Iowa State University
Electrical and Computer Engineering
E E 452. Electric Machines and Power Electronic Drives

Laboratory #5
Buck Converter – Embedded Code Generation

Summary

In this lab, you will design the control application for the buck converter you designed in Lab 4. All controls will be developed using the MATLAB/Simulink Embedded Coder toolbox. The Texas Instruments High-Voltage Motor Control and Power Factor Correction development kit, with the F28035 ControlCARD, will be used to implement the design in Lab 6.

Learning objectives

- Design an embedded controller using MATLAB/ Simulink Embedded Coder.
- Configure the enhanced Pulse Width Modulation (ePWM) and Analog-to-Digital Converter (ADC) peripherals on the Piccolo F28035 ControlCARD.

Background material (should be read before coming to the lab)

- *TMDSHVMTRPFCKIT Hardware Reference Guide* (*HVMotorCtrl+PFC_HWGuide.pdf*)
- TMDSHVMTRPFCKIT schematics (*HVMotorCtrl+PFC-SCH[R5].pdf*)
  – Peripheral Mapping and DCBUS Sensing
- *F28035 ePWM reference guide* (*ePWM_spruge9e.pdf*)
  – Peripheral Configuration
- *F28035 Analog-to-Digital Converter reference guide* (*ADC_spruge5f.pdf*)
  – Conversion equations
- *MATLAB User's Guide*
  – Sections on using ePWM and ADC blocks for TI C2000 devices

Exercises and Questions

Instructions: every student should deliver his/her own report at the end of the lab session, even though the experiments are conducted in groups. You may want to answer the questions as you go along the exercises. Time yourselves according to the recommendations below.

1. Pre-lab assignment

Read through the *TMDSHVMTRPFCKIT Hardware Reference Guide* to familiarize yourself with the purpose and capabilities of this platform. Pay particular attention to jumper locations and uses. Also note, the *Resource Allocation* table in section 4.1 has some typos. The ePWM modules are actually connected opposite of what the document shows; ePWM1A is actually connected to the High-Side switch, not the Low-Side switch. Verify these connections via the schematics.

DELIVERABLE 1: Identify and describe the major components of the hardware kit, which are denoted by Macro blocks. What is their purpose and how can they be used to implement a dc/dc converter?

Study the schematics for the development kit. (*HVMotorCtrl+PFC-SCH[R5].pdf*)

DELIVERABLE 2: Which ADC module (A or B), and what channel (0-7) is used to sense the DC-Bus voltage? Ignoring the 10 nF filter capacitor, how is the DC-Bus voltage related to the ADC input? Formulate the equation.

Study the ADC user guide (*ADC_spruge5f.pdf*) and the Embedded Coder help files (available through the MATLAB help menu).

DELIVERABLE 3: How is the input voltage to the ADC related to the digital output of the ADC? What is the data type of the ADC output value? What is the digital value of a 150V DC-Bus voltage?

2. Introduction to the TI HVMTRPFC Development Kit [15 minutes]

Unpack the HVMTRPFCKIT, and identify the location of the macro blocks and jumpers on your hardware kit. Familiarize yourself with the hardware. The control application designed in this lab will run on this development kit. Understand the parts of the kit that are being used to implement the buck converter.

3. Model Configuration and Code Generation [150 minutes]

The TMDSHVMTRPFCKIT contains the F28035 ControlCARD you programmed in Lab 2. Again, Embedded Coder will be used to design the control application. Because the F28035 controller is capable of controlling many different systems, its configuration can become very complicated. Embedded Coder simplifies this process by providing interactive GUIs to configure the ControlCARD.

