the library are provided for the TMS320F240, TMS320F243 and TMS320F2407 DSP controllers.





TECHNOSOFT

Permanent Magnet Synchronous Motor motion application

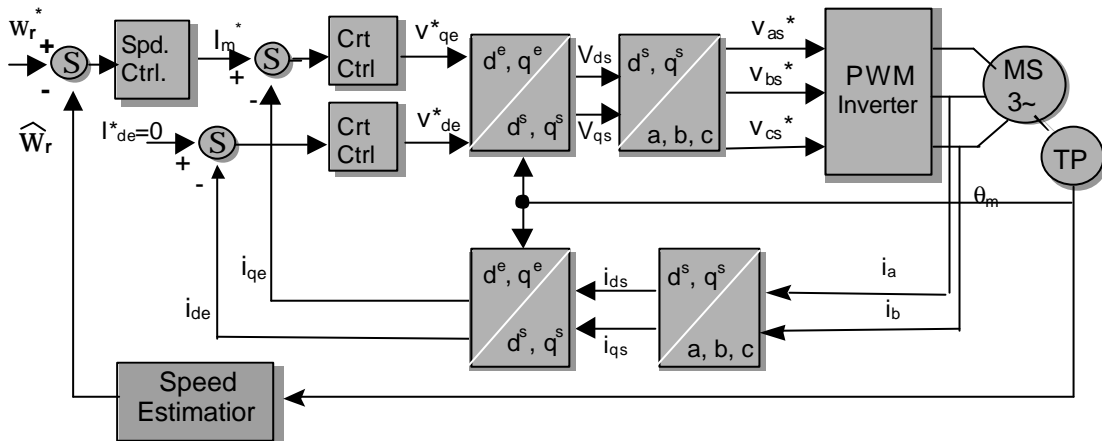
The permanent magnet synchronous motor (PMSM) motion application implements a vector control method to drive in sinusoidal mode the three-phase brushless motor included in the **MCK24xx** kit.

The demo is supplied as a TMS320F24xx application, structured as a project of the **DMCD-Pro** platform. The source files of the application kernel, where from the library function calls are performed, are included in the project structure.

The application is a speed control application of the brushless motor operating in sinusoidal mode.

Basic structure of the control scheme for the PMSM application

The **PMSM** application control scheme is presented in the figure below. As one can see, the scheme is based on the measure of two phase's currents and of the motor position. The speed estimator block is a simple encoder position difference block over one sampling period of the speed control loop. The measured phase currents, i_a and i_b , are transformed into the stator reference frame components, i_{ds} and i_{qs} . Then, based on the position information, these components are transformed into the rotor frame direct and quadrature components, i_{de} and i_{qe} . The speed and current controllers are **PI** discrete controllers. The inverse coordinates transformation is used for computation of the phase voltages references, v_{as}^* , v_{bs}^* and v_{cs}^* , applied to the inverter, starting from the values of voltage references computed in the d and q reference frame (v_{de}^* , v_{qe}^*). Thus, the 6 full compare PWM outputs of the DSP controller are directly driven by the program, based on these reference voltages.



PMSM control scheme

The direct current component reference i_{de}^* is set to **0**, case corresponding to the motion of the motor in the normal speed range, without considering a possible field weakening operation.

Based on this application, representing a complete, ready-to-run motion example, the user gets all the information needed as a convenient starting point for the development of his own applications.

Using the advanced features of **DMCD-Pro**, the **motion reference** can be defined at high-level, from the Windows environment. Calling the **data logger** function allows the user to visualize any of the global variables of the program, and effectively analyze and debug his application.



