TECHNOSOFT

Technosoft is a Third Party of Texas Instruments supporting the TMS320C24xx and TMS320F281x DSP controllers from the C2000 family.

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

DMCD-Pro (Digital Motion Control Developer Pro)

Digital Motion Control Developer for integrated DSP software development for the TMS320F24xx and TMS320F28xx

- 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
- 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

DMCode S(BL) - Permanent Magnet Synchronous Motor motion application

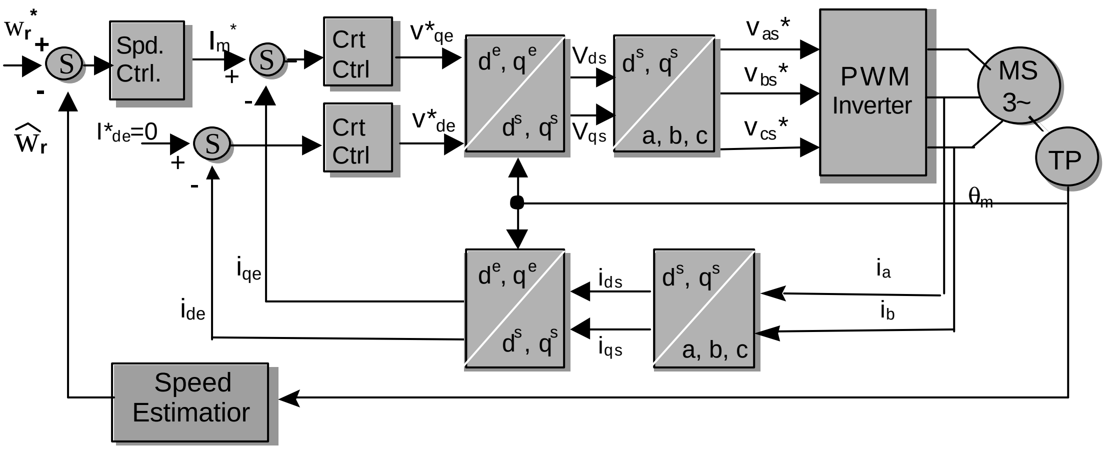
The permanent magnet synchronous motor (PMSM) motion application implements a vector control method to drive in sinusoidal mode the three-phase brushless motor included in the **MCK24xx** or **MCK28xx** kits.

The demo is supplied as a TMS320F24xx or TMS320F28xx application, structured as a project of the **DMCD-Pro** platform. The complete source files of the application 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, $\mathbf{i_a}$ and $\mathbf{i_b}$, are transformed into the stator reference frame components, $\mathbf{i_{ds}}$ and $\mathbf{i_{qs}}$. Then, based on the position information, these components are transformed into the rotor frame direct and quadrature components, $\mathbf{i_{de}}$ and $\mathbf{i_{qe}}$. The speed and current controllers are **PI** discrete controllers. The inverse coordinates transformation is used for computation of the phase voltages references, $\mathbf{v^*_{as}}$, $\mathbf{v^*_{bs}}$ and $\mathbf{v^*_{cs}}$, applied to the inverter, starting from the values of voltage references computed in the d and q reference frame ($\mathbf{v^*_{de}}$, $\mathbf{v^*_{qe}}$). 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 $\mathbf{i^*_{de}}$ 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 to understand its basic DSP implementation aspects, as well as a convenient starting point for the development of his own applications.

The code is developed both in C language – the C28x library, and in C language (the main structure of the application) and assembler (the time-critical parts, as controllers, coordinates transformations, etc.) – the C24xx library.

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 S.A. Rue des Courtils 8A, CH-2035 Corcelles, Switzerland
Phone: +41 32 732 5500 Fax: +41 32 732 5504 e-mail : contact@technosoft.ch WEB :www.technosoft.ch

TECHNOSOFT

DMCode S(BL) - Brushless DC motion demo application

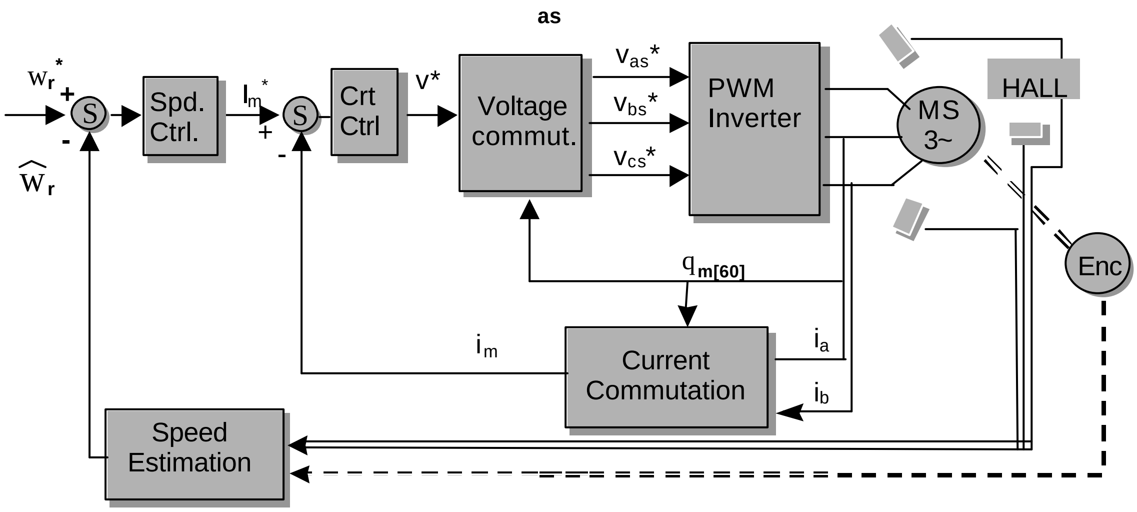
The brushless DC motor (BLDC) motion application implements a block commutation control method to drive the three-phase brushless motor included in the **MCK24xx** kit or **MCK28xx** kits.