The embedded control application will implement several key functions. It will sense the DC input voltage, calculate the duty cycle required to achieve the desired output voltage, and drive one of the PWM outputs. Additionally, the calculated DC-bus voltage will be output as an analog signal, for the sake of monitoring the variable in real time. Furthermore, the input voltage will be monitored to provide over-voltage protection, with application status provided via an on-board LED. Follow the steps outlined below to develop your embedded controller.

a) DC-Bus Voltage Measurement – Configure the ADC

Open MATLAB/ Simulink, and create a new model to program the F28035. Configure the *Target Preferences* block as you did in Lab 2.

Add the *ADC* block to your Simulink model, and open the block dialog box as shown in Figure 1. Set the *Input Channel* to the appropriate value to measure the DC-Bus voltage. Set the *Sampling Mode* to *Single Sample*.

The ADC starts the conversion process when signaled to. This signal can be provided by a variety of sources. Set the *Trigger Number* to *SOC1*, and the *Trigger Source* to *Software.* Do not generate any interrupts at the end-of-conversion (EOC). Set the *Data type* to a value appropriate for your mathematical methods. Set the *Sample time* to *0.001 s*; the value used by the mathematical blocks connected to this output will receive a new value every 1 ms, regardless of the speed at which a new "Start of Conversion" is initiated.

Note: The converted value will be stored in the ADC Result register corresponding to the trigger number, not the channel number.

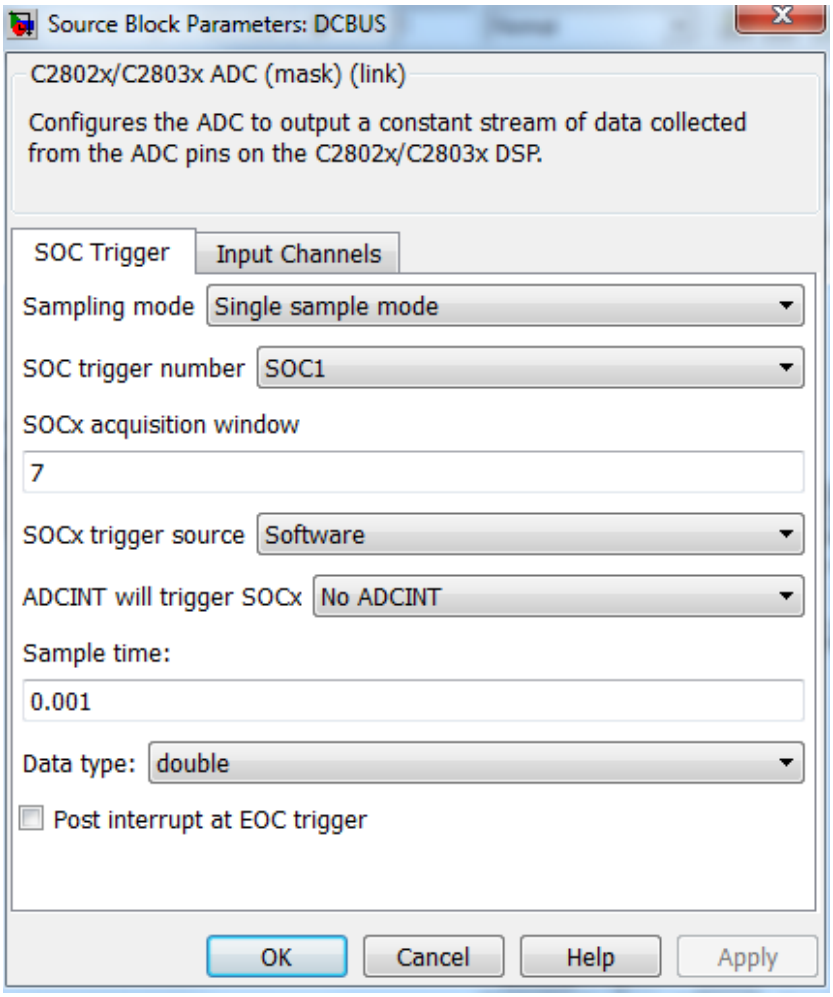


Figure 1. ADC dialog box to specify ADC operation.