TECHNOSOFT

Brushless DC motion demo application

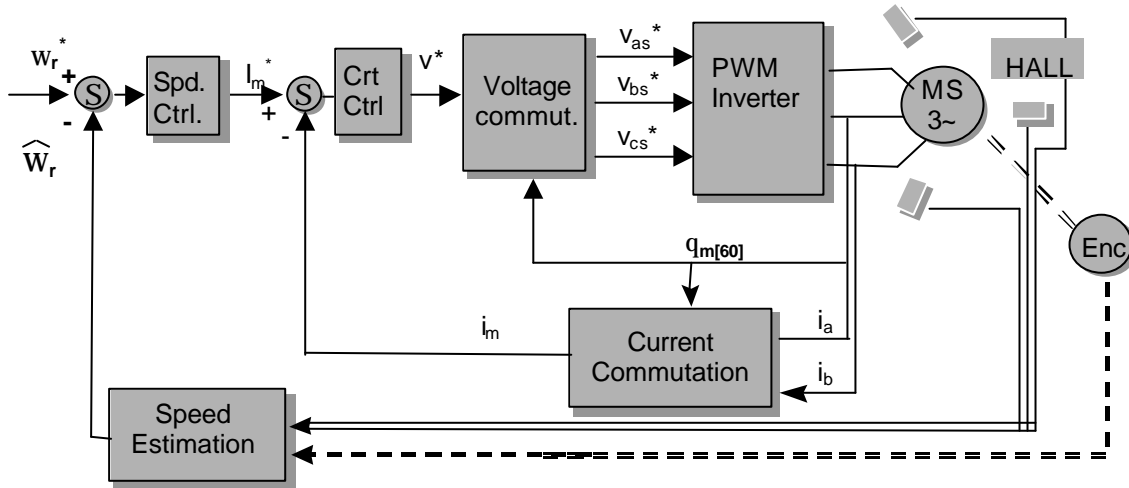
The brushless DC motor (BLDC) motion application implements a block commutation control method to drive the three-phase brushless motor included in the **MCK24xx** kit.

The demo is supplied as a TMS320F24xx application, structured as a project of the **DMCD-Pro** platform. The source files of the application kernel, where from the library function calls are performed, are included in the project structure.

The application is a speed control application of the brushless motor operating in trapezoidal mode.

Basic structure of the control scheme for the BLDC application

The **BLDC** application control scheme is presented in the figure below. As one can see, the scheme is based on the measure of two phase's currents and of the motor position. The speed estimator block is a simple encoder position difference block over one sampling period of the speed control loop. The measured phase currents, i_a and i_b , are used to compute the equivalent DC current in the motor, based on the Hall sensors position information. Remark that the Hall sensors give a 60 electrical degrees position information. The speed and current controllers are **PI** discrete controllers. Only one current controller is needed in this case, similar to a DC motor case. The voltage commutation block implements (by software) the computation of the phase voltages references, v_{as}^* , v_{bs}^* and v_{cs}^* , applied to the inverter. Practically, the 6 full compare PWM outputs of the DSP controller are directly driven by the program based on these reference voltages. In the **BLDC** case, only four of the inverter transistors are controlled for a given position of the motor. The scheme will commute to a specific command configuration, for each of the 60 degrees position sectors, based on the information read from the Hall sensors.



BLDC control scheme

Based on this application, representing a complete, ready-to-run motion example, the user gets all the information needed as a convenient starting point for the development of his own applications.

Using the advanced features of **DMCD-Pro**, the **motion reference** can be defined at high-level, from the Windows environment. Calling the **data logger** function allows the user to visualize any of the global variables of the program, and effectively analyze and debug his application.



TECHNOSOFT

Induction Motor Vector Control motion application

The Induction Motor Vector Control (**IMVC**) motion application implements a vector control method to drive the three-phase induction motor (Sieber motor) with ACPM750.

The demo is supplied as a TMS320F24xx application, structured as a project of the **DMCD-Pro** platform. The source files of the application kernel, where from the library function calls are performed, are included in the project structure.

