PSIM

TUTORIAL

Processor-In-the-Loop Simulation

June 2019

With PSIM's PIL Module, one can perform processor-in-the-loop (PIL) simulation with the power stage implemented and simulated in PSIM and the control code for the TI DSP running on the actual DSP hardware.

The PIL Module includes two blocks: *PIL Block* and *PIL Block* (*InstaSPIN*). The PIL Block is for general PIL simulation, while the PIL Block (InstaSPIN) is for both PIL simulation and auto code generation specifically for TI InstaSPIN motor control algorithms. For more information on how to use the PIL Block (InstaSPIN), refer to the tutorial "Tutorial – Simulation and code generation of TI InstaSPIN using DRV8305 EVM.pdf".

This tutorial describes, in step by step, how to set up and perform PIL simulation using the general PIL block. The process involves the following steps:

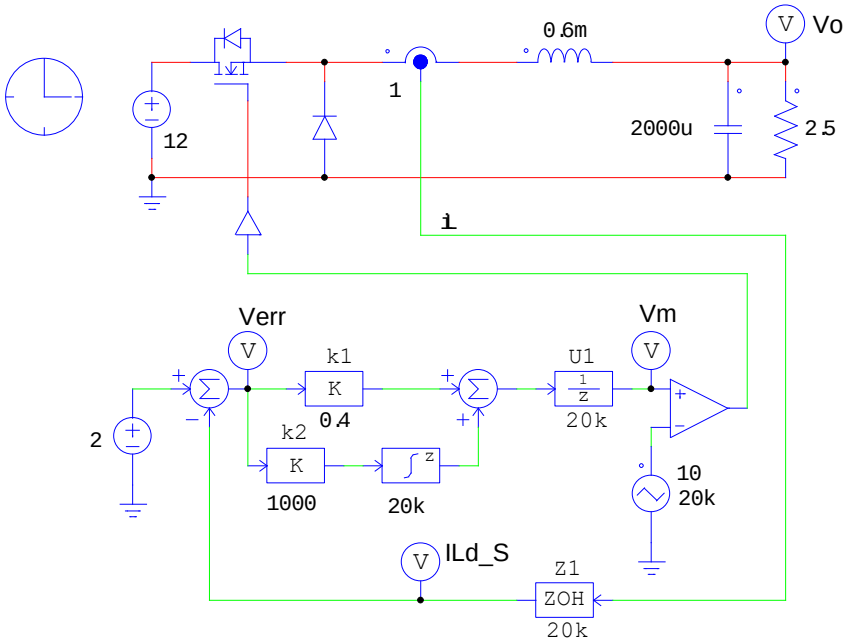
1. Preparing the code for PIL simulation
2. Setting up in PSIM
3. Setting up the hardware

To illustrate this process, we use a simple buck converter circuit as an example. This example is located in the sub-folder "examples\PIL\Buck Converter (F28335)"in the PSIM directory.

To keep the original example unchanged, we will copy the whole folder to "c:\PIL\Buck Converter (F28335)", and use it as the working folder in this tutorial.

1. Preparing the Code for PIL Simulation

The schematic of the buck converter is shown below.