Now that the ADC is configured, you must convert the ADC output from a digital value to the corresponding real world value. Use the ADC conversion equation and the DC-Bus voltage divider to accomplish this.

$$Digital\ Value = 4096 * (V_{in}/3.3) \quad (1)$$

$$V_{in} = \frac{R1}{R1+R2} * DCBUS \quad (2)$$

In your model, build the expression relating the DC-Bus voltage to the digital value. The input to this equation should be the digital value of the converted signal, and the output should be the DC-Bus voltage as a real world number.

To test your system, determine what the digital value is for a DC-Bus voltage of 100V. Simulate your system by using a *Constant* block to mimic the output of the ADC. Run the simulation to verify that your math is correct; your simulation time can be very short.

b) DC-Bus Voltage Measurement – Verifying calculations in real time

The TMDSHVMTRPFCKIT provides four digital-to-analog converters (DAC), which are meant to be used to monitor system variables. While not necessary, it is useful to monitor control variables. If the controller does not behave as expected, you will be able to verify that calculations of interest are being performed properly. The DACs on this board are driven by ePWM channels, and are filtered through a first order low-pass filter. See the schematics for further detail.

We will configure an ePWM module to verify that the DC-Bus voltage is being sensed properly. Choose a DAC output to monitor the DC-Bus voltage.

DELIVERABLE 4: What DAC did you choose, and what corresponding ePWM channel is used to drive this output? The DAC has a first order output filter, such that the output is a DC voltage corresponding to the driving signal's duty cycle. What is the corner frequency for this filter? What is an approximate minimum driving frequency for this DAC output?

Add an *ePWM* block to your model and open the block dialog box as shown in Figure 2. Set the *Module* to match the DAC output you are using. Set the peripheral *Timer period* (ePWMx PRD register) for 600 clock cycles, *Count mode* to *Up-Down*, and *TB prescale divider* to 1. This will configure the PWM output for a switching frequency of 50 kHz, well above the corner frequency of the DAC output filter, resulting in a very clean dc signal.

Note: The prescale divider specifies how the ePWMx module scales the 60 MHz system clock. A prescale divider of 2 results in a PWM clock of 30 MHz. The PWM switching frequency and duty cycle are based on this clock.

