T E C H N O S O F T

DMCode-S(IM) plug-in for DMCD-Pro

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

To help you get your project started rapidly, Technosoft offers **DMCode-S(IM),** a plug-in for DMCD-Pro, a complete source code library for digital motion control and demo application code for the speed control of a 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 TMS320F24x 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 you to use external module functions into your 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, then download it and execute it automatically on the DSP board

TECHNOSOFT

DMCode-S(IM). 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 the ACPM750 inverter, or the IMDM15 intelligent amplifier.

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.

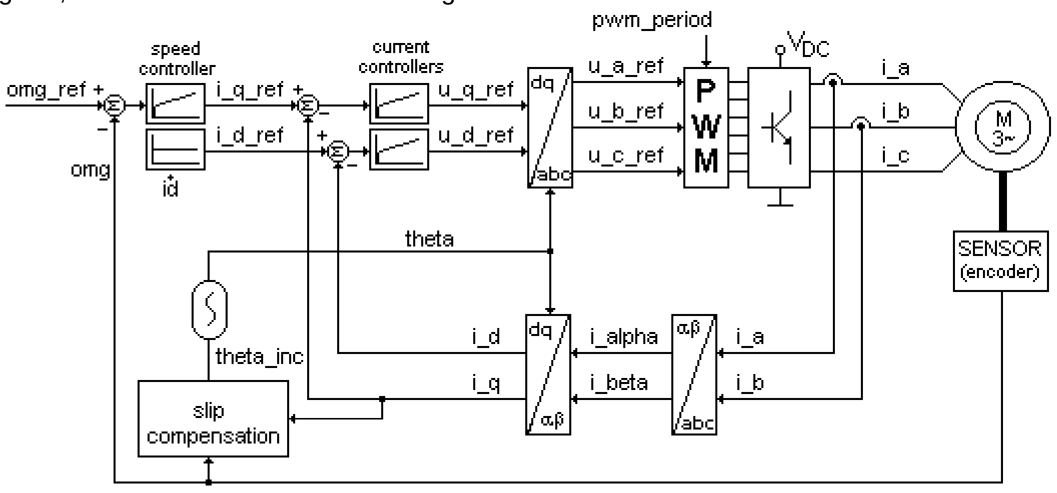
The application is the speed control of an induction motor.

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 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_alpha** and **i_beta**. Then, based on the field position information, these components are transformed into the direct and quadrature rotor frame components, **i_d** and **i_q**.

The speed and current controllers are **PI** discrete controllers. The inverse coordinates transformation is used for the computation of phase voltage 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, you get all the information needed to understand its basic DSP implementation aspects, as well as a convenient starting point for the development of your own applications.

The code is developed either in C language – the C28x library, or 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 a Windows environment. Calling the **data logger** function allows you to visualize any of the global variables of the program, and effectively analyze and debug your application.

T E C H N O S O F T

DMCode-S(IM). 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 the ACPM750 inverter, or the IMDM15 intelligent amplifier.

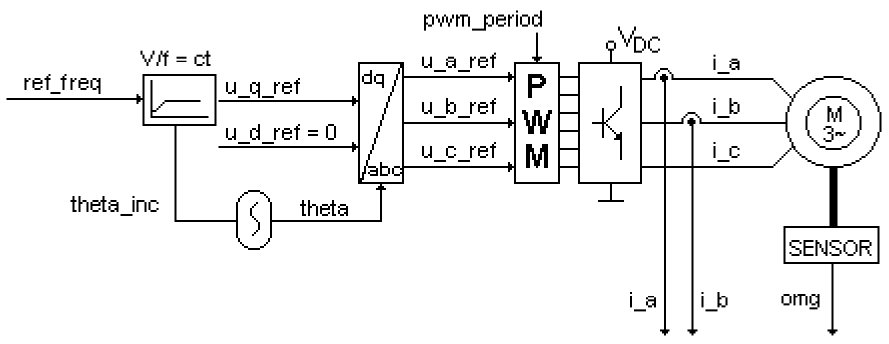
The demo is supplied as a TMS320F24xx or TMS320F28xx application, structured as a project of the **DMCD-Pro** platform. The complete source files of the application are included in the project structure.

This IMVF motion application uses a constant ratio V/f technique, in order to obtain a constant maximum torque of the motor, over the entire 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 independent parameter available for flux regulation is the stator voltage.



IMVF application control scheme

Based on this application representing a complete, ready-to-run motion example, you get all the information needed to understand its basic DSP implementation aspects, as well as a convenient starting point for the development of your own applications.

The code is developed either in C language – the C28x library, or 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 a Windows environment. Calling the **data logger** function allows you to visualize any of the global variables of the program, and effectively analyze and debug your application.

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 the parameters for the rotor field position estimation	*init_field_pos()*
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	Function which gets the low and high parts of the position reference	*ref_pos()*
21	Function which updates the reference	*update_ref()*
22	Function which computes scaled frequency reference	*frequency_ref ()*
23	Initialization routine for the current d-axis PI controller variables	*init_pi_reg_id()*
24	Function for d-axis current PI controller implementation	*pi_reg_id()*
25	Initialization routine for the current q-axis PI controller variables	*init_pi_reg_iq()*
26	Function for q-axis current PI controller implementation	*pi_reg_iq()*
27	Initialization routine for the speed PI controller variables	*init_pi_reg_omg()*
28	Function for speed PI controller implementation	*pi_reg_omg()*
29	Function for enable the QEP circuit for the encoder reading	*start_encoder()*
30	Function which reads and stores the encoder position (QEP capture pulses)	*read_encoder()*
31	Function which resets the ACPM error	*reset_Error_ACPM()*
32	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()*
33	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()*
34	Routine which updates the PWM signals (AC mode) by updating of the compare registers of the full compare unit	*update_ac_pwm()*
35	Function for estimating and updating the rotor field position	*update_field_pos()*
36	Initialization routine for V/f parameters	*init_Vf_params()*
37	Function which computes the voltage, speed and position factors.	*compute_Vf_factors()*
38	Function which enables the PWM signals generation	*start_pwm()*
39	Interrupt routine executed at EOC of ADC. Reads the conversion results	*read_int_adc()*
40	Function which reads the ADC conversion results in pooling mode	*get_adc_pair1()*
41	Function which enables GPT2 compare interrupt for current loop control	*start_ctr_crt()*
42	Function which enables GPT2 period interrupt for speed loop control	*start_ctr_ps()*
43	Current control interrupt routine executed at GPT2 timer compare event	*rtc_crt()*
44	Speed control interrupt routine executed at GPT2 time period event	*rtc_ps()*
45	Real-time control interrupt routine executed at PWM timer underflow event	*ISR_Kernel()*
46	Function for saturation level computing	*loadsatvals()*
47	Function which configures IOPB1 as digital line (output pin)	*cfgiopb1()*
48	Function which sets IOPB1 output pin	*setiopb1()*
49	Function which resets IOPB1 output pin	*resetiopb1()*
50	Function which computes the sinus of the position angle	*sine()*
51	Function which multiplies two numbers	*product()*
52	Function which sets 0 wait states	*wait_state()*
53	Initialization routine for SCSR register to ADC & EVM clock enable	*Initialize_SCSR()*