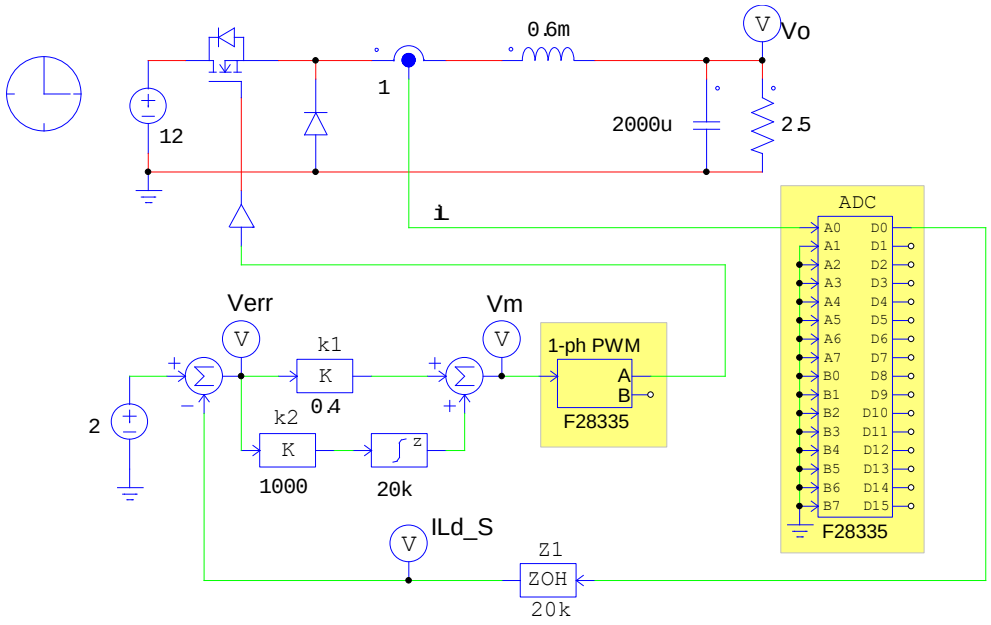
The buck converter uses average current mode control with a switching frequency of 20 kHz. The unit delay block U1 represents the one-cycle delay inherent in digital control.

The first step in PIL simulation is to prepare the code. The code can be written manually or automatically generated using PSIM's SimCoder function. In this tutorial, we will use the code generated by PSIM.

Creating Code

The circuit above is modified by adding an A/D converter after the current sensor, replacing the carrier wave and comparator with a PWM generator, and defining the hardware target. For further information on how to use SimCoder, refer to the tutorial "Tutorial - Auto Code Generation for F2833x Target.pdf".

Both the A/D converter and PWM generator are from PSIM's F2833x Target library. They simulate the behaviour of the A/D converter and PWM generator functions in the F28335 DSP hardware. The modified circuit is shown below, with the A/D converter and PWM generator highlighted in yellow.



Note that the A/D converter has a built-in limit to the input. If the input mode of the A/D converter is set to *DC*, the limit is from 0 to +3V. If the input mode is set to *AC*, the limit is from -1.5V to +1.5V. When the input is out of the range, it will be clamped to the limit. In this example, the input mode is set to DC, and the input for iL will be limited to 0 to +3V.

PSIM can simulate this circuit and generate code automatically that is ready to run on the DSP hardware. To generate the code, in PSIM, go to **Simulate** >> **Generate Code**. A sub-folder "buck converter (simcoder) (C code)" will be generated that contains the C code and other necessary files for Code Composer Studio (CCS). Make a copy of this sub-folder, and rename it to "buck converter (PIL) (C code)" for the modified code for PIL simulation.

Modifying Code for PIL Simulation

After the code is available (either written manually or from PSIM), the next step is to identify variables that need to interface with PSIM. A PIL block will be used as the interface. Interface variables include signals received from PSIM (such as measured voltages/currents) and signals sent to PSIM (such as PWM modulation signals). Typical interface variables are:

From PSIM: Variables at the A/D converter outputs

To PSIM: Variables at the PWM generator inputs, and any other internal variables for display/debugging

For example, in this example, the variable from PSIM is the A/D converter output, the variable to PSIM is the modulation signal Vm at the PWM generator input. If we wish to monitor the internal signal Verr, Verr will be an additional interface variable.

Note that PWM generator outputs cannot be the interface variables. PWM generator outputs are the actual PWM gating signals that appear at the output ports on the hardware, and they cannot be sent to PSIM. The PWM gating signals need to be generated through a circuit in PSIM.

Only two changes need to be made to the code:

- All interface variables need to be defined as global variables. Initialize the interface variables to 0.
- Statements in the code that assign values to ADC outputs need to be commented out. This is because there is no input at the actual ADC port on the hardware, and values of the interface variables must come from PSIM.