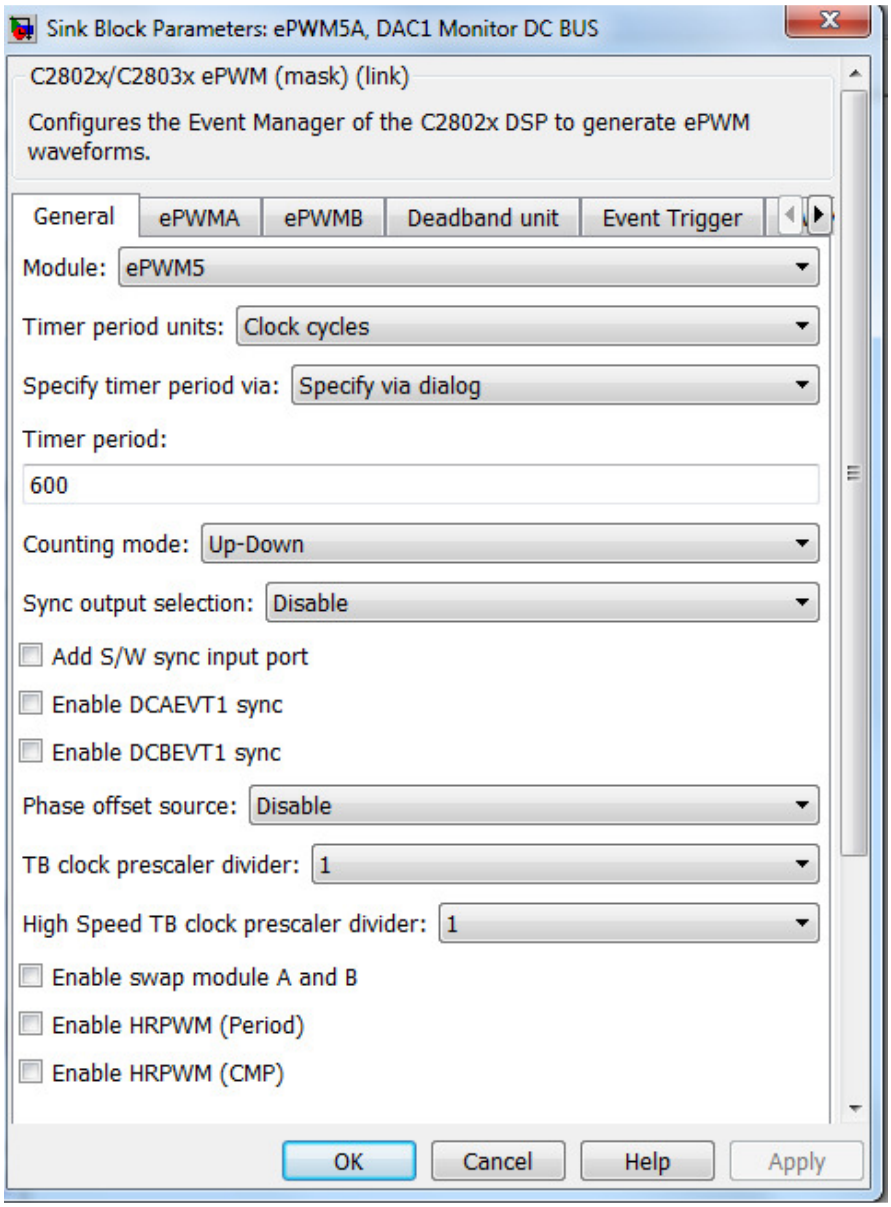
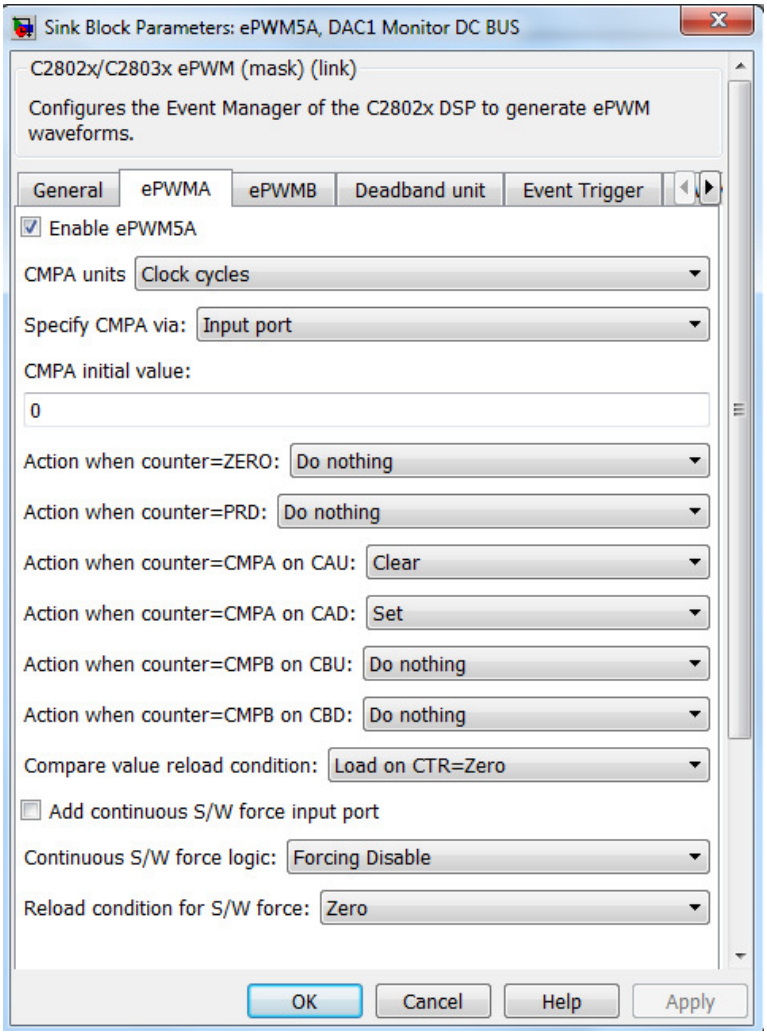


