SWITCHERGEAR
GEAR TO CONTROL YOUR POWER SWITCHES

Example BasicDriveVF

SwitcherGear Example Project

APPLICATION NOTE

Introduction

The BasicDriveVF example project shows how to use the SwitcherGear controller to drive a three-phase induction motor with a PWM-fed IGBT voltage source inverter. The motor control is based on the constant voltage/frequency control principle.

The project code resources and this manual make it easy to build the power converter system with SwitcherGear. The project can be used for demonstration or as a starting point for a more advanced project.

This application note covers:

- Description of power system and SwitcherGear controller.
- Structure and operation of the control firmware.
- Installation of code development tools.
- Configuring the SwitcherWare Library for use with controller.
- Build code and upload to controller.
- Operation of 3-phase induction motor using V/F control.
- Visualisation of digital variables in real-time using oscilloscope.
- Capture of digital variables for post-processing.

Support Material

- SwitcherGear Example BasicDriveVF Application Note (this document).
- Project files for Code Composer Studio:
  - SwitcherGear Example BasicDriveVF F28335.
  - SwitcherGear Example BasicDriveVF F28377D.
- SwitcherWare Library installer application, library version 0.7.0 and above.

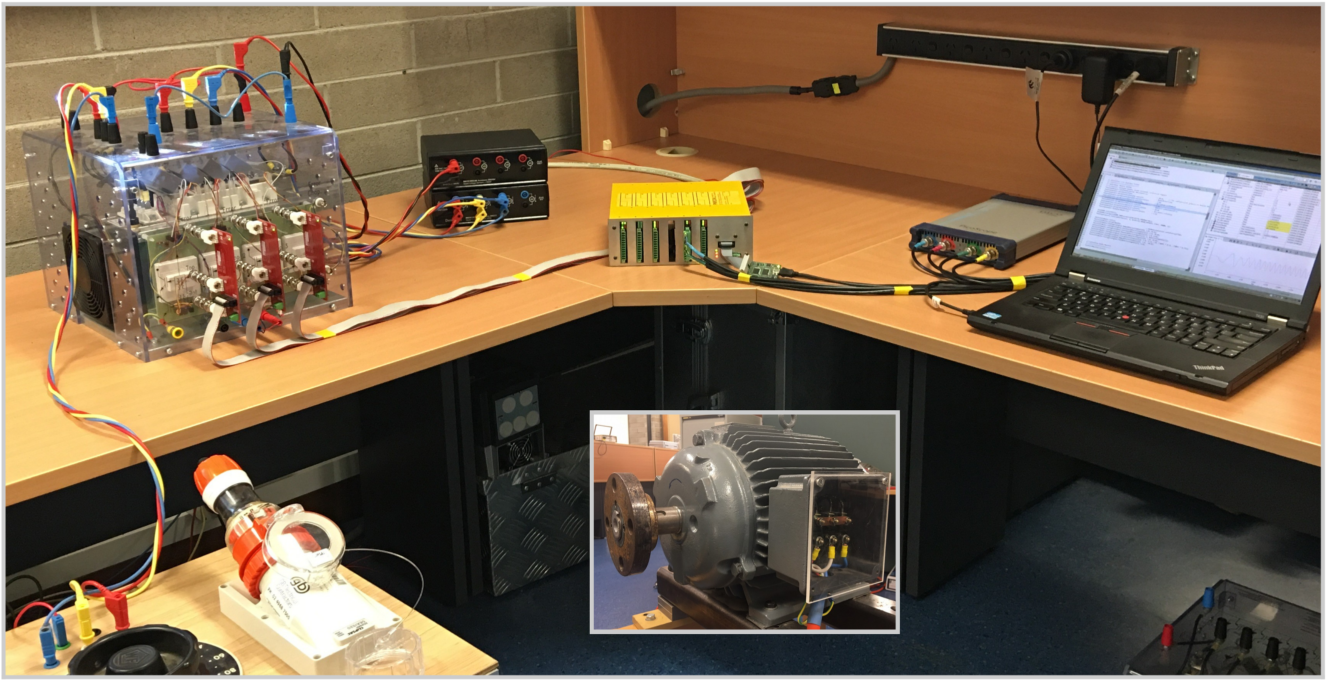
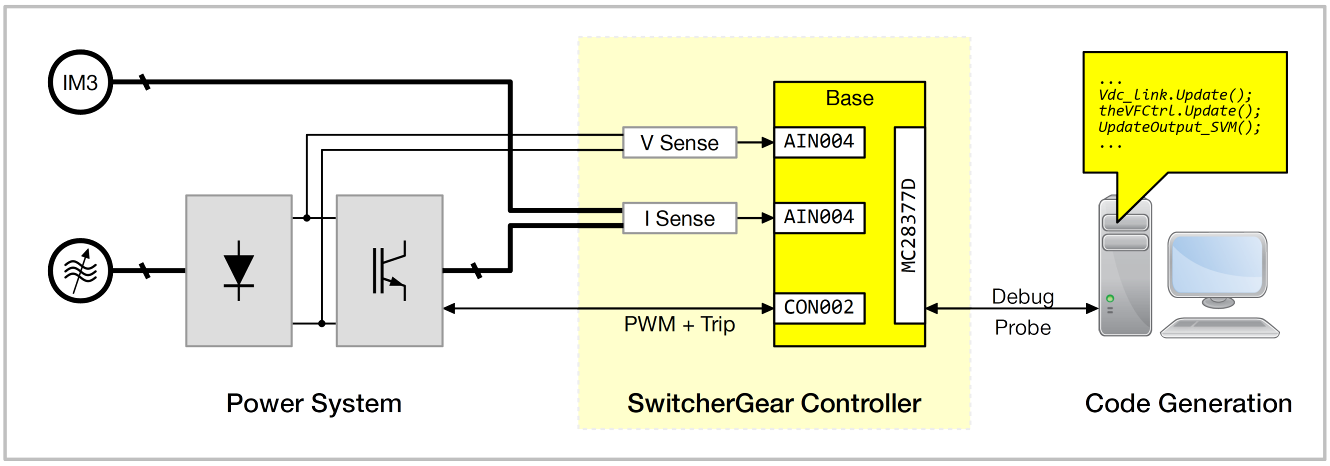


Figure 1: The equipment used for the BasicDriveVF system, clockwise from bottom-left: variable 3-phase mains supply; 3-phase rectifier and voltage source inverter; current and voltage sensors; SwitcherGear controller with XDS100 debug probe; 4-channel oscilloscope for real-time visualisation of digital controller variables; and, laptop running Code Composer Studio development software. The inset shows the load induction motor.

Denkinetic Pty Ltd
contact@denki.com.au



```
...
Vdc_Link.Update();
theVFCtrl.Update();
UpdateOutput_SVM();
...
```

Figure 2: Schematic diagram of the hardware for the BasicDriveVF project.

Hardware

- SwitcherGear controller:
  - Denkinetic Micro MC28377D or F28335 controlCARD.
  - 1x CON002 converter interface module, or equivalent.
  - 1x AIN004 sensor interface modules, or equivalent.
  - 1x AOV001 (or AOU001) analogue output module.
  - 1x XDS110 or XDS100v2 debug probe.
  - 1x 24 VDC power supply.
- SwitcherGear accessories:
  - 3x CBL002, 20-way ribbon cables for CON002.
  - 1x ADP002/3 adaptor kit for SemiTeach IGBT.
  - 3x SNI005, or equivalent, current sensors.
  - 1x SNV005, or equivalent, voltage sensor.
  - 4x CBL003, 3-way cables for sensors.
- Power system:
  - Variable 3-phase mains supply with voltage up to 440 VAC.
  - Semikron SemiTeach IGBT power stage, or equivalent rectifier and inverter.
  - 3-phase induction motor.
  - Hookup leads with shrouded 4 mm connectors.
- 4-channel digital oscilloscope.

Microcontroller (MCU) Variants

This manual is written for the example project as implemented on a SwitcherGear controller with the Denkinetic MC28377D microcontroller card, which uses the Texas Instruments TMS320F28377D microcontroller. Unless otherwise mentioned, references to the F28377D MCU can be replaced with other supported MCU types. Such references may include text descriptions in this manual, or the name of code objects.

Warning

This example project involves the use of hazardous voltages. You should only attempt this project if you are familiar with power systems and the proper safety precautions to be used with power systems.

Project Description

Hardware

The experimental hardware consists of the power system, the SwitcherGear controller and the code generation system.

Power System

The power system is comprised of a Semikron SemiTeach IGBT power stage, a variable voltage 3-phase mains supply and a 3-phase induction motor.

A variable voltage 3-phase supply is used as a simple way to limit the charging current of the DC link capacitors. The voltage should be set to zero before turning on the supply.

SwitcherGear Controller

The SwitcherGear controller consists of a SwitcherGear base unit with a microcontroller (MCU) module and three hardware interface modules.

The PWM outputs of the MCU are routed internally to the converter interface module. The signals are fed through line drivers to ensure clean signals reach the IGBT gate drivers in the SemiTeach power stage. The module processes the fault signals from the SemiTeach power stage and routes them back to the MCU's trip input. It also provides 15 V power to the gate drivers.

Separate voltage and current sensor accessories are used to safely measure the DC link voltage and motor phase currents. The outputs from the sensors are connected to a sensor input module that performs the signal conditioning. The output signals of the module are routed internally to the high-speed ADCs in the MCU.

An analogue output module is used to convert digital controller variables in the MCU into analogue ±10 V signals that can be visualised in real-time with an oscilloscope.

SwitcherGear Configuration Document

This document is used to define low-level code objects based on the physical configuration of the controller. The document captures the configuration of the controller by listing the hardware interface modules that are installed in the base connector slots, and the MCU pins to which the modules' signals connect.

Each MCU pin can be assigned a digital or analogue or SPI code object. These objects can be passed to higher-level SwitcherWare Library objects and they provide a simple way to use I/O pins in your own application code.

The Configuration Document includes scripting that auto-generates code source files based on the physical configuration. They can be found in the example project **`SwitcherGear Configuration`** folder. The source files are included into the project and are linked with the SwitchWare Library and the project code.

SwitcherWare Library Objects

To simplify code development, the BasicDriveVF project uses the SwitcherWare Library from Denkinetic. This library provides code resources that configure the MCU and implement standard functionality used in power converters, including conversion of analogue sensor signals, PWM generation, coordinate transformations, etc.

The SwitcherWare Library and BasicDriveVF project are implemented in embedded C++, which provides object-oriented features while maintaining the familiarity and computational speed of C.

Objects that are used throughout the project (global objects) are defined in the source file **`AppObjects.cpp`**.

Sensor Inputs

The four objects **`Iu`**, **`Iv`**, **`Iw`** and **`Vdc_link`** are instances of the **`AinPinScaled`** class that is a code wrapper for analogue input pins. Their definition includes the number of the input pin and the physical range of the signal being measured – this information comes from the Configuration Document auto-generated source files. During the application initialisation, the objects are registered with the **`theF2837xD_ADC`** object, which handles the configuration of the low-level MCU registers and conversion sequencer. At each execution of the control algorithm, a call to the **`Update()`** function of each object retrieves the result of the ADC conversion and applies the required scaling. The scaled result is available in the **`outScaled`** member variable of the object, e.g. **`Iu.outScaled`**.