Below is the original interrupt service routine code before the change in the file "buck_converter__simcoder_.c" in the sub-folder "buck converter (simcoder) (C code)":

```
DefaultType     fGblVm = 0;
DefaultType     fGblILd_S = 0;
DefaultType     fGblVerr = 0;
interrupt void Task()
{
        DefaultType     fSUMP1, fB4, fk2, fk1, fSUM1, fZ1, fTI_ADC1, fVDC2;

        PS_EnableIntr();

        fTI_ADC1 = PS_GetDcAdc(0);
        fVDC2 = 2;
        fZ1 = fTI_ADC1;
        fSUM1 = fVDC2 - fZ1;
        fk1 = fSUM1 * 0.4;
        fk2 = fSUM1 * 1000;
        {
                staticDefaultTypeout_A = 0;
                fB4 = out_A + 1.0/20000 * (fk2);
                out_A = fB4;
        }
        fSUMP1 = fk1 + fB4;
```

```
        PS_SetPwm1RateSH(fSUMP1);
#ifdef  _DEBUG
        fGblVm = fSUMP1;
#endif
#ifdef  _DEBUG
        fGblILd_S = fZ1;
#endif
#ifdef  _DEBUG
        fGblVerr = fSUM1;
#endif
        PS_ExitPwm1General();
}
```

From the code, we identify the interface variables as: ADC output fTI_ADC1, PWM generator input fSUMP1, and fSUM1 for the error signal Verr, all highlighted in yellow.

Use a text editor to open the file "buck_converter__simcoder_.c" in the sub-folder "buck converter (simcoder) (C code)", and make the changes as discussed above. Note that interface variables need to be initialized to 0. Below is the code after the change:

```
DefaultType     fTI_ADC1 = 0, fSUM1 = 0, fSUMP1 = 0;

DefaultType     fGblVm = 0;
DefaultType     fGblILd_S = 0;
DefaultType     fGblVerr = 0;
interrupt void Task()
{
//      DefaultType     fSUMP1, fB4, fk2, fk1, fSUM1, fZ1, fTI_ADC1, fVDC2;
        DefaultType     fB4, fk2, fk1, fZ1, fVDC2;

        PS_EnableIntr();

//      fTI_ADC1 = PS_GetDcAdc(0);
        fVDC2 = 2;
        fZ1 = fTI_ADC1;
        fSUM1 = fVDC2 - fZ1;
        fk1 = fSUM1 * 0.4;
        fk2 = fSUM1 * 1000;
        {
                staticDefaultTypeout_A = 0;
                fB4 = out_A + 1.0/20000 * (fk2);
                out_A = fB4;
        }
        fSUMP1 = fk1 + fB4;
        PS_SetPwm1RateSH(fSUMP1);
#ifdef  _DEBUG
        fGblVm = fSUMP1;
#endif
#ifdef  _DEBUG
        fGblILd_S = fZ1;
#endif
#ifdef  _DEBUG
        fGblVerr = fSUM1;
```

```
#endif
        PS_ExitPwm1General();
}
```

The modified lines are highlighted in yellow. The changes are:

- Variables fTI_ADC1, fSUM1, and fSUMP1 are defined as global variables, and their definitions within the interrupt function are removed.
- The assignment of the ADC variable fTI_ADC1 is commented out.

Now compile the project of the modified code. In CCS, go to **Project** >> **Import Legacy CCS v3.3 Project**, and select "buck_converter__simcoder_.pjt" in the folder "buck converter (PIL) (C code)". Note that importing the legacy CCS v3.3 project needs to be done only once. Once it is converted to CCS v6.1 (or higher version), the project file can be opened directly even if the source code is modified later.

The next step is to define the target configuration file corresponding to the DSP hardware used. In this example, the target configuration file "F2833x.ccxml" with the TI XDS100v2 emulator has already been created and will be selected. If this file does not exist, or if you are using a different DSP or a different emulator, you must create and use your own target configuration file.

To compile, in CCS, go to **Project** >> **Build All** to build the project. This will generate the hardware executable file "buck_converter__simcoder_.out" in the folder "buck converter (PIL) (C code)/RamDebug". Note that the executable file .out can also be compiled in RamRelease and FlashRelease.

