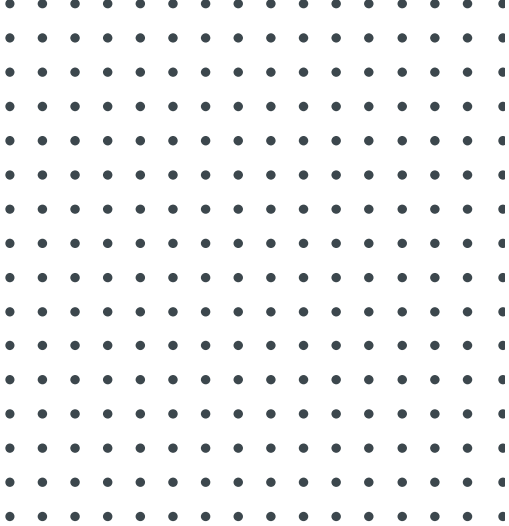


PSIM

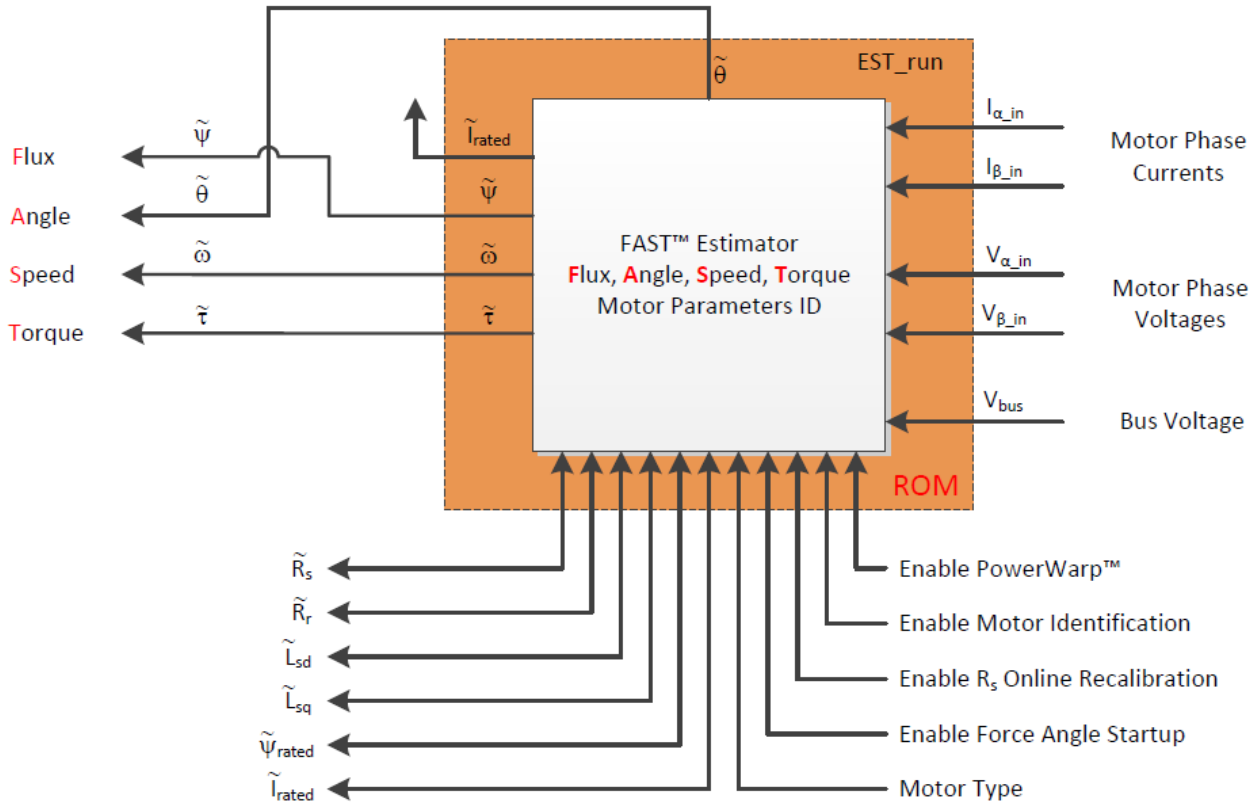
TUTORIAL

Simulation and Code Generation with TI InstaSPIN Block

November 2016

PSIM supports TI's InstaSPIN-FOC sensorless motor control algorithm in simulation and SimCoder auto code generation. With this capability, PSIM provides the easiest way for users to evaluate the performance of a motor control algorithm with InstaSPIN.

The core of the InstaSPIN algorithm is a FAST estimator that performs parameter identifications and calculates flux, angle, speed, and torque based on motor phase voltages and currents and dc bus voltage. The FAST block is illustrated below.