The demo is supplied as a TMS320F24xx or TMS320F28xx application, structured as a project of the **DMCD-Pro** platform. The complete source files of the application 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, $\mathbf{i_a}$ and $\mathbf{i_b}$, 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, $\mathbf{v^*_{as}}$, $\mathbf{v^*_{bs}}$ and $\mathbf{v^*_{cs}}$, 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 to understand its basic DSP implementation aspects, as well as a convenient starting point for the development of his own applications.

The code is developed both in C language – the C28x library, and in C language (the main structure of the application) and assembler (the time-critical parts, as controllers, coordinates transformations, etc.) – the C24xx library.

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 S.A. Rue des Courtils 8A, CH-2035 Corcelles, Switzerland
Phone: +41 32 732 5500 Fax: +41 32 732 5504 e-mail : contact@technosoft.ch WEB :www.technosoft.ch

TECHNOSOFT

Crt.No.	Function description	Function name
1	Application program which performs initialization, activates interrupts and waits in an infinite loop	*main()*
2	Speed loop control RTI routine for speed control implementation	*rtc_ps_int() / rtc_slow_int()*
3	Current loop control RTI routine for current control implementation	*rtc_crt_int() / rtc_fast_int()*
4	Initialization routine for the I/O registers shared by several initialization functions	*init_IO_registers()*
5	Initialization routine for the parameters of the d axis current controller.	*init_reg_id()*
6	Initialization routine for the parameters of the q axis current controller.	*init_reg_iq()*
7	Initialization routine for the parameters of the speed controller.	*init_reg_omg()*
8	Initialization routine for the parameters of the encoder interface.	*init_encoder()*
9	Initialization routine for setup the I/O pins of port A (pins1,2,3) as inputs, for reading the HALL sensors connected to them	*init_hall()*
10	Initialization routine for the parameters of the PWM module	*init_pwm()*
11	Initialization routine for the parameters of ADC currents measurement	*init_adc()*
12	Initialization routine for setting of slow sampling interrupt parameters	*init_ctr_ps()*
13	Initialization routine for setting of fast sampling interrupt parameters	*init_ctr_crt()*
14	Initialization routine for the interrupts Kernel	*initializeKernel()*
15	Function for offset detection of the two current measurement channels	*get_ia_ib_offsets()*
16	Initialization routine for the data logger parameters	*init_logger()*
17	Logger routine which performs data logging	*logger()*
18	Initialization routine for the reference generator parameters	*init_reference()*
19	Reference generator routine	*reference()*
20	Initialization routine for the current d-axis PI controller variables	*init_pi_reg_id()*
21	Function for d-axis current PI controller implementation	*pi_reg_id()*
22	Initialization routine for the current q-axis PI controller variables	*init_pi_reg_iq()*
23	Function for q-axis current PI controller implementation	*pi_reg_iq()*
24	Initialization routine for the speed PI controller variables	*init_pi_reg_omg()*
25	Function for speed PI controller implementation	*pi_reg_omg()*
26	Function for enable the QEP circuit for the encoder reading	*start_encoder()*
27	Function which reads and stores the encoder position (QEP capture pulses)	*read_encoder()*
28	Electrical angle computing routine	*enc2theta()*
29	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)	*tdqabc()*
30	Transformation routine of coordinates from abc to dq frame. Returns the transformed currents i_d, i_q and also computes the sine and cosine of theta	*tabcdq()*
31	Routine which updates the PWM signals (AC mode) by updating of the compare registers of the full compare unit	*update_ac_pwm()*
32	Routine which updates the PWM signals (DC mode) by updating of the compare registers of the full compare unit	*update_dc_pwm()*
33	Function which enables the PWM signals generation	*start_pwm()*
34	Interrupt routine executed at EOC of ADC. Reads the conversion results	*read_int_adc()*
35	Function which reads the ADC conversion results in pooling mode	*get_adc_pair1()*
36	Function which enables GPT2 compare interrupt for current loop control	*start_ctr_crt()*
37	Function which enables GPT2 period interrupt for speed loop control	*start_ctr_ps()*
38	Current control interrupt routine executed at GPT2 timer compare event	*rtc_crt()*
39	Speed control interrupt routine executed at GPT2 time period event	*rtc_ps()*
40	Real-time control interrupt routine executed at PWM timer underflow event	*ISR_Kernel()*
41	Function for saturation level computing	*loadsatvals()*
42	Function which saturates the PWM reference voltages	*sat_pwm_voltages()*
43	Function which saturates DC reference voltage (q-axis reference voltage)	*sat_dc_voltage()*

44	Function which computes the sinus of the position angle	*sine()*
45	Function for reading the Hall sensors of the Pittman motor	*read_hall_Pittman()*
46	Function for reading the Hall sensors of the Escap motor	*read_hall_Escap()*
47	Initialization routine for SCSR register to ADC & EVM clock enable	*Initialize_SCSR()*