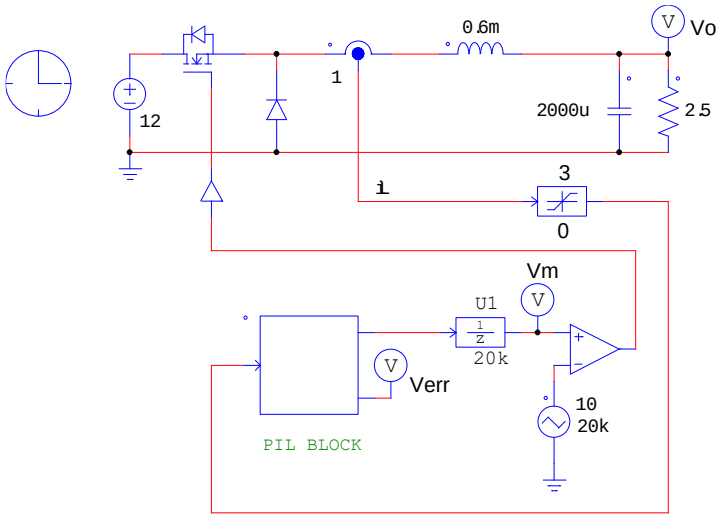
Now we can move on to the setup in PSIM.

2. Setting Up in PSIM

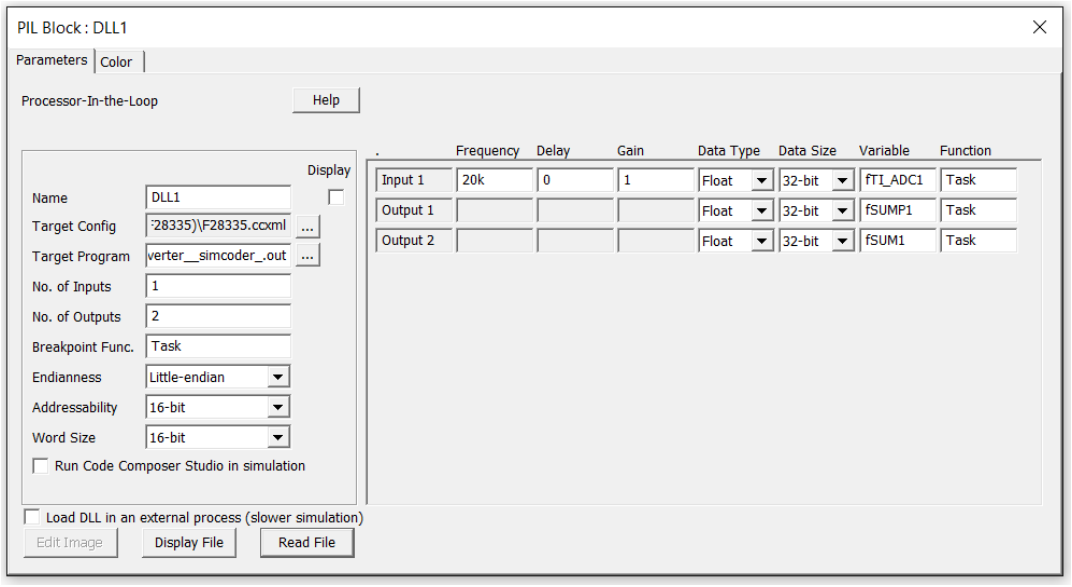
To prepare the schematic for PIL simulation, first make sure that the CCS folder path is supported. If CCS is installed in the folder "c:\TI\ccsv5" for CCS v5 or "c:\TI\ccsv6" for CCS v6, nothing needs to be done since PSIM has included these two folders already. If CCS is installed in a folder other than these two folders, it needs to be added to the PSIM path by going to **Options** >> **Set Path**, and add the CCS folder to **PSIM Search Path**.

Modify the original circuit in Page 1 by removing the control circuit, except the PWM generator and the unit delay block U1, and replacing it with a PIL block from the PSIM library under **Elements** >> **Control** >> **PIL Module**. The reason the unit delay block is present is because the PIL block does not include any delay. Also, a limiter with a range of 0 to 3V is used at the PIL block input. This is to simulate the built-in limiter of the A/D converter in case the input goes over the limit.

The modified schematic file is shown below.



The definition of the PIL block for this example is shown below:



The PIL block parameters are explained as below:

Target Config	Target configuration file .ccxml used in CCS for the specific DSP hardware. The configuration file must match the hardware connected to the computer for PIL simulation. In this example, a TI Experimenter Kit is used, and the configuration file "F28335.ccxml" that uses TI XDS100v2 emulator is provided in the schematic folder. If you are using a different DSP or different emulator, you must create and use your own hardware configuration file.
Target Program	Target hardware executable .out file. In this example, it is "buck_converter__simcoder_.out".
No. of Inputs	Number of inputs to the PIL block. There is one input for the interface variable fTI_ADC1 in this example.