Figure 2. ePWM dialog box to specify ePWM operation.

On the *ePWMA* tab, shown below, enable the ePWMxA, where *x* is the value of the *module*. This tab configures the behavior of the ePWMxA output. The *CMPA* (ePWMx Compare A register) value is used to specify a relative point in time at which a switching action should occur. Set *CMPA units* to *Clock cycles* specified via *Input port*. Set the *Actions* such that a signal is produced with variable duty cycle, based on the value of *CMPA*. See the Help Menu for this block, for clarification on how to set actions appropriately.



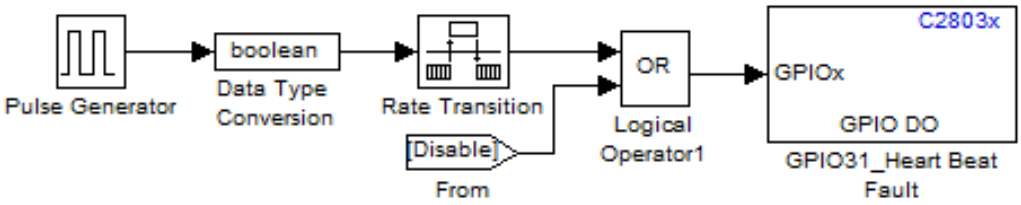
Disable *ePWMxB*; there is no need to use this output. Also, there is no need to use the *Deadband, Event Trigger*, or other submodules.

Previously, you set the *ePWMx PRD* value to 600 clock cycles; this is also the value of ePWMx CMPA which will generate a 100% duty cycle. In your Simulink model, multiply the DC-Bus voltage by 600/330, such that 330 V corresponds to a duty cycle of 100 %; i.e. 3.3 V. By doing so, a DAC output of 10 mV will correspond to 1 V on the DC-Bus.

c) Fault Detection and Program Status LED

Power switching devices must not be operated beyond their specified limits. It is wise to include fault detection and system status operations in your control program. This will visually show you the state of your embedded application.

Add some math and/or logic blocks to alert if the DC-Bus voltage goes above some threshold, say 150 V. Also, add the following blocks to show program status; the "Disable" input is an output of the fault detection subsystem, and the pulse generator block provides a heartbeat to show the application is running.



Set the *Digital Output* to correspond to the ControlCARD's red LED, which is connected to *GPIO31*. This LED will provide a heartbeat, indicating that code is running, and will stay illuminated if the DC-Bus voltage is too high.

Blocks from the *Sources* library may operate at a different rate from what the control application is running. Because we are using a *Pulse Generator* in our control application, it is necessary to include a *Rate Transition* block to transfer data between blocks running at different rates. If this block is not used, you will see an error during the build process.