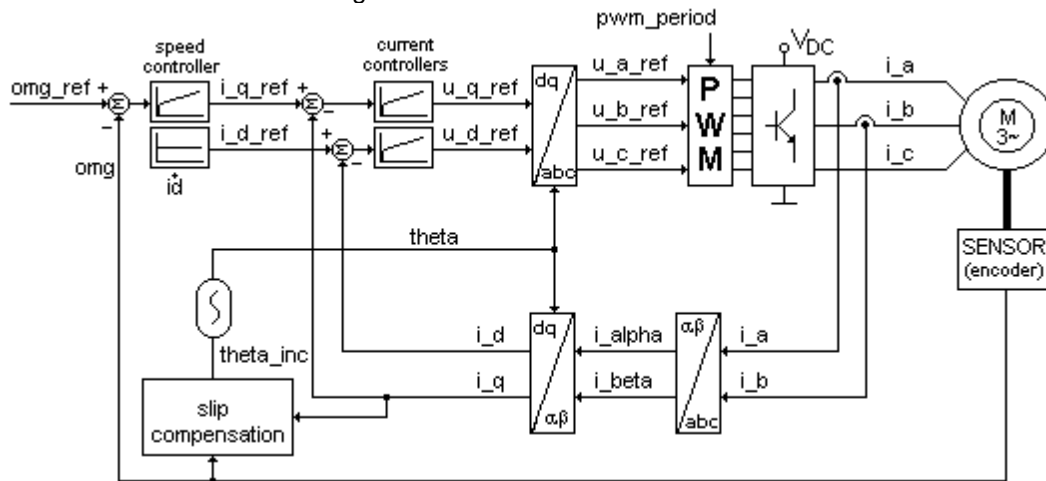
The application is a speed control application of the induction motor.

Basic structure of the control scheme for the IMVC application

The **IMVC** application control scheme is presented in the figure below. As one can see, the scheme is based on the measure of two phase's currents and of the motor position. The Slip Compensation block is used to estimate the stator field position.

The speed is computed as an encoder position difference over one sampling period of the speed control loop. The measured phase currents, i_a and i_b , are transformed into the stator reference frame components, i_{α} and i_{β} . Then, based on the field position information, these components are transformed into the rotor frame direct and quadrature components, i_d and i_q .

The speed and current controllers are **PI** discrete controllers. The inverse coordinates transformation is used for computation of the phase voltages references, u_a_{ref} , u_b_{ref} and u_c_{ref} , applied to the inverter, starting from the values of voltage references computed in the d and q reference frame (u_d_{ref} , u_q_{ref}). Thus, the 6 full compare PWM outputs of the DSP controller are directly driven by the program, based on these reference voltages.



IMVC application control scheme

Based on this application, representing a complete, ready-to-run motion example, the user gets all the information needed as a convenient starting point for the development of his own applications.

Using the advanced features of **DMCD-Pro**, the **motion reference** can be defined at high-level, from the Windows environment. Calling the **data logger** function allows the user to visualize any of the global variables of the program, and effectively analyze and debug his application.



TECHNOSOFT

Induction Motor V/F motion demo application

The Induction Motor V/f Control (**IMVF**) motion application implements an open loop V/f control technique to drive the three-phase induction motor (Sieber motor) with ACPM750.

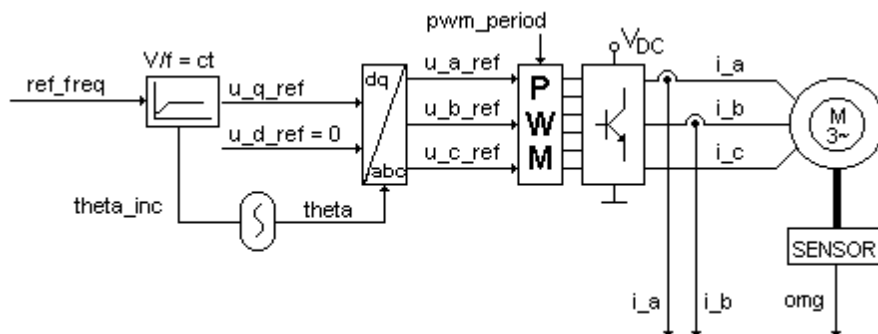
The demo is supplied as a TMS320F24xx application, structured as a project of the **DMCD-Pro** platform. The source files of the application kernel, where from the library function calls are performed, are included in the project structure.

This IMVF motion application uses a constant ratio V/f technique, in order to obtain a constant maximum torque of the motor, over all the operating frequency range.