No. of Outputs	Number of outputs from the PIL block. There are two outputs for the interface variables fSUMP1 and fSUM1 in this example.
Breakpoint Func.	Names of the functions for all input variables where breakpoints are set. If there are more than one function, function names are separated by comma with no space. For example, if there are two input variables vin1 and vin2, with the functions as "Task1" and "Task2" respectively, the breakpoint function will be defined as "Task1,Task2". In this example, there is only one input variable fTI_ADC1 with the function "Task", the breakpoint function will be defined as "Task".
Endianness	It refers to the order of the bytes of a word in the DSP. It can be one of the following: *Big-endian* or *Little-endian*. With Big-endian, the digits are written from the left to right, with the most significant byte of a word to the left. With Little-endian, the digits are written from the right to left, with the most significant byte of a word to the right. TI F2833x, F2803x, F2806x, and F2802x DSP use Little-endian. TI C6657 uses Big-endian.
Addressability	It refers to the way the DSP accesses memory locations. It is the smallest memory unit that can be accessed by DSP. It can be: *8-bit*, *16-bit*, *32-bit*, or *64-bit*. TI F2833x, F2803x, F2806x, and F2802x use 16-bit. TI C6657 uses 8-bit.
Word Size	Word size of the DSP. It can be: *8-bit*, *16-bit*, *32-bit*, or *64-bit*. The word size of TI F2833x, F2803x, F2806x, and F2802x is 16-bit, and the word size of TI C6657 is 32-bit.
Run Code Composer Studio in simulation	When this checkbox is checked, CCS will be launched in PIL simulation. This allows the DSP code to be debugged in CCS during PIL simulation. Note that launching CCS is not required for PIL simulation.
Frequency	Sampling frequency for input variables. In this example, it is 20 kHz for fTI_ADC1.
Delay	Sampling delay from the beginning of the PWM cycle to the moment when the ADC sampling occurs. It ranges from 0 to 1, where 0 means there is no delay, and 0.5 means that the sampling occurs at half of the PWM cycle. Note that for variables of the same sampling frequency, sampling delays must be the same. In this example, it is 0.
Gain	Gain of the input variables. It should be the same as the A/D converter gain. In this example, fTI_ADC1 = Gain * iL, and the gain is set to 1.
Data Type	Type of the variable data. It can be: *Default*, *Float*, *Integer*, *IQ1*, *IQ2*, …*IQ30*

When set to Default, the data type will be the same as the default data type in **Simulation Control** >> **SimCoder**. In this example, it is Float.

Note that variable data types defined here must match the data types used in the DSP code. Otherwise the values in PSIM and in the code will not be same and will not be correct. For example, if a variable is defined as IQ20, the data type in this dialog window must be defined as IQ20 also.

In fixed point code, the definition of "_iq" means that the variable takes the global data type definition. For example, for the code below:

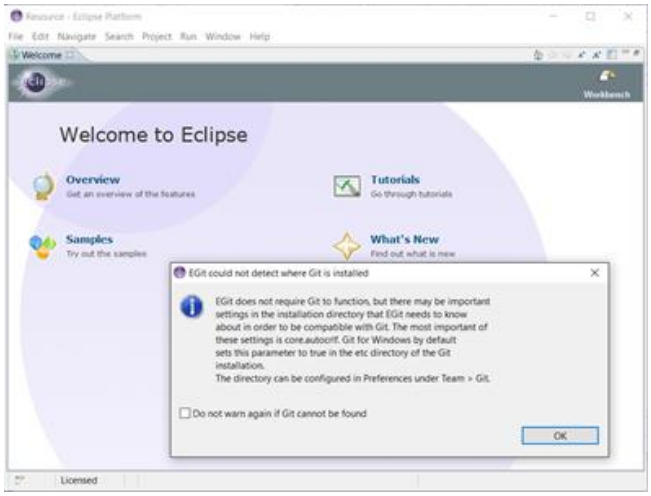
```
#define GLOBAL_Q 20
_iq fGlbWm = 0;
```

the data type of the variable fGblWm is IQ20.