Converter Interface

The object **`theConverter`** is an instance of the **`HB3Sym_F2837xD`** class that is a code wrapper for interfacing to 3-phase, 2-level converters. It uses the EPWM peripheral hardware in the F28377D to generate gate signals according to a variety of methods, including space vector modulation (SVM) and sinusoidal PWM. The input variables used to calculate duty cycle values are the demand voltage space vector **`theConverter.inVdemandS3`** and the measured DC link voltage **`theConverter.inVdclink`**. It also reacts to gate driver fault signals by instantly disabling the gate signals.

In this project, the PWM is symmetric – generated by a timebase up-down counter running at a period of 5 kHz (black triangle wave in Figure 3). The transitions of the gate PWM signal occur when the value of the timebase counter equals the value of the compare register, which is a low-level register representation of duty cycle. The compare register can be set differently for the count up and count down sections of the timebase. This allows independent modulation of the turn-on and turn-off gate edges and results in a control rate that is double the PWM switching frequency, that is 10 kHz. This double rate control is configured by the use of the constant **`ControlRate_Double`** in the call to **`theConverter.Init()`**.

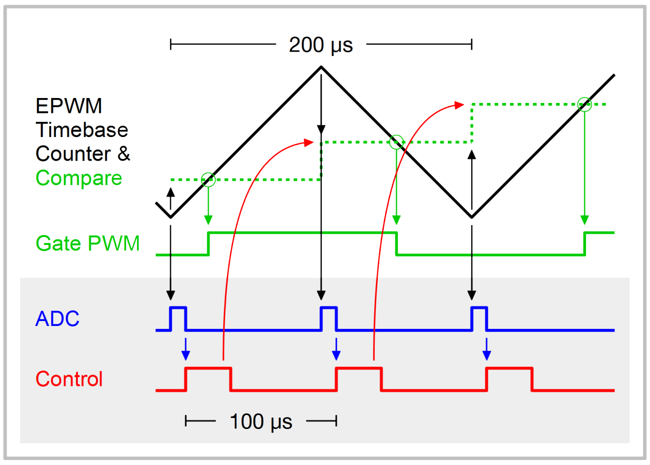


Figure 3: Generation of symmetric PWM with independent control of gate PWM turn-on and turn-off. The ADC start-of-conversion is synchronised to the PWM timebase and occurs twice per PWM cycle.

To take advantage of the doubled PWM update rate, the control algorithm must be executed at double-rate to calculate a new duty cycle value for each of the two PWM edges in each switching cycle. In turn, this requires that the ADC must capture new measurements of voltage and current at double-rate.

The PWM peripheral generates a hardware ADC start-of-conversion signal at the timebase's zero and period instants (black arrows in Figure 3) that triggers the ADC conversion of the analogue voltage and current sensor signals. In turn, the completion of the ADC conversion sequence triggers an interrupt to run the control algorithm (blue arrows). The control algorithm calculates the new duty cycles for each phase and updates the compare registers (red arrows). The new compare values take effect in the next PWM half-cycle (black arrows).

The various events, triggers and interrupts are configured by the SwitcherWare Library resources at initialisation time of the application. The real-time orchestration of them at run time is handled by the MCU hardware without any intervention from user code.

Other PWM and control schemes are easily achieved, but are beyond the scope of this manual.

Constant V/F Controller

The object **`theVFCtrl`** is an instance of the **`VFControl`** class that implements a constant voltage-frequency controller for induction machines. It has parameter variables for the voltage-frequency constant and the boost voltage. These parameters and the demanded angular frequency are used to calculate the magnitude and angular velocity of the output rotating voltage space vector. This voltage space vector is passed as the demand input to the converter object.

Ramp Generator

The object **`theFreqRamp`** is an instance of the **`Ramp`** class that implements a ramp limiter. It has separate parameter variables for the increment and decrement rates. The output value is ramped at these rates towards the target value.

This object is used to rate limit the user demanded value of angular frequency. This prevents discontinuities in the demand value that is sent to the motor control object.

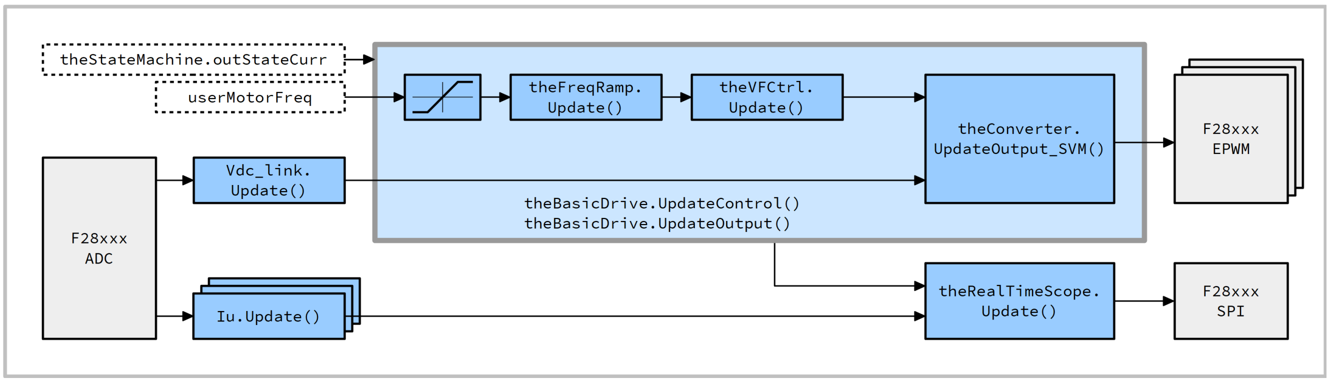


Figure 4: Block diagram for the interrupt service routine **ISR_PWMControl()** that executes the motor control algorithm. Blue blocks represent operations performed by calls to object member functions or C statements. Grey blocks represent operations performed by the MCU hardware peripherals. The blocks with dotted outline represent variables.

Analogue Outputs

The object **theRealTimeScope** is an instance of the **ModuleAOV001** class that is a code wrapper for interfacing to a AOV001 4-channel isolated analogue voltage output module. The object is initialised in **main()**, where, for each output channel, a pointer to the target floating-point variable and scaling parameters are stored. The **Update()** function is called at the control frequency after the motor control is performed. The object sends the current value of the target variables by SPI bus to the DAC on the module.

Project Objects

State Machine

The object **theStateMachine** tracks the basic operational states for the drive – Ready, Run, Fault. Transitions between states are controlled by input commands (Start, Stop, Reset) and flags. This object provides a structured way to start and stop the drive, handle faults and allow extension of features. See Figure 5.

The current state is stored in the member variable **theStateMachine.outStateCurr**. By itself, the value of the current state has no meaning and does not affect the operation of the application. It is up to the developer to read the value of the current state and to modify the operation of the application according to what this state means for the application. For example, when the state is Run, the application should execute the control algorithm and enable PWM to the converter. And when the state is Ready or Fault, the application should reset the controller to suitable initial conditions and disable the PWM.

Similarly, the state machine has no knowledge about when the current state should be changed. It is up to the user to identify when a change of state is required. You should not change the state by modifying **theStateMachine.outStateCurr** directly. Instead, you should send a command to the state machine to request the change of state. You do this by writing the request into the member variable **theStateMachine.inCommand** and then calling the member function **theStateMachine.Update()**. If the request is successful, the value of the current state will be changed and the application will respond according to the new state.

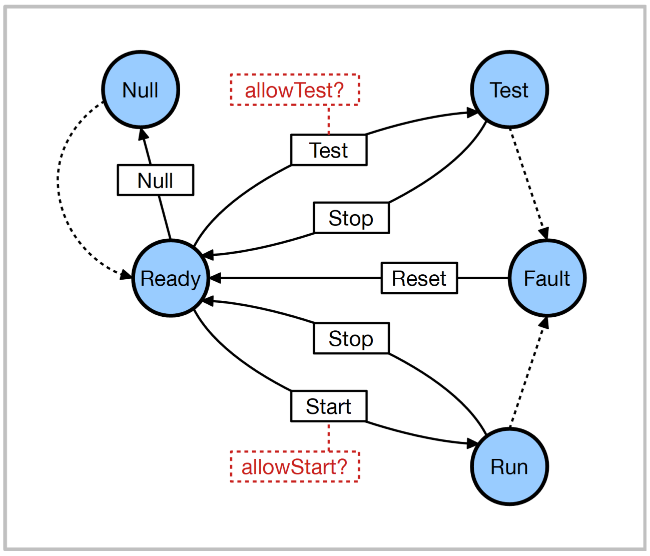


Figure 5: State transition diagram for the BasicDriveVF project. The blue circles are the states and the arrows between them are the state transitions. The commands that request the transitions are labelled on the transition lines. Flags that allow various transitions are shown in the dashed red boxes. The dashed black lines indicate automatic (non-user) transitions.

In the project of this application note, the user interacts with the state machine through the debug environment of CCS. The input command and flags are manually changed by the user to request changes to the state. A background task periodically calls the **Update()** function to process the input command and flag values.

The state transition Ready to Run is requested by using the Start command, but is also controlled by the flag **theStateMachine.inFlags.bit.allowStart**. The flag must be set to allow the transition. In this application, the flag is set at initialisation time, so the transition is allowed every time it is requested. In other applications, the transition can be controlled by setting and clearing the flag according to suitable conditions, e.g. the DC link voltage must be in a certain range, etc.

This project features two extensions to the basic state machine. See the next section Basic Drive for more details.

It adds a Null state, whose intention is to allow the offset in current sensors to be nulled. It can be accessed only from the Ready state and automatically returns to the Ready state when nulling is complete.

It also adds a Test state, whose intention is to allow developers to validate the hardware inputs and outputs of the SwitcherGear controller with the control loops disabled. Each request for entry to the Test state must be allowed by manually setting the flag **`theStateMachine.inFlags.bit.allowTest`**.

Basic Drive

A **`BasicDrive`** object is created in the project to initialise the above objects and implement the V/F drive functionality. The object provides separate functions for reading control inputs, performing the control algorithm and writing the control outputs. The behaviour implemented by these functions depends on the state of the state machine.

When the state of the drive is Run, the user input variable **`userMotorFreq`** is passed through a range limiter and the ramp generator before being passed to the VF control object. This object calculates the magnitude and orientation (i.e. the space vector) of the motor voltage, and passes it to the PWM generator. The state of the state machine is used as an input to the **`BasicDrive`** object to allow or disallow motor control and the generation of PWM signals. See Figure 4 for a block diagram of the control algorithm execution.

When the current state is Null, the application should disable the PWM (to force the currents in the power system to be zero) and null the offset in the current sensors by calling **`Iu.Null(0.0)`**, etc.

When the state is Test, the application reads the controller inputs (current and voltage sensors) and enables the PWM duty calculation and PWM outputs. The control algorithm is not executed. The user can use the debug environment to check the values of the input variables and manually change the controller outputs to validate the effect on the converter switching. The converter should be disconnected from supplies, loads, etc. before entering this state.