Basic structure of the control scheme for the IMVF application

The **IMVF** application control scheme is presented in the figure below. As one can see, the control scheme is based on the constant flux level. In order to operate at a constant flux level, the two controllable parameters, stator supply voltage and frequency have to be adjusted for each operating condition. The relation between voltage and frequency is linear except at low speeds.

Since the frequency is closely related to the shaft speed and is imposed by the motor speed requirements, the only parameter independently available for flux regulation is the stator voltage.



IMVF application control scheme

Based on this application, representing a complete, ready-to-run motion example, the user gets all the information needed as a convenient starting point for the development of his own applications.

Using the advanced features of **DMCD-Pro**, the **motion reference** can be defined at high-level, from the Windows environment. Calling the **data logger** function allows the user to visualize any of the global variables of the program, and effectively analyze and debug his application.



TECHNOSOFT

Functions used in the DMCode-L

No.	Function description	Function name
1	Initialization routine for the PWM unit	<i>init_pwm_unit(...)</i>
2	Initialization routine, which configures the PWM unit registers and starts the PWM counter	<i>init_pwm_registers(...)</i>
3	Initialization routine, which enables PDPINT and sets its interrupt vector	<i>init_pdpint()</i>
4	Initialization routine for GPT2 and QEP to be used for encoder reading and enables QEP capture	<i>init_qep()</i>
5	Initialization routine for the Hall module	<i>init_hall_0(...)</i>
6	Initialization routine for the ADC module, which configures AD registers for working in pooling mode	<i>init_adc()</i>
7	Initialization routine for the ADC module, which configures AD registers for working in interrupt mode	<i>init_adc_int(...)</i>
8	Routine which sets GPT2 counter register with the argument	<i>set_qep_timer(...)</i>
9	Initialization routine for the KERNEL module (RTC interrupts).	<i>init_rtc_kernel(...)</i>
10	Initialization routine for a PI structure of the PI controller	<i>init_RegPI(...)</i>
11	Initialization routine for a PI_b structure of the PI controller with set-point weighting for reference	<i>init_RegPI_b(...)</i>
12	Initialization routine for a PID16 structure of the 16-bit PID controller with set-point weighting for reference	<i>init_RegPID16(...)</i>
13	Initialization routine for a PID32 structure of the 32-bit PID controller with set-point weighting for reference	<i>init_RegPID32(...)</i>
14	Initialization routine for the field position variables	<i>init_field_pos()</i>
15	Initialization routine for the 32-bit Reference Generator (computes the Increment Reference table for 32-bit reference initialization)	<i>init_reference32(...)</i>
16	Initialization routine for the 16-bit Reference Generator	<i>init_reference()</i>
17	Initialization routine for the LOGGER module	<i>init_logger()</i>
18	Initialization routine for the POSITION_2_THETA module	<i>init_pos2theta(...)</i>
19	Function which loads 12 saturation values into the data memory	<i>loadsatvals()</i>
20	Real-time Interrupt routine that handles the Kernel interrupts	<i>rtc_kernel</i>
21	Interrupt routine executed at EOC of ADC. Reads the conversion results	<i>adc_rti</i>
22	Interrupt routine executed at the PDPINT interrupt	<i>pdpint_rti</i>
23	Function which reads the position from encoder through QEP interface	<i>read_qep()</i>
24	Function which reads the position from Hall sensors	<i>read_hall()</i>
25	Function which reads the values of the first AD pair (currents <i>_i_a</i> , <i>_i_b</i>)	<i>get_adc_pair1()</i>
26	Function which reads the ADC measures	<i>get_adc(...)</i>
27	Function which sets a new channel for the next measurement	<i>set_ADC_channels(...)</i>
28	Function which starts a new conversion	<i>start_ADC()</i>
29	Function which tests the EOC, reads and returns the first ADC measured variable	<i>get_adc_var1()</i>
30	Function which reads and returns the second ADC measured variable	<i>get_adc_var2()</i>
31	Function which immediately starts the ADC conversion	<i>start_first_ADC_conversion()</i>
32	Function which waits for the EOC and calls the AD interrupt	<i>wait_until_first_EOC()</i>
33	Control function for the AC control of the PWM module	<i>update_pwm_AC(...)</i>
34	Control function for the DC control of the PWM module	<i>update_pwm_DC(...)</i>
35	Control function for the Space Vector control of the PWM module	<i>update_pwm_SVEC(...)</i>
36	Functions for computing sin(theta) and cos(theta) based on a 100-point sine table	<i>sinus_0(...), cosine_0(...)</i>
37	Functions for computing sin(theta) and cos(theta) based on a 128-point sine table	<i>sinus_1(...), cosine_1(...)</i>
38	Functions for computing sin(theta) and cos(theta) based on a 256-point sine table	<i>sinus_2(...), cosine_2(...)</i>
39	Functions for computing sin(theta) and cos(theta) based on a 1024-	<i>sinus_4(...), cosine_4(...)</i>
40	Function which computes the electrical-degree position based on the position read from the encoder	<i>pos2theta(...)</i>