Data Size	Size of the variable data. It can be: *8-bit, 16-bit, 32-bit, 64-bit* In this example, it is 32-bit.
Variable	Name of the variable that interfaces with PSIM. In this example, three variables are defined: fTI_ADC1, fSUMP1, fSUM1.
Function	Function of the variable that interfaces with PSIM. In this example, all variables are in one function "Task".

When the option **Run Code Composer Studio in simulation** is selected, wait until CCS is launched before proceed. Also, do not exit CCS in the middle of PIL simulation. To debug the code in CCS, in PSIM, select **Simulate** >> **Pause Simulation** to place the PIL simulation on hold. Then step through the code in CCS. Note that values sent from PSIM to DSP will remain unchanged when the PIL simulation is on hold. To resume simulation, in PSIM, select **Simulate** >> **Restart Simulation**.

When the option **Run Code Composer Studio in simulation** is selected, if CCS does not launch, and if you see the Eclipse Platform Resource window instead, as shown below:



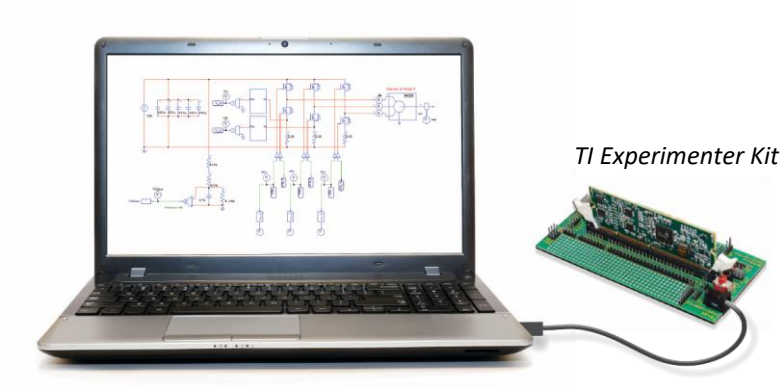
Go to **Window** >> **Open Perspective**, and select **CCS Debug**. It will open the CCS window.

Note that at the moment, the operation of the PIL simulation with multiple sampling frequencies has not been validated yet. It is recommended that, when there are multiple sampling frequencies, sample all the input variables with the highest sampling frequency, and then down sample the selected variables after ADC. For example, if there are two inputs with one at 10 kHz and the other at 1 kHz, sample both of them at 10 kHz, and then down sample the second input at 1 kHz through a 1-kHz zero-order-hold.

This completes the PSIM setup for PIL simulation.

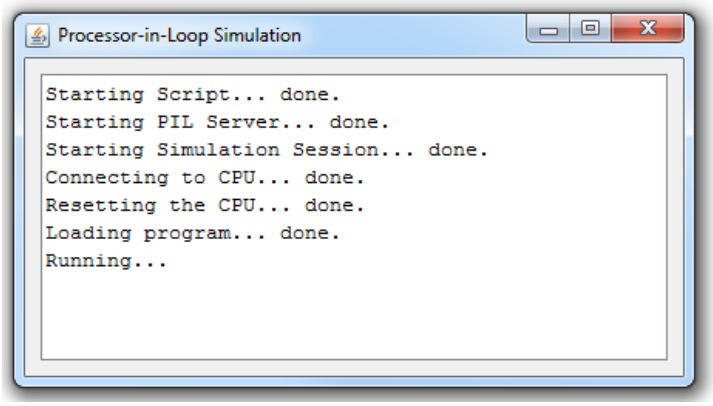
3. Setting Up the Hardware

The only hardware setup needed for PIL simulation is to connect the controller hardware to the computer. In this example, a TI Experimenter Kit is connected to the computer through a USB cable, as shown below.