For all other states, the object resets the various control objects and disables the PWM peripheral hardware.

Task Scheduling

The operation of the application is achieved using the concept of tasks. Each tasks is executed by the MCU as either a background or a foreground task – see Figure 6. The tasks are implemented as functions and are defined in **`main.cpp`**.

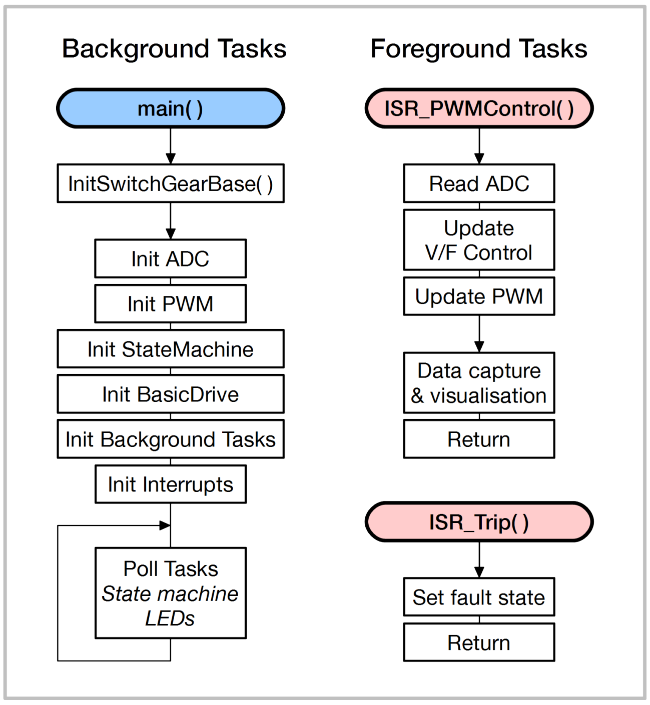


Figure 6: Tasks for the BasicDriveVF project.

Background tasks are low priority tasks and are executed one at a time on a periodic schedule in the **`main()`** context. After initialisation is complete, **`main()`** drops into an infinite loop that polls for pending tasks. The scheduling of tasks is handled by the background task manager, the object **`theBackgroundTaskMgr`**. Background tasks in this project include updating the state machine object, flashing the front panel LEDs, reading the temperature of the SwitcherGear base.

Foreground tasks are critical tasks that must be executed when selected events occur. They are implemented as hardware interrupts. The various SwitcherWare Library objects configure the MCU registers to trigger the interrupts automatically. When events occur, the execution of the current background task is immediately suspended and the interrupt is executed. The background task resumes after the interrupt has completed.

The foreground tasks in this project are the drive control algorithm (see Figure 4) and the trip fault handler. The execution of the drive control interrupt **`ISR_PWMControl()`** is triggered by the completion of the ADC conversion sequence.

The execution of the trip fault interrupt **`ISR_Trip()`** is triggered when the fault signal from any of the half-bridge gate drivers goes active (logic low). The EPWM peripherals are configured to latch all gate signals off when the trip signal is active.

Data Capture

The object **`theDataCapture`** is an instance of the **`DataCapture`** class, which provides a simple method to capture application data. At each capture time, it copies a vector of up to 16 floating-point (**`float`**) variables to a buffer array in memory.

The copying process uses the DMA peripheral of the MCU, so data can be captured at mega-samples per second rates without significantly affecting the execution time and without the limitations of transferring real-time data over a serial communications link. After the data capture is complete, the buffer can be transferred at a slower rate to the PC.

The data can be captured every control cycle or at some other rate that is related to the control process. This greatly simplifies post processing and comparison with simulation results.

If the process to be captured is periodic, the capture can be started manually and left to capture data for as many periods as required. For transient events, the capture can be triggered on the event.

The amount of data that can be captured is limited primarily by the available memory. The MC28377D micro card includes a large external SDRAM memory to allow large data captures.

Code Generation Tools

Code Composer Studio

The Code Composer Studio (CCS) software from Texas Instruments (TI) is required for all installations because it includes the C/C++ compilers for the host MCUs.

CCS also provides:

- an integrated development environment based on Eclipse to edit, build and debug embedded C/C++ applications;
- the GUI Composer tool to build and deploy custom graphical user interfaces.

Download CCS Installer

Download the latest version of Code Composer Studio from the Texas Instruments website.

http://processors.wiki.ti.com/index.php/Download_CCS

Installers are available as web installers or off-line installers.

Install CCS

Execute the installer application and follow the installation instructions.

The typical installation option will install code generation tools for all TI MCUs and multiple debug probes, including those required by SwitcherGear.

To reduce the installation size, you can choose the custom installation option. In this case, you must ensure that the C2000 code generation tools and the drivers for your debug probe are installed.

Licensing

A licence is not required for CCS version 7 and above.

See the CCS licensing web page for more information.

http://processors.wiki.ti.com/index.php/Licensing_-_CCS

C2000Ware

The C2000Ware package must be installed because it provides the device support header files and the C/C++ run-time libraries for the C2000 MCUs. C2000Ware also provides much reference material for the C2000 family of MCUs and includes the datasheets and user guides for all MCUs, application libraries and example projects for MCU core and peripherals.

If C2000Ware is already installed on your PC, it is recommended that you update to the latest version.

Download C2000Ware from the TI website and install on your PC.

http://www.ti.com/tool/c2000ware

Install LibreOffice

The LibreOffice office suite is required to open the SwitcherGear Configuration document.

Download LibreOffice for free from

http://www.libreoffice.org/

Debug Probe

The debug probe is a hardware tool that connects to the JTAG interface of the target MCU. A Texas Instruments XDS110 debug probe is recommended.

The 14-pin connector of the debug probe is connected to the JTAG interface connector on the front panel of the SwitcherGear controller. The JTAG interface is isolated to eliminate ground loops and reduce electromagnetic interference (EMI).

Use a USB cable to connect from the USB port of the debug probe to the laptop or PC that runs the Code Composer Studio software.

Hardware Configuration

Power System

Connect the power system as shown in Figure 7. The connections should be made using insulated test leads that have shrouded 4 mm safety connectors and a suitable current rating. The leads should be twisted into harnesses to minimise the generation of external fields.

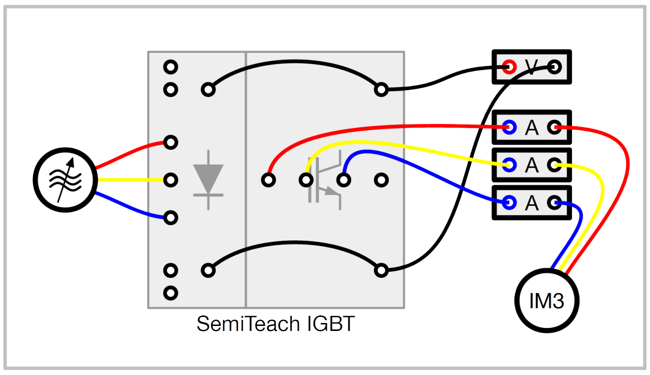


Figure 7: Connections of the power system.

SwitcherGear Controller

Microcontroller

The BasicDriveVF example project is available for two target MCUs: F28335 and F28377D. In this reference manual, the Denkinetic MC28377D microcontroller card [1] is installed in the MCU slot of the SwitcherGear controller base [2] and the corresponding project is used.

Converter Interface

The gate and fault signals of the SemiTeach power stage are brought to BNC connectors on the front face of the unit. A SwitcherGear ADP002 3-phase adaptor kit converts the BNC connections for each half-bridge to a ribbon cable connection.

A 3-phase half-bridge interface module [3] is mounted in slot MRE of the SwitcherGear base.

Use 20-way ribbon cables to connect from the SemiTeach power stage connect to the CON002 module. The connections should be ordered so that half-bridges 1/2/3 of the SemiTeach power stage are connected to the HB1/2/3 half-bridge interfaces of the CON002 module.

The gate drive and fault signals in the ribbon cable all use 15 V logic levels and the ribbon cable carries alternating ground traces. This provides a high degree of protection against EMI.

Current Measurement

The motor currents from half-bridges 1/2/3 of the SemiTeach power stage are connected through three SNI005 current sensors [4]. These current sensors have a transfer gain of 1 mA output current for every 1 A input current.

An AIN004 4-channel sensor input module is mounted in slot MRB of the SwitcherGear base. For this example project, the first three channels of the module are configured for a bipolar input range of ±20 mA – refer to the module reference manual [5] for instructions. This gives a sensor measurement range of ±20 A, which is suitable for measuring AC currents up to 14 A RMS.

Connect the output cable of the first current sensor accessory to pins 1–3 of the system connector of the AIN004 module. Connect the second and third sensors to pins 4–6 and pins 7–9. Refer to the module reference manual for the connector pin-out.

The processed current signals are on pins A0/A1/A2 of the module's MCU interface. Because the module is installed in slot MRB, these connect to pins MRBA0/A1/A2 of the base's module slot connector.

Voltage Measurement

The DC link voltage is connected to a SNV005 voltage sensor [6]. This voltage sensor has a transfer gain of 20 µA output current for every 1 V input voltage.

For this example project, the fourth channel of the sensor input module is configured for a unipolar input range of 0 to 20 mA – refer to the module reference manual [5] for instructions. This gives a sensor measurement range of 0 to 1000 V.

Connect the output cable of the voltage sensor accessory to pins 10–12 of the system connector of the AIN004 module.

The processed voltage signal is on pin A3 of the module's MCU interface. Because the module is installed in slot MRB, this connects to pin MRBA3 of the base's module slot connector.

Analogue Output

An AOV001 4-channel isolated analogue output module [7] is mounted in slot MFB of the SwitcherGear base. (If you are using an AOU001 module, all channels of the module must be configured for a ±10 V output range – refer to the reference [8] manual for instructions.)

Use coaxial cables to connect from the module's system connector to a 4-channel oscilloscope. This is used in this example project to allow signals in the digital domain to be visualised on an oscilloscope.

Install The SwitcherWare Library

Each SwitcherGear controller includes a license for the SwitcherWare Library. Refer to the Install SwitcherWare Application Note [9] for detailed installation instructions.

Documentation for the SwitcherWare Library is located in the **`document`** folder in the install folder.

CCS Workspace

You should build the starter project in a new CCS workspace.

The first time that you open CCS, a new workspace will be created and you will be prompted to choose the file system location.

If you already have an open workspace, you can make a new workspace by selecting from the **File** menu **Switch Workspace > Other...** and browsing for a new file system location.

Set Build Variables

The example project uses path variables to define the locations of the C2000Ware device support resources and the SwitcherWare Library resources. The path variables must be defined in CCS before the project can be built. They can be set manually or imported from a file.