d) ePWM Configuration for Power Switching Device Control

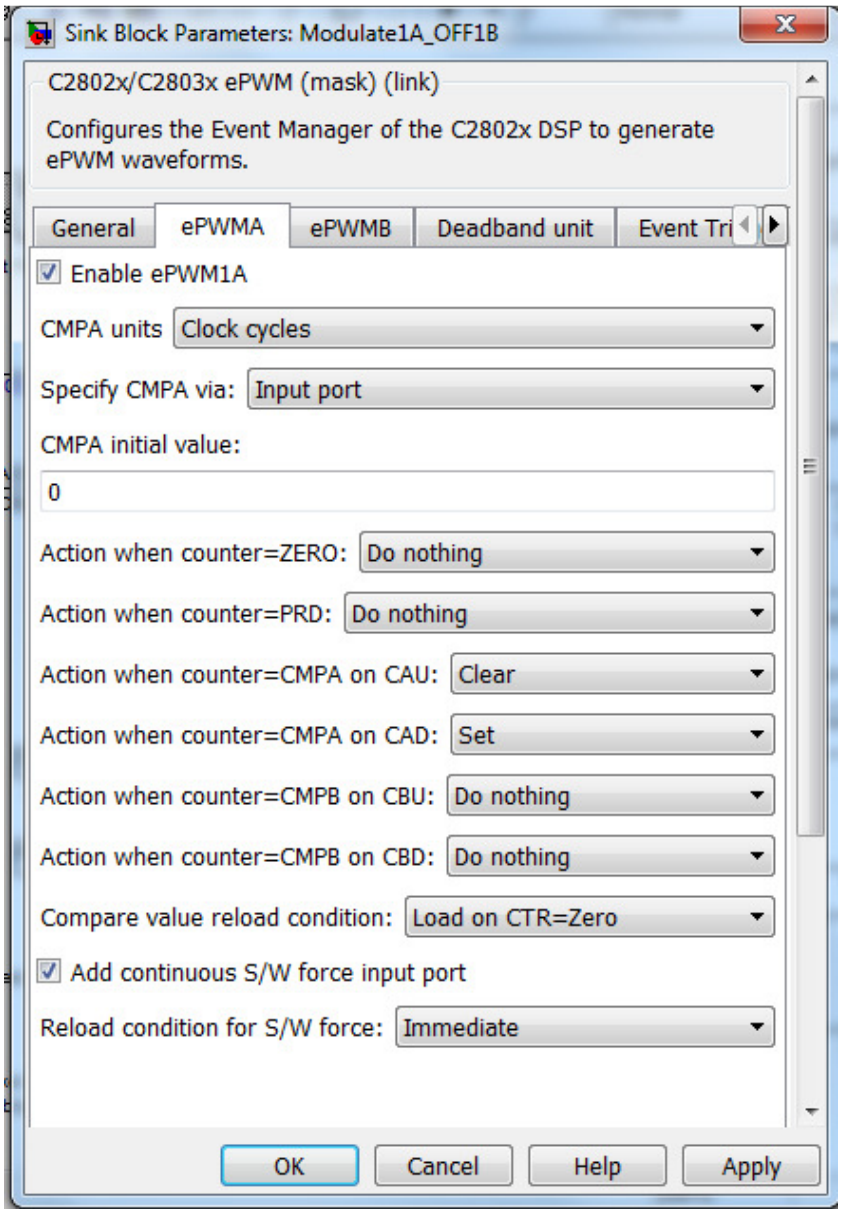
Connect your circuit to use the *Phase-U* output of the three-phase bridge. Study the schematics if needed.

DELIVERABLE 5: What ePWM modules (1-7) correspond to the modulating switch and free-wheeling diode you will use?

Add another *ePWM* block to your Simulink model. Set the *Module* appropriately. Set the period units to *Clock Cycles* specified via *dialog*, and use the *Up-Down* counting mode. Enter the number of clock cycles necessary to generate the switching frequency for your design.

It is not necessary to use any other functions on the *General* tab.

Go to the *ePWMA* tab, shown below, and enable ePWMxA. Specify the *CMPA units* as *Clock cycles* specified via *Input port*. This will add an input to the ePWM block which will allow you to dynamically change the duty cycle. Set the *Actions* appropriately, such that the duty cycle will be controlled via the value of *CMPA*. Check the box to *Add continuous S/W force input port*, and set the *Reload condition* to *Immediate*. An input of "1", with data type *unsigned integer,* will force this switch to the OFF state.