TI Experimenter Kit

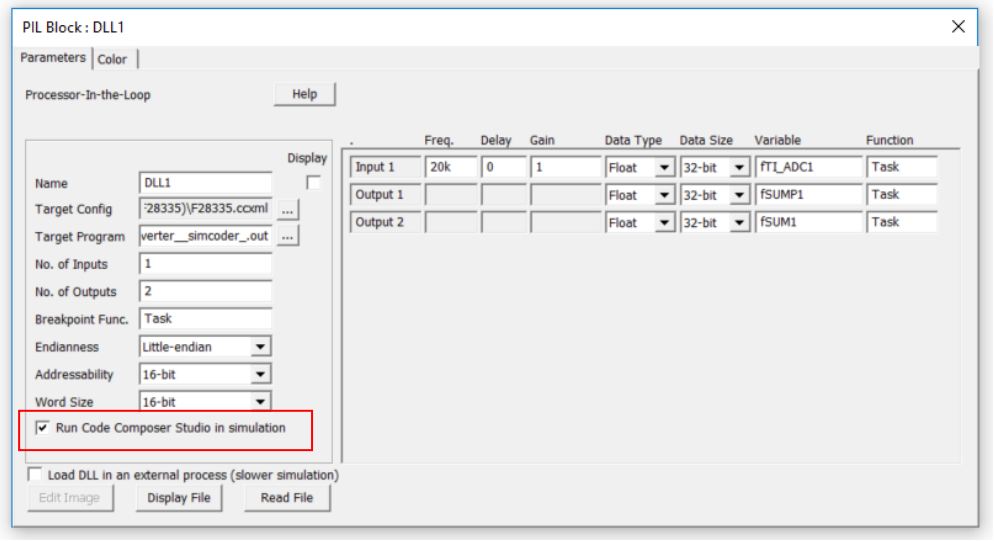
Once the controller hardware is connected, run simulation in PSIM by selecting **Simulate** >> **Run Simulation**. PSIM will establish connection with the hardware. If the connection is properly established, you will see the following dialog:



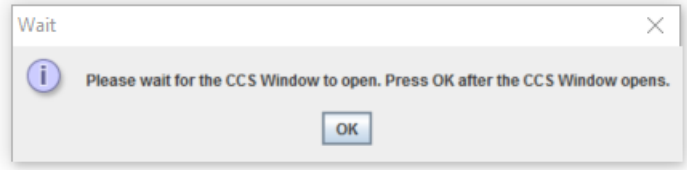
Do not close this dialog during simulation. You will be able to display waveforms during or at the end of simulation in the same way as you would in a regular PSIM simulation.

4. Debug PIL Simulation with TI Code Composer Studio

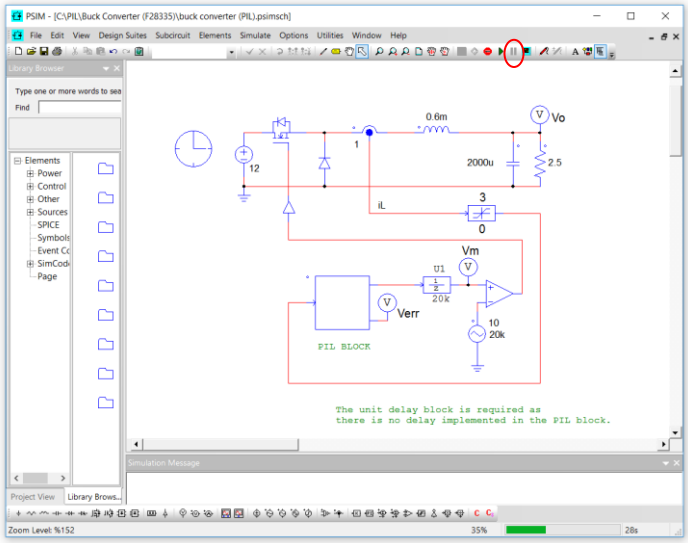
The DSP code can be debugged with Code Composer Studio during PIL simulation. To debug the code in the buck converter example, make sure that the checkbox "Run Code Composer Studio in simulation" is checked, as shown below.