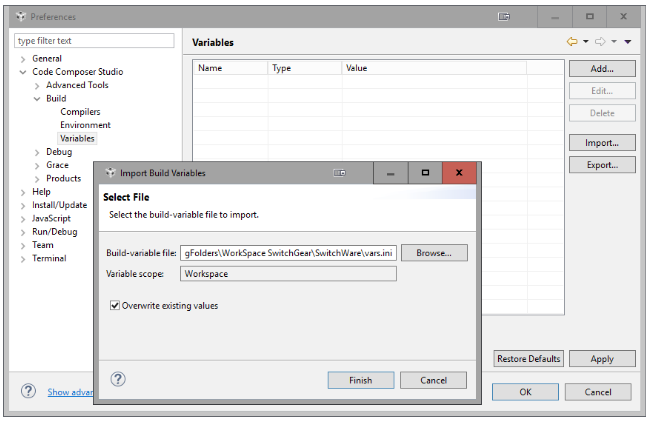
The simplest way is to import the path variables from a file. The SwitcherWare Library install folder contains the file **`vars.ini`** that contains the required path variables.

Before importing the path variables, you must open the **`vars.ini`** file in a text editor (e.g. drag the file into CCS) and ensure that the paths are defined correctly for your system. The path definitions in the file reflect the paths based on standard install locations and versions of the resources when the file was created. You must edit the paths if your code resources are installed in different locations, or the resources are a different version. There are instructions in the file itself.

Then import the path variables into your CCS workspace:

- In the **Windows** menu, select the **Preferences** menu item to show the **Preferences** dialogue box. See Figure 14.
- In the navigation pane, select **Code Composer Studio > Build > Variables** to show the Variables preferences.
- Click on the **Import...** button to show the **Import Build Variables** dialogue box.
- Click on the **Browse...** button, then navigate to and select the **`vars.ini`** file in the SwitcherWare Library install folder.
- Make sure the **Overwrite existing values** check button is checked.
- Click the **Finish** button to import the path variables.

You can also manually setup the path variables – see the Install SwitcherWare Application Note for details.



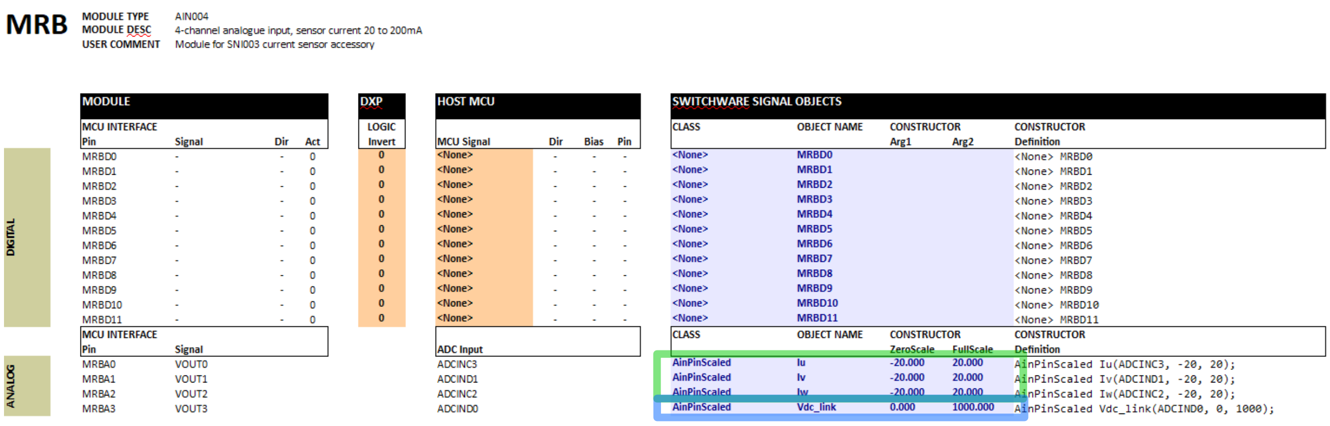
Select the build-variable file to import.

Build-variable file: gFolders\WorkSpace SwitchGear\SwitchWare\vars.ini

Variable scope: Workspace

[✓] Overwrite existing values

Figure 8: Importing build variables into the CCS workspace.



MRB

MODULE TYPE	AIN004
MODULE DESC	4-channel analogue input, sensor current 20 to 200mA
USER COMMENT	Module for SNI003 current sensor accessory

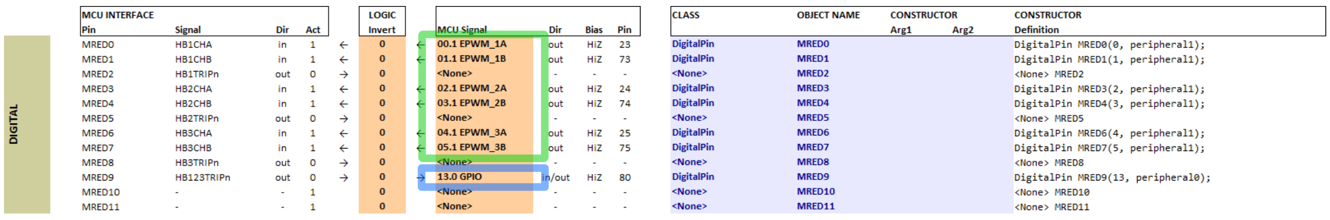
	MODULE				DXP	HOST MCU				SWITCHWARE SIGNAL OBJECTS				
	MCU INTERFACE				LOGIC					CLASS	OBJECT NAME	CONSTRUCTOR		CONSTRUCTOR
	Pin	Signal	Dir	Act	Invert	MCU Signal	Dir	Bias	Pin			Arg1	Arg2	Definition
DIGITAL	MRBD0	-	-	0	0	<None>	-	-	-	<None>	MRBD0			`<None> MRBD0`
	MRBD1	-	-	0	0	<None>	-	-	-	<None>	MRBD1			`<None> MRBD1`
	MRBD2	-	-	0	0	<None>	-	-	-	<None>	MRBD2			`<None> MRBD2`
	MRBD3	-	-	0	0	<None>	-	-	-	<None>	MRBD3			`<None> MRBD3`
	MRBD4	-	-	0	0	<None>	-	-	-	<None>	MRBD4			`<None> MRBD4`
	MRBD5	-	-	0	0	<None>	-	-	-	<None>	MRBD5			`<None> MRBD5`
	MRBD6	-	-	0	0	<None>	-	-	-	<None>	MRBD6			`<None> MRBD6`
	MRBD7	-	-	0	0	<None>	-	-	-	<None>	MRBD7			`<None> MRBD7`
	MRBD8	-	-	0	0	<None>	-	-	-	<None>	MRBD8			`<None> MRBD8`
	MRBD9	-	-	0	0	<None>	-	-	-	<None>	MRBD9			`<None> MRBD9`
	MRBD10	-	-	0	0	<None>	-	-	-	<None>	MRBD10			`<None> MRBD10`
	MRBD11	-	-	0	0	<None>	-	-	-	<None>	MRBD11			`<None> MRBD11`

	MCU INTERFACE			CLASS	OBJECT NAME	CONSTRUCTOR		CONSTRUCTOR
	Pin	Signal	ADC Input			ZeroScale	FullScale	Definition
ANALOG	MRBA0	VOUT0	ADCINC3	AinPinScaled	Iu	-20.000	20.000	`AinPinScaled Iu(ADCINC3, -20, 20);`
	MRBA1	VOUT1	ADCIND1	AinPinScaled	Iv	-20.000	20.000	`AinPinScaled Iv(ADCIND1, -20, 20);`
	MRBA2	VOUT2	ADCINC2	AinPinScaled	Iw	-20.000	20.000	`AinPinScaled Iw(ADCINC2, -20, 20);`
	MRBA3	VOUT3	ADCIND0	AinPinScaled	Vdc_link	0.000	1000.000	`AinPinScaled Vdc_link(ADCIND0, 0, 1000);`

Figure 9: This figure shows the complete Signals entry in the SwitcherGear Configuration document for the MRB module slot, which has an AIN004 module installed in it. It shows the pins of the modules MCU Interface (left side), the pins of the Host MCU (centre) and the SwitcherWare Signal Objects entries. Each row represents the routing of a single signal from the module pin to the MCU pin in the physical domain to the code domain.

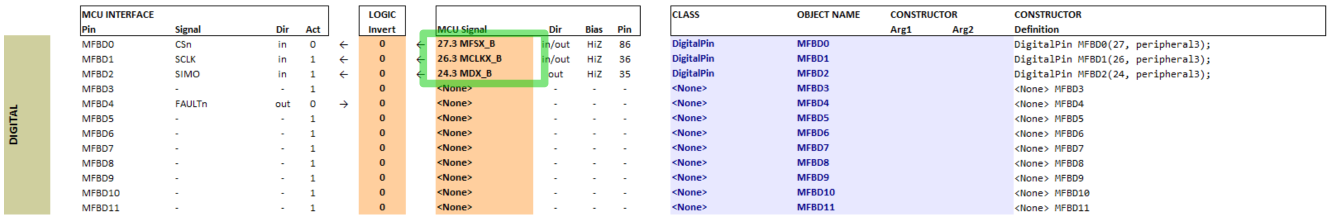
In this example project, the current sensors are wired to Channels 0/1/2 of the module's system connector and the processed signals output the module from the MCU Interface Pins MRBA0/1/2. Similarly, the voltage sensor is connected to Channel 3 of the system connector and is available at MCU Interface Pin MRBA3.

The names of the current sensor objects must be set as shown so that they match the names used in the example project code. (You can use different names if you refactor the object names in the code.) The ZeroScale and FullScale constructor arguments must match the physical measurement range of the sensors, which is determined by the gain of the sensors and the jumper settings of the AIN004 module.



	MCU INTERFACE					LOGIC						CLASS	OBJECT NAME	CONSTRUCTOR		CONSTRUCTOR
	Pin	Signal	Dir	Act		Invert		MCU Signal	Dir	Bias	Pin			Arg1	Arg2	Definition
DIGITAL	MRED0	HB1CHA	in	1	←	0	←	00.1 EPWM_1A	out	HiZ	23	DigitalPin	MRED0			`DigitalPin MRED0(0, peripheral1);`
	MRED1	HB1CHB	in	1	←	0	←	01.1 EPWM_1B	out	HiZ	73	DigitalPin	MRED1			`DigitalPin MRED1(1, peripheral1);`
	MRED2	HB1TRIPn	out	0	→	0		<None>	-	-	-	<None>	MRED2			`<None> MRED2`
	MRED3	HB2CHA	in	1	←	0	←	02.1 EPWM_2A	out	HiZ	24	DigitalPin	MRED3			`DigitalPin MRED3(2, peripheral1);`
	MRED4	HB2CHB	in	1	←	0	←	03.1 EPWM_2B	out	HiZ	74	DigitalPin	MRED4			`DigitalPin MRED4(3, peripheral1);`
	MRED5	HB2TRIPn	out	0	→	0		<None>	-	-	-	<None>	MRED5			`<None> MRED5`
	MRED6	HB3CHA	in	1	←	0	←	04.1 EPWM_3A	out	HiZ	25	DigitalPin	MRED6			`DigitalPin MRED6(4, peripheral1);`
	MRED7	HB3CHB	in	1	←	0	←	05.1 EPWM_3B	out	HiZ	75	DigitalPin	MRED7			`DigitalPin MRED7(5, peripheral1);`
	MRED8	HB3TRIPn	out	0	→	0		<None>	-	-	-	<None>	MRED8			`<None> MRED8`
	MRED9	HB123TRIPn	out	0	→	0	→	13.0 GPIO	in/out	HiZ	80	DigitalPin	MRED9			`DigitalPin MRED9(13, peripheral0);`
	MRED10	-	-	1		0		<None>	-	-	-	<None>	MRED10			`<None> MRED10`
	MRED11	-	-	1		0		<None>	-	-	-	<None>	MRED11			`<None> MRED11`

Figure 10: The digital MCU signal allocations (orange cells) for the CON002 3-phase converter interface module. In this configuration, the HBxCHA/B signals of the CON002 module are mapped to the A and B outputs of the EPWM1/2/3 peripherals (green outline) – the base peripheral is EPWM1. The combined trip signal is routed to GPIO13 (blue outline). The converter and ADC wrapper objects must be initialised with this physical configuration data.