On the *ePWMB* tab, enable ePWMxB. Force the action of this switch to be OFF for all time; only the free-wheeling diode will be used.

Go to the *Trip Zone* tab. The Trip Zone function is provided in the F28035 hardware to automatically take appropriate switching actions upon detection of a fault. Check the box to enable *One-Shot TZ1*. Set the *forcing action* to *low*.

The PS21765 IGBT module has built in fault detection. Study the TMDSHVMTRPFCKIT schematics and the PS21765 datasheet.

DELIVERABLE 6: What GPIO number is this Fault output connected to? What faults does the IGBT module detect?

Unless event-triggered current sensing is desired, it is not necessary to use any other ePWM sub-modules.

Add two more *ePWM* blocks to control the other four switches in the three-phase bridge. Configure them for continuous software forcing low; they will not be used.

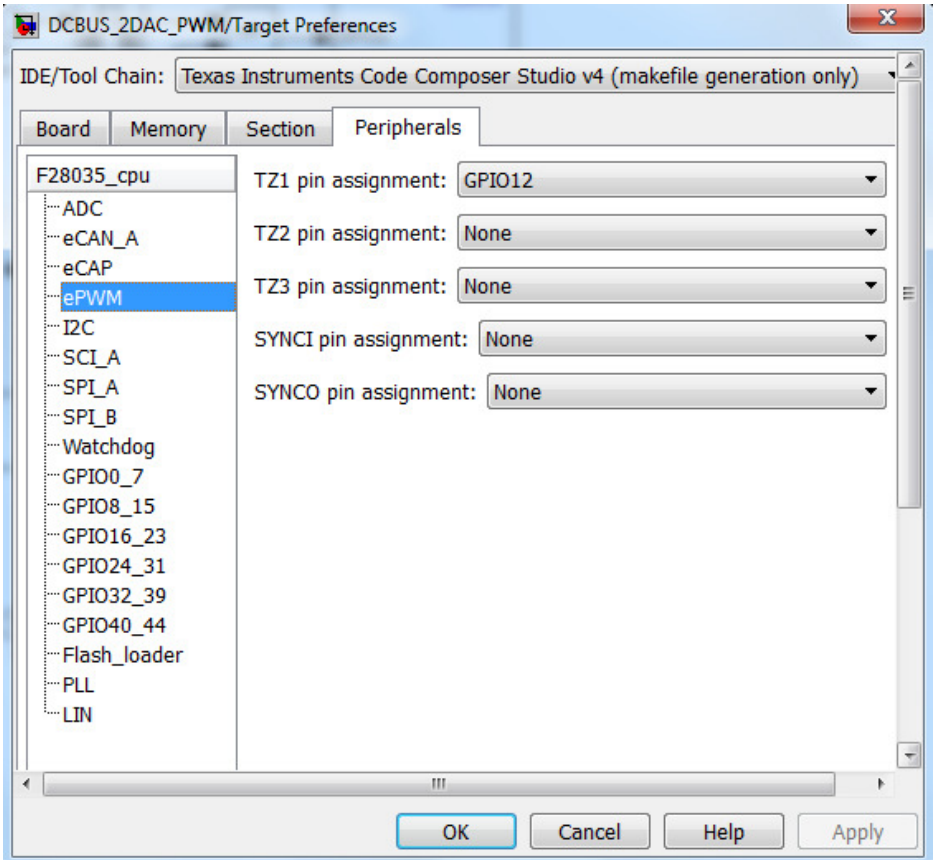
e) Output Voltage Specification

Add a *Constant* block to your model to specify the desired output voltage. Add the necessary blocks to calculate the desired duty cycle, based on the sensed DC-Bus voltage. Multiply the duty cycle by the maximum CMPA value, which you specified earlier, to generate the appropriate duty cycle input (number of clock cycles) for the ePWM module. Connect this value to the *ePWM* block.