Click on **Run PSIM simulation** to start the simulation. A pop-up window with the message below will appear:



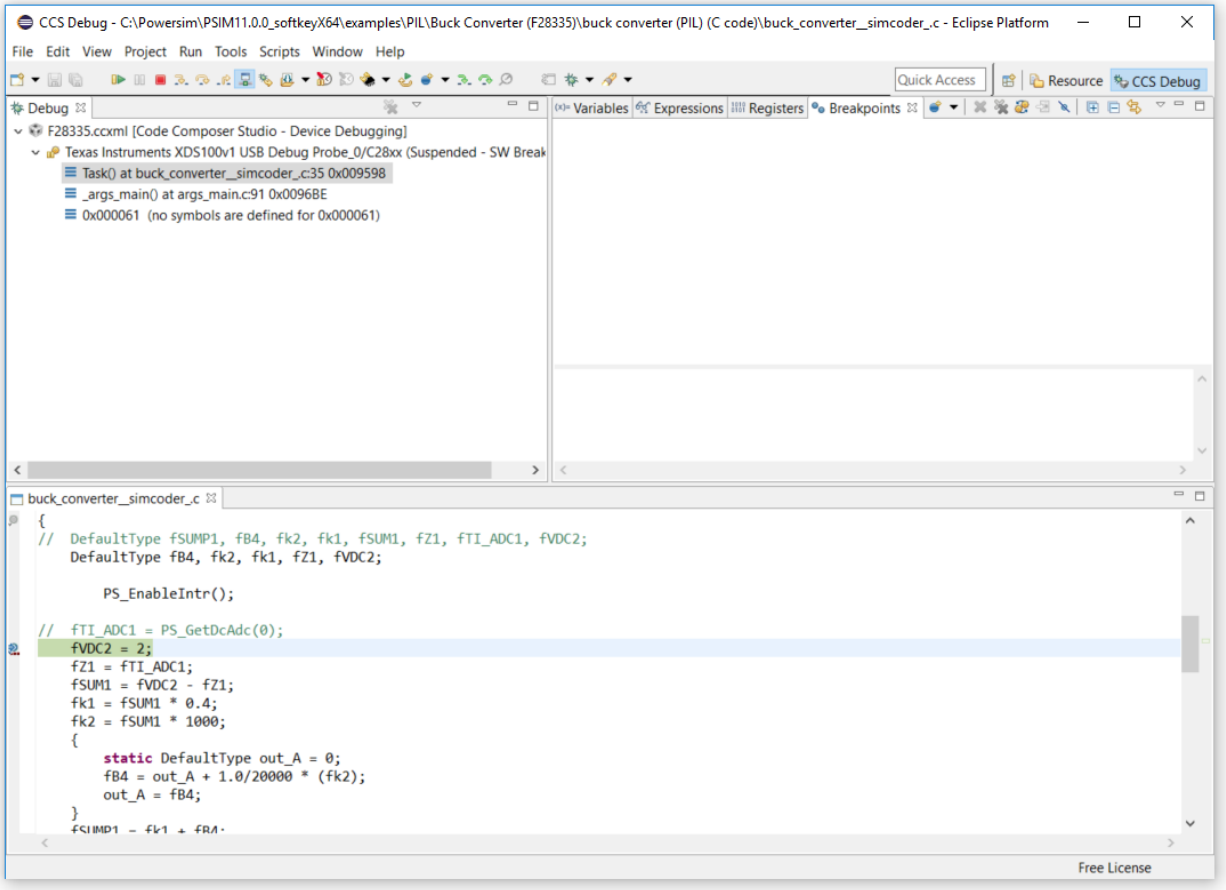
CCS will launch. Click on **OK** to close the pop-up window, and PIL simulation will start. While simulation is running, in PSIM, pause the simulation by clicking on the **Pause Simulation** icon (on the left of the SIMVIEW icon in the toolbar), as shown below.



Then go back to CCS. If you see the message:

> Can't find a source file at "C:/PIL/Buck Converter/buck converter (PIL) (C code)/buck_converter__simcoder_.c"
> Locate the file or edit the source lookup path to include its location.

Click on the **Locate File…** button, and locate the .c file in the correct folder. With the .c file loaded, the CCS window will appear as below:



You will be able to set the breakpoint, and step into or over the code, and debug the code in the same way as before. You will be able to inspect values of variables and registers.

Once debugging is done in CCS, go back to PSIM and click on **Restart Simulation** to resume the PIL simulation.

PSIM's PIL simulation is very easy to use and set up. It provides a powerful tool to validate your controller code without the physical presence of the power stage, and allows you to perform various tests that would be difficult to do on the actual hardware setup (for example, fault analysis), and speed up the development process significantly.