	MCU INTERFACE					LOGIC						CLASS	OBJECT NAME	CONSTRUCTOR		CONSTRUCTOR
	Pin	Signal	Dir	Act		Invert		MCU Signal	Dir	Bias	Pin			Arg1	Arg2	Definition
DIGITAL	MFBD0	CSn	in	0	←	0	←	27.3 MFSX_B	in/out	HiZ	86	DigitalPin	MFBD0			`DigitalPin MFBD0(27, peripheral3);`
	MFBD1	SCLK	in	1	←	0	←	26.3 MCLKX_B	in/out	HiZ	36	DigitalPin	MFBD1			`DigitalPin MFBD1(26, peripheral3);`
	MFBD2	SIMO	in	1	←	0	←	24.3 MDX_B	out	HiZ	35	DigitalPin	MFBD2			`DigitalPin MFBD2(24, peripheral3);`
	MFBD3	-	-	1		0		<None>	-	-	-	<None>	MFBD3			`<None> MFBD3`
	MFBD4	FAULTn	out	0	→	0		<None>	-	-	-	<None>	MFBD4			`<None> MFBD4`
	MFBD5	-	-	1		0		<None>	-	-	-	<None>	MFBD5			`<None> MFBD5`
	MFBD6	-	-	1		0		<None>	-	-	-	<None>	MFBD6			`<None> MFBD6`
	MFBD7	-	-	1		0		<None>	-	-	-	<None>	MFBD7			`<None> MFBD7`
	MFBD8	-	-	1		0		<None>	-	-	-	<None>	MFBD8			`<None> MFBD8`
	MFBD9	-	-	1		0		<None>	-	-	-	<None>	MFBD9			`<None> MFBD9`
	MFBD10	-	-	1		0		<None>	-	-	-	<None>	MFBD10			`<None> MFBD10`
	MFBD11	-	-	1		0		<None>	-	-	-	<None>	MFBD11			`<None> MFBD11`

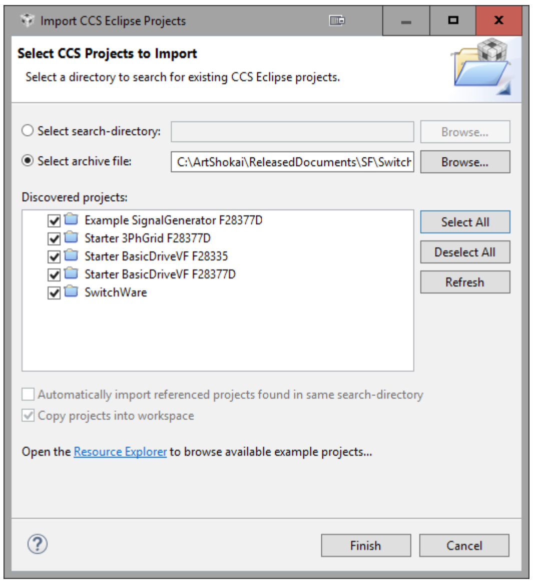
Figure 11: The digital MCU signal allocation (orange cells) for the AOV001 4-channel isolated analogue voltage output module. The CSn, SCLK and SIMO signals of the module must be mapped to the corresponding MCU signals (green outline) of the McBSP_A or McBSP_B peripheral. The AOV001 Module wrapper object must be initialised in main() with this peripheral data.

Import BasicDriveVF Example Project

The BasicDriveVF example project is included in the SwitcherWare Library install folder. You can import it into the new workspace by following these steps:

- In the **Project** menu, select the **Import CCS Projects...** menu item to show the **Import CCS Eclipse Projects** dialogue box. See Figure 12.
- Select the **Select search-directory** radio button, then click the **Browse...** button to the right and use the browser to navigate to and select the SwitcherWare Library install folder. Click **OK**.
- The **Discovered projects:** check list now shows all projects found, including the BasicDriveVF example project.
- Click on the check box to select the BasicDriveVF example project, or click on the **Select All** button to select all discovered projects.
- Click on the check box to select **Copy projects into workspace**.
- Click on the **Finish** button.

The project is imported into the workspace.



Import CCS Eclipse Projects

Select CCS Projects to Import

Select a directory to search for existing CCS Eclipse projects.

( ) Select search-directory: Browse...

(•) Select archive file: C:\ArtShokai\ReleasedDocuments\SF\Switch Browse...

Discovered projects:

- ☑ Example SignalGenerator F28377D
- ☑ Starter 3PhGrid F28377D
- ☑ Starter BasicDriveVF F28335
- ☑ Starter BasicDriveVF F28377D
- ☑ SwitchWare

Select All | Deselect All | Refresh

☐ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

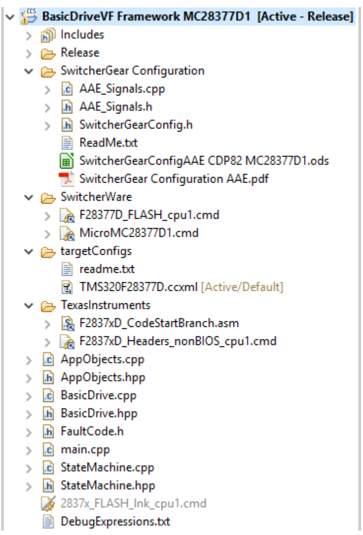
Open the Resource Explorer to browse available example projects...

Finish | Cancel

Figure 12: Importing the SwitchWare Library into CCS.

If the version of your installation of CCS is older than the version that was used to build the example project, then the import will fail and CCS with display an error message explaining the conflict in the compiler versions. The best way to resolve this issue is to install the latest version of CCS.

The file structure of the imported project is shown in Figure 13.



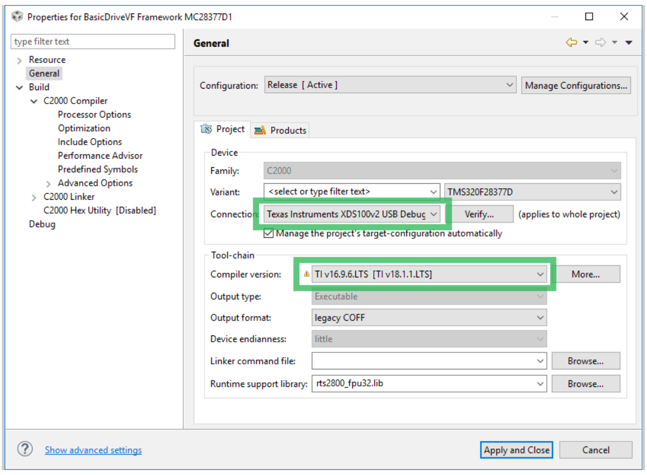
```
BasicDriveVF Framework MC28377D1  [Active - Release]
  Includes
  Release
  SwitcherGear Configuration
    AAE_Signals.cpp
    AAE_Signals.h
    SwitcherGearConfig.h
    ReadMe.txt
    SwitcherGearConfigAAE CDP82 MC28377D1.ods
    SwitcherGear Configuration AAE.pdf
  SwitcherWare
    F28377D_FLASH_cpu1.cmd
    MicroMC28377D1.cmd
  targetConfigs
    readme.txt
    TMS320F28377D.ccxml [Active/Default]
  TexasInstruments
    F2837xD_CodeStartBranch.asm
    F2837xD_Headers_nonBIOS_cpu1.cmd
  AppObjects.cpp
  AppObjects.hpp
  BasicDrive.cpp
  BasicDrive.hpp
  FaultCode.h
  main.cpp
  StateMachine.cpp
  StateMachine.hpp
  2837x_FLASH_lnk_cpu1.cmd
  DebugExpressions.txt
```

Figure 13: The file structure of the BasicDriveVF project.

Project Properties

You must check the project properties to ensure that they match the debug probe that you intend to use and the version of the C2000 build tools that are installed.

- In the **Project** menu, select the **Properties** menu item to show the **Properties** dialogue box.
- In the navigation pane at the left, select **General** to show the **General** properties. See Figure 14.
- In the **Connection** drop down list, select your debug probe. Click on the check box below to allow automatic target-configuration.
- In the **Compiler version** drop down list, select the newest compiler version that is listed.
- Click on the **Apply and Close** button.



Properties for BasicDriveVF Framework MC28377D1

type filter text

- Resource
- General
- Build
  - C2000 Compiler
    - Processor Options
    - Optimization
    - Include Options
    - Performance Advisor
    - Predefined Symbols
    - Advanced Options
  - C2000 Linker
  - C2000 Hex Utility [Disabled]
- Debug

General

Configuration: Release [Active] Manage Configurations...

Project | Products

Device

Family: C2000

Variant: <select or type filter text> TMS320F28377D

Connection: Texas Instruments XDS100v2 USB Debug Verify... (applies to whole project)

☑ Manage the project's target-configuration automatically

Tool-chain

Compiler version: TI v16.9.6.LTS [TI v18.1.1.LTS] More...

Output type: Executable

Output format: legacy COFF

Device endianness: little

Linker command file: Browse...

Runtime support library: rts2800_fpu32.lib Browse...

Show advanced settings

Apply and Close | Cancel

Figure 14: The general properties of the project, showing the selections for debug probe and compiler version.

Configuration Document

The SwitcherGear Configuration document and the automatically generated source/header files in the example project archive are based on a generic SwitcherGear configuration and reflect the SwitcherGear configuration described in this manual. They are provided for reference only. They may not match the actual configuration of your specific SwitcherGear controller.

To run the example project with your SwitcherGear controller, you must use the specific SwitcherGear Configuration document that was supplied with your SwitcherGear controller. This document contains the correct data for the physical configuration of your controller.

Configurations are identified by a unique 3 letter code. If the configuration of your SwitcherGear controller (see label on under side of base unit) is the same as the files in the example project, you can choose to ignore the following steps.

If your controller has a different configuration, the next sections explain how to change over the files for the BasicDriveVF project. The file **`ReadMe.txt`** in the SwitcherGear Configuration project folder also has instructions that are generally applicable.

Delete Existing Project Configuration Files

In the **Project Explorer** pane in CCS, select all files inside the **`SwitcherGear Configuration`** project folder. See Figure 13 for the structure of the project.