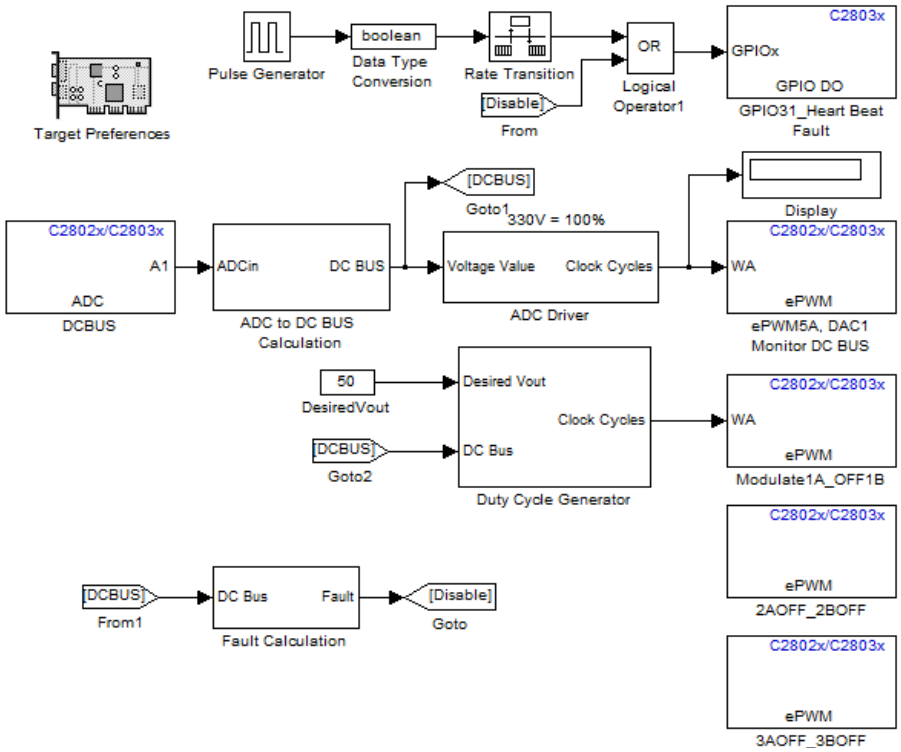
Your ePWM module should now be configured to generate a fixed switching frequency with variable duty cycle, based on the DC-Bus voltage and desired output voltage. If a fault is detected by the IGBT module, the switching action will be immediately interrupted, independent of the software. Additionally, if an over-voltage condition is detected, the switching action will be interrupted.

f) Trip Zone Pin Assignment

Edit the *Target Preferences* block to set which pin provides the input to the Trip Zone unit. Go to the Peripherals tab/ePWM, and set *Trip Zone 1* to be triggered by the active low input on *GPIO 12*.



Once your model is finished, you should have something similar to that shown below. *Note:* We are not including current sensing in this lab.



g) Build the model as you did in Lab 2. Make sure to verify the settings of the *Target Configuration* and *Simulation Configuration Parameters*. You should use the Board named "F28035", not the one named "F28035 FLASH". When the build is finished, you should see a new folder created in the location of your *Current Folder*, named "projectname_ticcs". Verify this folder contains the .out file necessary to program the device.

h) Open Code Composer Studio. Open your target configuration file and verify the type of connection (…USB…v1). Go to Run/Debug Configurations. Set the path of your target configuration file. Set the path of the .out file.

i) Debug the project. When the debugger launches, your program will be loaded onto the device. The debugger will suspend the project at the start of the main routine.

When ready to apply the control application to the hardware, continue with the instructions of Lab 6.

4) Conclusion [15 minutes]

Write about one or two things you learned in this lab that you think are important or interesting, and why.