TECHNOSOFT

No.	Function description	Function name
41	Function that computes the position of the rotating field for Induction Motor Slip Control applications	<i>IMSCFldPos(...)</i>
42	Function that computes the position of the rotating field for Induction Motor V/f Control applications;	<i>IMVFFldPos(...)</i>
43	Transformation routine of coordinates from abc to dq frame. Returns the transformed currents i_d, i_q	<i>t_abc_dq(...)</i>
44	Transformation routine of coordinates from dq to abc frame. Returns the reference voltages in the natural frame of the motor (u_a_ref, u_b_ref, u_c_ref)	<i>t_dq_abc(...)</i>
45	Function which transforms coordinates from abc frame to DC (computes DC current)	<i>t_abc_dc(...)</i>
46	The function implements the voltage comutation based on the Hall sensors information	<i>t_dc_abc()</i>
47	Implementation routine for the PI current controller (d & q axis)	<i>RegPI(...)</i>
48	Function that computes _u_q_ref voltage for Induction Motor applications that do not require current	<i>compute_uqref(...)</i>
49	Function that computes the rotor field position	<i>update_field_pos(...)</i>
50	Function that reads the encoder position through QEP interface	<i>read_qep()</i>
51	Speed measuring function using a tachometer	<i>get_omg_tacho(...)</i>
52	Function that implements an analog reference	<i>RefAnalog(...)</i>
53	Implementation routine for the PI controller with set-point weighting for reference (for speed or position)	<i>RegPI_b(...)</i>
54	Implementation routine for the 16-bit PID controller with set-point weighting for reference (for speed or position)	<i>RegPID16(...)</i>
55	Implementation routine for the 32-bit PID controller with set-point weighting for reference (for position)	<i>RegPID32(...)</i>
56	Flux controller implementation routine (for Induction Motor only)	<i>FluxCtrl(...)</i>
57	16-bit Reference generator routine	<i>reference()</i>
58	32-bit Reference generator routine used to the position loop when a 32-bit positioning is needed	<i>reference32()</i>
59	Logger routine which performs data logging	<i>logger()</i>
60	Function which sets 0 wait states	<i>wait_state()</i>
61	Function for grounding protection	<i>GND_protection(...)</i>
62	Function that reactivates PDPINT handling	<i>reactivate_pdpint()</i>
63	Function that reads the <i>pdpint_event</i> parameter (PDPINT counter).	<i>read_pdpint_event()</i>
64	Function that computes the AD channel offsets	<i>detect_offset_gain(...)</i>
65	Function that performs the initial positioning to phase A	<i>positioning_to_phase_A(...)</i>
66	Function that enables the PWM output	<i>enable_pwm_output()</i>
67	Function that disables the PWM output	<i>disable_pwm_output()</i>
68	Function that computes arctangent based on a table	<i>arctan(...)</i>
69	Function that computes the product of two numbers	<i>product(...)</i>
70	Function that computes the square root of an argument	<i>sqrt_ths(...)</i>