Press the **delete** key to delete the files, and confirm their removal from the file system.

Copy the Configuration Document

Open an explorer window and navigate to the location of the SwitcherGear Configuration Document that corresponds to your SwitcherGear controller.

Use the mouse cursor to drag the Configuration Document from the explorer window to the **`SwitcherGear Configuration`** project folder in the CCS **Project Explorer** pane. This is the same project folder that was emptied above.

In the **File Operation** dialogue box that appears, select the **Copy files** radio button and click OK.

Open the Configuration Document

The document can be opened either from an Explorer window, or from inside CCS. The first time you open a configuration document in CCS, do so by right-clicking the file, select **Open With > System Editor** from the context menu, navigate to the LibreOffice program folder and select the **calc.exe** application.

LibreOffice may show a security warning about macros contained in the document. Click on the **Enable Macros** button to enable editing and automated generation of the SwitcherGear Configuration source files. If you are not presented with this option, in LibreOffice, open the preferences window by selecting the menu **Tools > Options**. In the navigation pane select **LibreOffice > Security**. Then click on the **Macro Security** button and change the security level to **Medium**. Close the file and open again with the new security setting.

Modify the Configuration Document

Current sensor module

You must make the correct entries in the SwitcherGear Configuration document to define software objects for the current sensor signals. Refer to Figure 9 for the following steps:

- identify the hardware interface module in your SwitcherGear controller that you will use for the current sensors;
- identify the slot in which it is installed;
- locate the signal entries for this slot on the Signals tab in the SwitcherGear Configuration document; and
- edit the blue-highlighted analogue SwitcherWare Signal Object fields to match those shown in Figure 9.

You may wish to set the **`class`** entry of unused signals on this and other unused sensor modules to **`<None>`**. This removes the definition for these objects, which saves memory by not allocating space for them.

Voltage sensor module

You must make the correct entries in the SwitcherGear Configuration document to define software objects for the voltage sensor signal. Refer to Figure 9 for the following steps:

- identify the hardware interface module in your SwitcherGear controller that you will use for the voltage sensor;
- identify the slot in which it is installed;
- locate the signal entries for this slot on the Signals tab in the SwitcherGear Configuration document; and
- edit the blue-highlighted analogue SwitcherWare Signal Object fields to match those shown in Figure 9.

You may wish to set the **`class`** entry of unused signals on this and other unused sensor modules to **`<None>`**.

Converter module EPWM

If your SwitcherGear is configured to use a base EPWM peripheral other than EPWM1, then you must edit the **`main.cpp`** source file. Refer to Figure 10 for the following steps:

- identify the hardware interface module in your SwitcherGear controller that you will use to interface to the converter;
- identify the slot in which it is installed;
- locate the signal entries for this slot on the Signals tab in the SwitcherGear Configuration document;
- find the name of the MCU Signal that has been allocated to the **`HB1CHA`** MCU interface signal, which indicates the base EPWM peripheral.
- If the base EPWM peripheral is *not* **`EPWM1`**, you must edit the following line of the source file **`main.cpp`** so that the EPWM number matches the EPWM value that you found above:

```
theConverter.Init(PERIPHERAL_ID_EPWM1, ...); // FIXME
```

The above lines of code in main.cpp are followed by a C comment that begins **`// FIXME`**. (CCS indicates the position in the file of these special comments with a blue marker in the right margin of the text editor, next to the vertical scroll bar.) The comments provide specific information on the code resources that must be matched to the hardware configuration.

Converter module Trip signal

If your SwitcherGear is configured to use a GPIO pin for the converter trip signal other than GPIO13, then you must edit the **main.cpp** source file. Refer to Figure 10 for the following steps:

- find the name of the MCU Signal that has been allocated to the **HB123TRIPn** MCU Interface Signal. It starts with numbers in the format XX.Y, where XX is the GPIO number.
- Ensure that the second argument of the following function in the source file **main.cpp** is equal to the GPIO number of the Trip signal:

```
theTripSignal = theF2837xDCPU.CreateTripSignal(13, true);
```

Analogue output module

The initialisation function for the analogue output module must refer to the McBSP peripheral that is allocated in your SwitcherGear configuration, either peripheral A or B. Refer to Figure 11 for the following steps:

- identify the hardware interface module in your SwitcherGear controller that you will use for the analogue signal output;
- identify the slot in which it is installed;
- locate the signal entries for this slot on the Signals tab in the SwitcherGear Configuration document; and
- locate the MCU Interface Signals for the SPI bus, **CSn/SCLK/SIMO**.
- If the corresponding MCU Signals are **MFSX_A/MCLKX_A/MDX_A**, you must ensure the following line of the source file **main.cpp** has the first argument as highlighted here:

```
theRealTimeScope.Init(PERIPHERAL_ID_MCBSPA, DMA_CHANNEL1);
```

- If the corresponding MCU Signals are **MFSX_B/MCLKX_B/MDX_B**, you must ensure the following line of the source file **main.cpp** has the first argument as highlighted here:

```
theRealTimeScope.Init(PERIPHERAL_ID_MCBSPB, DMA_CHANNEL1);
```

Generate Documents

Even if no changes were made to the SwitcherGear Configuration document, you must generate the new source/header files.

Save the document.

Show the Project sheet by clicking the Project tab at the bottom of the window.

Then click on the **Generate Documents** button to generate the source files and PDF documentation for the SwitcherGear configuration.

Code Execution

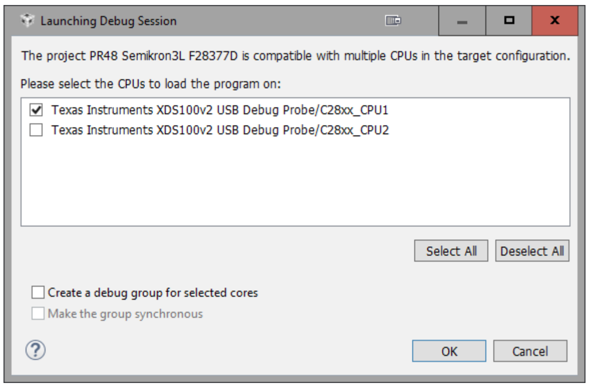
Make sure the debug probe is connected, SwitcherGear controller is turned on and power system is turned off.

Click the toolbar **Debug** button or press the **F11** key to debug the project. This performs the following tasks:

- compiles the project source files;
- links the project with the SwitcherWare Library;
- connects the debug probe to the MCU target;
- erases the flash memory of the MCU target;
- writes the build object into the MCU flash memory;
- starts execution of the MCU to run the C environment initialisation code; and
- pauses the program execution at the start of **main()**.

Any errors are shown in the **Console View**.

If you are building the project for the F28377D MCU, you will be prompted to select the CPUs to load the program onto. Select **C28xx_CPU1** and deselect **C28xx_CPU2**, as shown in Figure 15.


Launching Debug Session

The project PR48 Semikron3L F28377D is compatible with multiple CPUs in the target configuration.

Please select the CPUs to load the program on:

- [✓] Texas Instruments XDS100v2 USB Debug Probe/C28xx_CPU1
- [ ] Texas Instruments XDS100v2 USB Debug Probe/C28xx_CPU2

Select All | Deselect All

- [ ] Create a debug group for selected cores
- [ ] Make the group synchronous

OK | Cancel

Figure 15: CPU selection when loading code to a dual-core F2837xD for the first time.

During the above process, the view is changed from the Edit Perspective to the Debug Perspective, which shows information related to the debug process.

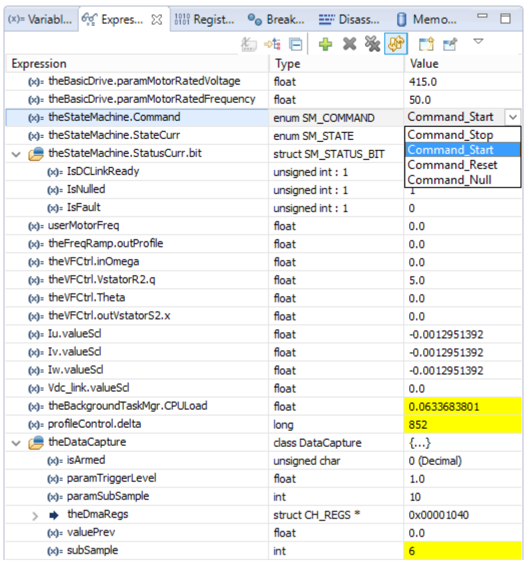
Now, click the toolbar **Resume** button or press the **F8** key to start program execution.

Expressions

The **Expressions View** can be used to observe code variables and control the program execution. The values in the **Expression View** should be configured to update periodically by clicking on the icon (continuous refresh) in the toolbar area – see Figure 16.

The project file **DebugExpressions.txt** contains the key variables that you can use to observe and control the application. Import the variable names into the **Expressions View** by right-clicking in the list area, selecting **Import...** from the context menu and selecting the file in the file browser.

When the program is running, the real-time values of the variables are shown in the list. In particular, you should see the voltage and current signals fluctuating slightly – the **Value** field of a variable is highlighted in yellow when its value is different from the previous sample.



Expression	Type	Value
theBasicDrive.paramMotorRatedVoltage	float	415.0
theBasicDrive.paramMotorRatedFrequency	float	50.0
theStateMachine.Command	enum SM_COMMAND	Command_Start
theStateMachine.StateCurr	enum SM_STATE	Command_Stop
theStateMachine.StatusCurr.bit	struct SM_STATUS_BIT	Command_Start
IsDCLinkReady	unsigned int : 1	Command_Reset
IsNulled	unsigned int : 1	Command_Null
IsFault	unsigned int : 1	0
userMotorFreq	float	0.0
theFreqRamp.outProfile	float	0.0
theVFCtrl.inOmega	float	0.0
theVFCtrl.VstatorR2.q	float	5.0
theVFCtrl.Theta	float	0.0
theVFCtrl.outVstatorS2.x	float	0.0
Iu.valueScl	float	-0.0012951392
Iv.valueScl	float	-0.0012951392
Iw.valueScl	float	-0.0012951392
Vdc_link.valueScl	float	0.0
theBackgroundTaskMgr.CPULoad	float	0.0633683801
profileControl.delta	long	852
theDataCapture	class DataCapture	{...}
isArmed	unsigned char	0 (Decimal)
paramTriggerLevel	float	1.0
paramSubSample	int	10
theDmaRegs	struct CH_REGS *	0x00001040
valuePrev	float	0.0
subSample	int	6

Figure 16: The Expressions View in the Debug Perspective. When clicked for editing, the view shows the enumerated constants for the state machine command variable. Use the up/down arrow keys to select and Enter key to accept.

Change the parameters for the motor voltage and frequency ratings by clicking on the **Value** field of the expression, typing the required value and pressing the **Enter** key.

Verify Controller Inputs And Outputs

Before operating the power system, you must verify that the inputs to and the outputs from the SwitcherGear controller are functioning correctly. These test are performed using low voltages.

⚠	Remove the 3-phase supply connections from the output of the variable supply.
⚠	Use a multimeter to verify that the DC link is discharged.

Voltage Sensor

Before turning on the mains supply to the power stage, you must validate that the voltage measurement is working correctly. This is important because the PWM generation algorithm uses the measured DC link voltage to calculate duty cycle.

Use a multimeter to measure the DC link voltage in the un-powered state. Note that the voltage may not be zero because the gate driver circuitry has parasitic charging paths.

Confirm that the value of the **`Vdc_link.outScaled`** variable agrees with the multimeter reading within 3 V.

Now connect the output of a laboratory power supply to the DC link and apply a voltage between 30 and 50 V. Confirm that the value of the **`Vdc_link.outScaled`** variable agrees with a new multimeter reading.

Disconnect the multimeter and laboratory power supply.

Current Sensors

You should also validate the current sensor measurements, although these are for monitoring purposes only.

Unplug the motor wiring connections from the phase U motor current sensor. Confirm that the value of the **`Iu.outScaled`** variable is within 0.1 A of zero. The zero current reading should be quite accurate because the sensors were nulled at start-up.

Use the laboratory power supply in current source mode to apply a 3 to 5 A current through the sensor. Use the multimeter to measure the current through the sensor. Confirm that the magnitude and sign of the **`Iu.outScaled`** variable agrees with the multimeter reading within 0.2 A.

Repeat the above steps with the other current sensors.

Disconnect the power supply and multimeter from the current sensor, and reconnect the motor wiring.

Gate Drive Outputs

Disconnect the motor from the output of the converter.

Connect a laboratory power supply across the DC link, making sure to observe the correct polarity. Apply a voltage of 30 V to the DC link.

Change the state of the BasicDriveVF application to the Test state. First, the entry to the Test state must be allowed. Do this by clicking on the “>” expansion symbol to the left of the **`theStateMachine.inFlags.bit`** variable in the **Expressions View** to show the flags **`allowStart`** and **`allowTest`**. Click on the value field of the **`allowTest`** flag, change the value to **1** and press the **Enter** key.

Next, click on the **Value** field of the **`theStateMachine.inCommand`** variable, click on the disclosure button that appears in the field to show a drop-down list of the enumerated command constants – see Figure 16. Select **`command_Test`** from the list and press the **Enter** key. This will cause the state to change and the value of the **`theStateMachine.outStateCurr`** variable will change from **`state_Ready`** to **`state_Test`**. Notice that the value of the **`allowTest`** flag is automatically changed to **`0`**.

The PWM outputs are now active and the IGBTs are being switched to generate a voltage space vector. Confirm that the current drawn from the laboratory supply is small. A large current may indicate a mistake in the wiring of the motor circuit.

Use an oscilloscope to confirm that the voltage on each of the three half-bridge outputs is a square wave that is switching between the DC link negative and positive voltages. Confirm that the switching frequency is 5 kHz.

You can manually change the demanded output voltage space vector by writing new values to the variables **`theConverter.inVdclink`** and **`theConverter.inVdemandS3`**. You should confirm that a change to **`theConverter.inVdemandS3.u`** affects the PWM duty of the phase U output of the converter, and so on. (This project uses SVM, so the duty of the other phases will be affected slightly.)

Issue the Stop command to change the state to Ready and disable drive operation.

Disconnect the laboratory power supply.

Reconnect the motor to the output of the converter.

If all the above checks were successful, proceed to the next section.

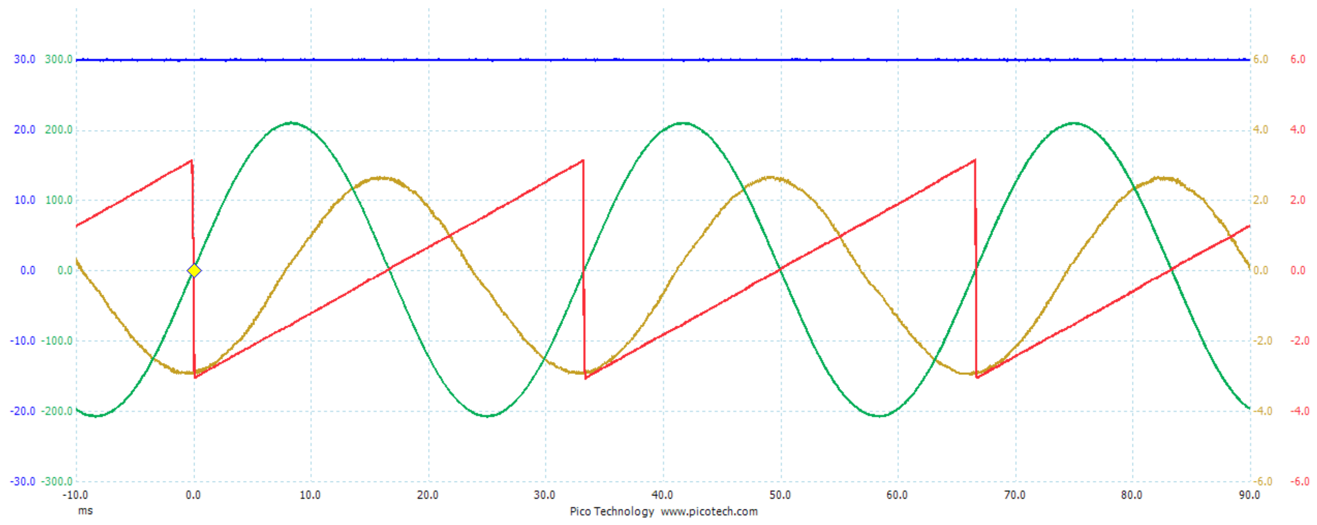


Figure 17: Real-time visualisation of controller variables using the AOV001 Module and a 4-channel oscilloscope (trace colour, unit): user demand frequency at output of ramp limiter (blue, Hz); voltage space vector angle (red, radian); demand phase U output voltage (green, V); and, measured phase U motor current (yellow, A). All the input channels are set to a physical range of ±10 V and the signal scaling seen here is achieved using the probe scaling method described in the text. The yellow diamond indicates the oscilloscope trigger position on the falling edge of the space vector angle, when its value wraps from π to -π.

Drive Operation

⚠	Hazardous voltages are present in the following steps.
⚠	Before proceeding, you must make a risk assessment and put in place measures to ensure safety of personnel and equipment.

Turn off the variable supply. Then reconnect the 3-phase supply connections to the output of the variable supply.

Connect a multimeter to measure the line-line input voltage to the SemiTeach rectifier.

Turn the voltage adjustment of the variable supply to zero. Then switch on the supply.

Issue the Start command to change the state to Run and enable drive operation. The converter is now actively switching and applying a PWM modulated voltage to the motor.

Increase the supply voltage gradually until the nominal supply voltage is reached, or no more than 400 VAC.

Now use the multimeter to measure the DC link voltage. Confirm that this value agrees with the DC link voltage measured by the SwitcherGear controller in the **`Vdc_link.outScaled`** variable.

Change the value of **`userMotorFreq`** to 10 Hz.

The value of the **`theFreqRamp.outProfile`** variable will ramp slowly from 0 to 10 Hz. Correspondingly, the value of the **`theVFCtrl.inOmega`** variable will ramp from 0 to 62.8 rad/s. The motor should be spinning and the measured currents should be fluctuating.

The **Expressions View** can be used to observe variables that change slowly from one sample to the next, i.e. variables that are static or varying slowly during drive operation or in the rotating reference frame. Variables that change faster than the sample rate will be hard to interpret, i.e. the space vector angle and the voltages and currents in the stationary reference frame.

Data Visualisation

Real-time visualisation of digital variables is ideal for observing AC and transient behaviour. It can be used to observe any **`float`** variable in the controller, which can include measured quantities.

The AOV001 module generates an output voltage in the range of ±10 V. For the best performance, you should set the input range of the oscilloscope channels to this range, or the closest available range. Some oscilloscopes have a ±10 V range setting or a 2.5 V/div setting that matches the module's output range. For other oscilloscopes, a setting of 2 V/div is the closest, which gives a display range of ±8 V. *Once you have set the input range do not adjust the range settings.*

Most oscilloscopes have a probe scaling feature that can be used to recover the original scaling of the digital variables. Set the probe gain of each oscilloscope channel to the **`paramValuePerOutput`** value used in the initialisation of each analogue output channel. If you use this method, you must not adjust the oscilloscope scaling by changing the input range setting that was made in the previous paragraph.

You can use the oscilloscope in the usual way to investigate the waveforms from the SwitcherGear controller. Figure 17 shows a screen image of the variables that are output with the default settings of the BasicDriveVF project. The input range of the oscilloscope was set to ±10 V and the probe scaling of each channel was set to recover the original scaling of the digital variables.

Some oscilloscopes have a high resolution mode that enhances the detail of the traces. This can improve the clarity of the traces significantly. An oscilloscope with a high resolution ADC (10 to 12 bits) will also give improved results.

The oscilloscope can also be used to capture the results by saving the screen image or the raw channel data. This is a simple method to capture the data, but the analogue signal chain introduces errors and noise. Also, the captured data is not sampled at the control frequency and requires post-processing to re-sample it.

Data Capture

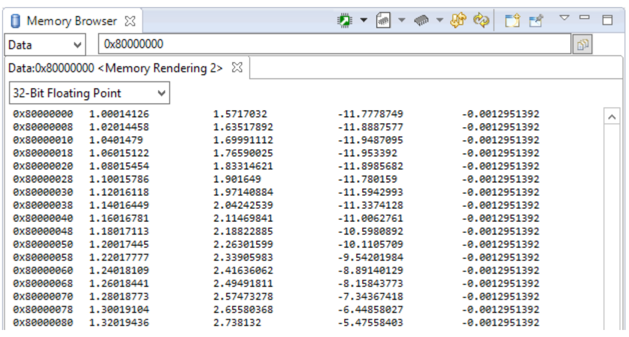
In this project, the data capture object is initialised to capture the same four variables that are output by the analogue output module, with 1000 samples captured into the external SDRAM of the MC28377D micro card.

In the **Expressions View**, expand the object **`theDataCapture`** to show its member variables. The default format of the value of the **`bool`** variable **`varTriggerArmed`** is not clear. Right-click on the variable **`varTriggerArmed`** and select the contextual menu item **Number Format > Decimal**. The value will now show as 0 for false and 1 for true.

In the armed condition, the object waits for the condition that will trigger the data capture. The member function **`ConfigParams_TriggerRisingEdge()`** triggers the data capture when the trigger signal crosses **`inTriggerLevel`** in the rising direction. This is the same trigger behaviour as the rising edge trigger feature used in all oscilloscopes. The trigger signal is passed as the argument in the function call, and in this project the ramp limited user demand frequency is used.

If the motor is already running, set **`userMotorFreq`** to 0.0 and wait for the ramp to reach the target value. Then, set the member variables **`inTriggerLevel`** to 5.0 and **`varTriggerArmed`** to 1. Now set the value of **`userMotorFreq`** to a value of 10.0. The data capture will be triggered as the frequency ramps through 5 Hz and will capture the 1000 samples at the control rate.

To see the captured data, click on the **View** menu and select the **Memory Browser** menu item. Select **Data** in the memory type drop-down list. In the address field to the right, type **0x80000000** and press **Enter**. In the Data Format drop-down list, select **32-Bit Floating Point**. See Figure 18.



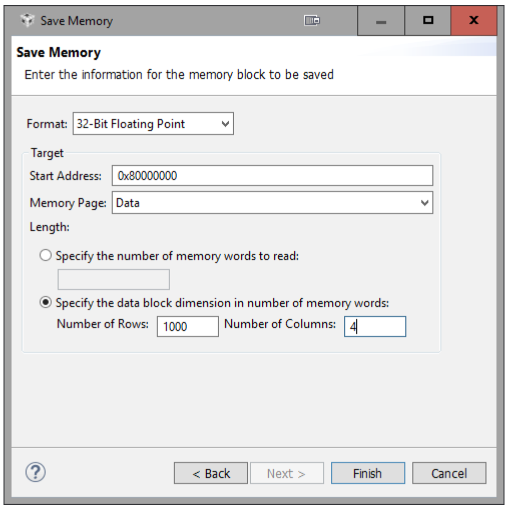
```
0x80000000   1.00014126    1.5717032     -11.7778749   -0.0012951392
0x80000008   1.02014458    1.63517892    -11.8887577   -0.0012951392
0x80000010   1.0401479     1.69991112    -11.9487095   -0.0012951392
0x80000018   1.06015122    1.76590025    -11.953392    -0.0012951392
0x80000020   1.08015454    1.83314621    -11.8985682   -0.0012951392
0x80000028   1.10015786    1.901649      -11.780159    -0.0012951392
0x80000030   1.12016118    1.97140884    -11.5942993   -0.0012951392
0x80000038   1.14016449    2.04242539    -11.3374128   -0.0012951392
0x80000040   1.16016781    2.11469841    -11.0062761   -0.0012951392
0x80000048   1.18017113    2.18822885    -10.5980892   -0.0012951392
0x80000050   1.20017445    2.26301599    -10.1105709   -0.0012951392
0x80000058   1.22017777    2.33905983    -9.54201984   -0.0012951392
0x80000060   1.24018109    2.41636062    -8.89140129   -0.0012951392
0x80000068   1.26018441    2.49491811    -8.15843773   -0.0012951392
0x80000070   1.28018773    2.57473278    -7.34367418   -0.0012951392
0x80000078   1.30019104    2.65580368    -6.44858027   -0.0012951392
0x80000080   1.32019436    2.738132      -5.47558403   -0.0012951392
```

Figure 18: The captured data shown in the Memory Browser View.

To download the captured data from the MCU to your PC, right-click on the **Memory Browser View** and select the **Save Memory ...** contextual menu item.

On the first page of the **Save Memory** dialogue box, enter the path and name for the saved data file, select **TI Data** from the **File Type** drop-down list, then click the **Next >** button.

On the second page, select **32-Bit Floating Point** in the **Format** drop-down list and type **0x8000 0000** in the **Start Address** text box. For the **Length** radio group, select the second option to specify the data block dimensions. Type 1000 in the **Number of Rows** text box for the number of vectors stored in the buffer. Type 4 in the **Number of Columns** text box for the size of the vector. See Figure 19. Click the **Finish** button. Depending on the amount of data and the speed of the debug probe connection, it can take a few seconds or a few minutes to transfer the data.



Save Memory
Enter the information for the memory block to be saved

Format: 32-Bit Floating Point
Target
Start Address: 0x80000000
Memory Page: Data
Length:
( ) Specify the number of memory words to read:
(●) Specify the data block dimension in number of memory words:
Number of Rows: 1000 Number of Columns: 4

Figure 19: Settings for saving the capture data buffer.

Optimisation

By default, the compiler optimisations in the BasicDriveVF project are turned off. You can use some simple tools in the SwitcherWare Library to estimate the number of instruction cycles the control algorithm takes to execute and the overall load on the MCU.

The object **`profileControl`** is an instance of the Profiler class. The **`Entry()`** and **`Exit()`** member functions are used to bracket the code section of interest. The **`outCycles`** member variable contains an estimate of the number of instruction cycles that elapsed between these function calls.

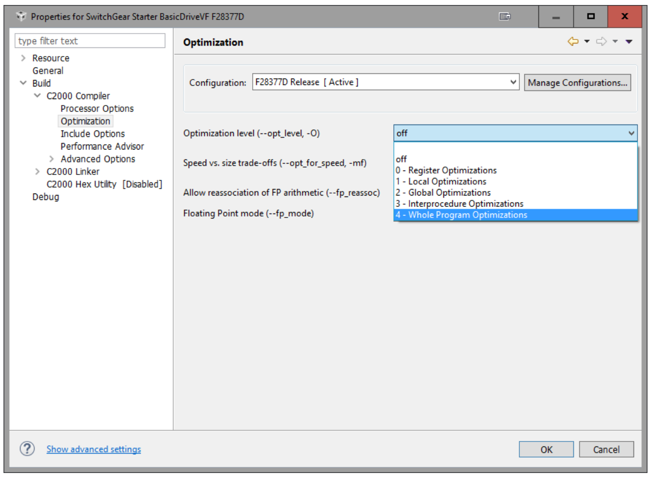
The variable **`theBackgroundTaskMgr.varCPULoad`** is an estimate of the total load on the F28 CPU. The value ranges from 0.0 to 1.0, where 0.0 indicates that the CPU is idle. This is is an estimate of the total time spent executing the background and foreground tasks, including “hidden” overheads such as context saving for interrupts, etc.

These are the basic tools for evaluating the effect of code changes and optimisations.

Issue the Stop command to the state machine and confirm that the state changes to Ready. Notice that the CPU load and the execution time of the profiled control code decrease because the V/F control blocks are not executed in the Ready state.

Click the toolbar **Terminate** button to end the debug session. The environment changes back to the **Edit Perspective**.

Right-click on the BasicDriveVF project in the **Project Explorer** and select the **Properties** context menu item. Navigate the tree structure on the left side of the **Properties** dialogue box to show the **Build > C2000 Compiler > Optimization** properties. Set **Optimization level** to **4 – Whole Program Optimizations** and **Speed vs. size trade-offs** to **5**. See Figure 20. Click the **OK** button.



Optimization

Configuration: F28377D Release [Active]

Optimization level (--opt_level, -O): off

Speed vs. size trade-offs (--opt_for_speed, -mf)

Allow reassociation of FP arithmetic (--fp_reassoc)

Floating Point mode (--fp_mode)

off
0 - Register Optimizations
1 - Local Optimizations
2 - Global Optimizations
3 - Interprocedure Optimizations
4 - Whole Program Optimizations

Figure 20: Optimisation level setting in the project Properties dialogue box.

Debug the project to re-build and run the application.

Table 1 shows the typical reduction in execution time and CPU load after the optimisations are enabled. It should be noted that even with compiler optimisation turned off for the BasicDriveVF project, there are many other optimisation that are already in play:

- the SwitcherWare Library is coded to take advantage of the C28x instruction set and the MCU hardware peripherals.
- the SwitcherWare Library is compiled with optimisation turned on.
- time critical code is executed from RAM, which is slightly faster than executing code from flash memory.
- the FPUfastRTS library is used to improve floating-point maths operations.

Table 1: Improvement of code execution time due to compiler optimisation. Data for the state machine in Run state.

	Optimisation		Unit
	None	Level 4	
`theBackgroundTaskMgr.varCPULoad`	0.0636	0.0444	-
`profileControl.outCycles`	855	523	cycle
Execution time of profiled code (instruction rate 200 MHz)	4.28	2.62	µs

Extensions To The BasicDriveVF Project

GUI Composer Interface

You can build a graphical user interface for this controller using GUI Composer. Use standard controls (buttons, sliders, dials, etc.) and displays (text fields, indicators, etc.) to build an interface. Then bind these interface elements to variables in the application code.

GUI Composer is a web-based tool on the Texas Instruments website at

https://dev.ti.com/

Add Inrush Current Limiting

To use the basic drive from a non-variable supply, an inrush current limiting circuit (also known as a soft-starter or pre-charger) must be added to protect the power stage. This consists of current limiting resistors in series with the supply and a 3-phase contactor that bypasses the resistors after the DC link is charged.

A DIO003 8-channel digital input/output module is added to the SwitcherGear controller to provide a 24 V switched output to control the contactor. This can be used to directly drive contactors whose coils are rated for 24 V with a steady-state current requirement up to 0.5 A, (or 1.6 A if four module channels are used in parallel).

Finally, the state machine is altered to introduce a new charging state, which must be active when the DC link voltage is low. The state is used to control the bypass contactor and enabling of drive operation.

Control The Drive With Digital I/O

To allow control of the drive without CCS and the debug probe, you can build a digital I/O interface. Using a SwitcherGear DIO003 module, you can connect push buttons that allow users to control the application, and indicators that allow the application to show operational status to the user. Three buttons can be used to issue commands to the state machine – Start, Stop and Reset. Two more buttons can be used to jog the demand frequency up and down.

Control The Drive Over Serial Communications

To allow control of the drive without CCS and the debug probe, you can build a serial communications interface. SwitcherGear provides interface modules for the standard serial communications RS-232, RS-485, CAN and Ethernet.

Port The Control Algorithm To The CLA Co-Processor

The MC28377D is a dual-core microcontroller and each F28 CPU core has a CLA co-processor. All operate at the same maximum instruction clock frequency, 200 MHz. The CLA is optimised to perform real-time control and it is ideal for executing the foreground control task. Not only can the CLA usually execute the control task more quickly, but it also frees the F28 CPU to perform other tasks.

Complex control tasks can be split into smaller tasks that can be run simultaneously on the two F28 CPUs and two CLA co-processors.

References

[1] ‘SwitcherGear Micro MC28377D’, Denkinetic Pty Ltd, Reference Manual DD00045.

[2] 'SwitcherGear Module B12CC1', Denkinetic Pty Ltd, Reference Manual DD00022.

[3] 'SwitcherGear Module CON002', Denkinetic Pty Ltd, Reference Manual DD00024.

[4] 'SwitcherGear Accessory SNI005', Denkinetic Pty Ltd, Reference Manual DD00058.

[5] 'SwitcherGear Module AIN004', Denkinetic Pty Ltd, Reference Manual DD00042.

[6] 'SwitcherGear Accessory SNV005', Denkinetic Pty Ltd, Reference Manual DD00054.

[7] 'SwitcherGear Module AOV001', Denkinetic Pty Ltd, Reference Manual DD00056.

[8] 'SwitcherGear Module AOU001', Denkinetic Pty Ltd, Reference Manual DD00026.

[9] 'Install SwitcherWare Library', Denkinetic Pty Ltd, Application Note DD00051.

Revision History

Revision	Date	Changes From Previous Release
1	14 Sep 2016	■ Original release.
2	08 Aug 2018	■ Updated for SwitcherWare Library version 0.6.3 and new SwitcherGear hardware.
3	29 Mar 2019	■ Updated state machine for SwitcherWare Library version 0.7.0. ■ Updated hardware configuration.