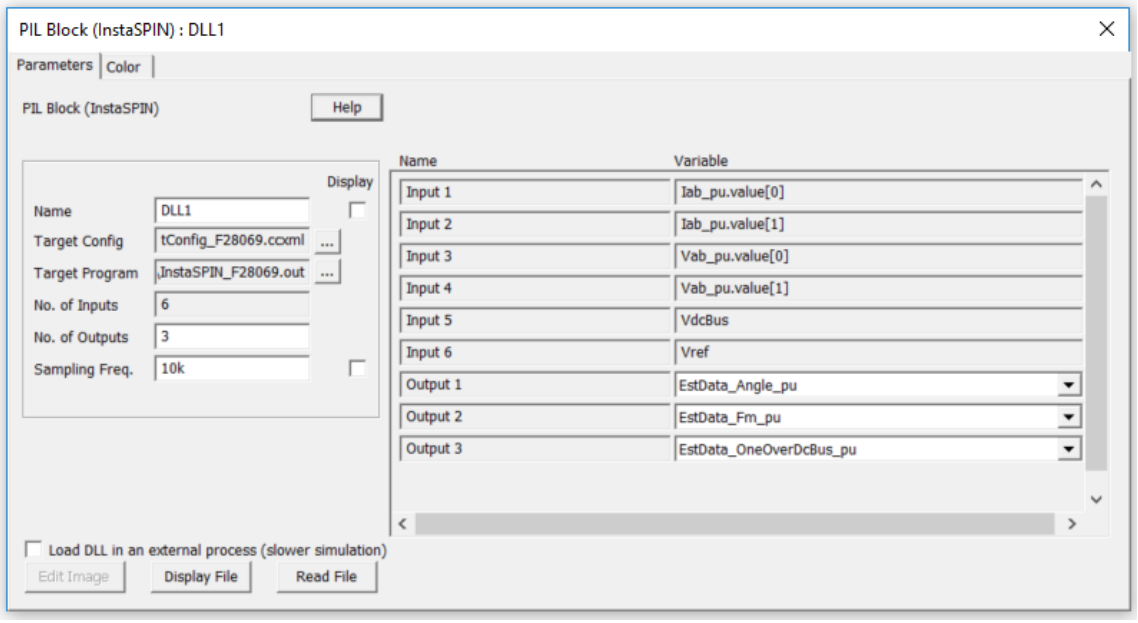


In PSIM, a block called "PIL Block (InstaSPIN)" is provided under **Elements** >> **Control** >> **PIL Module** is provided to implement the functions of InstaSPIN's FAST block. This block can be used in general simulation or in a SimCoder circuit for auto code generation for DSP hardware. At the moment, only F2806x DSP is supported. F2802x DSP can be supported upon request.

This tutorial describes how the PIL InstaSPIN block is used for simulation and for auto code generation. For further information on InstaSPIN functions, please refer to relevant TI documents.

1. PIL InstaSPIN Block Definition

The PIL InstaSPIN block has 6 inputs and 3 outputs by default. The dialog window is shown below.



The number of inputs is fixed, but the number of outputs can be changed. All inputs are in per unit, and the data format is IQ24.

The 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. By default, a F28069 configuration file "TargetConfig_F28069" in the lib subfolder is used.

Target Program — Target hardware executable .out file. Be default, the file "InstaSPIN_F28069.out" in the lib subfolder is used.

No. of Inputs — The number of inputs to the block is fixed to 6 at the moment. These inputs are:

- Iab_pu.value[0]: alpha component of the 3-phase currents after abc-alpha/beta transformation
- Iab_pu.valu e[1]: beta component of the 3-phase currents after abc-alpha/beta transformation
- Vab_pu.value[0]: alpha component of the 3-phase voltages after abc-alpha/beta transformation
- Vab_pu.value[1]: beta component of the 3-phase voltages after abc-alpha/beta transformation
- Vdcbus: DC bus voltage
- Vref: Speed reference to the controller

No. of Outputs	Number of outputs from the block. It can be changed to any number as needed. Also, each output variable can be changed through the drop-down menu. The three default outputs are: EstData_Angle_pu: Rotor angle, in per unit EstData_Fm_pu: Mechanical frequency of the motor, in per unit EstData_OneOverDcBus: Inverse of the dc bus voltage, in per unit
Sampling Freq.	Sampling frequency of the block, in Hz. It is the frequency in which the block runs.

The block can have the following outputs, all in IQ24 format and in per unit unless otherwise stated.

gIdq_pu.value[0]:	Id value of the current
gIdq_pu.value[1]:	Iq value of the current
EstData_Angle_pu:	Rotor angle
EstData_DcBus_pu:	DC bus value
EstData_ErrorCode:	Error code
EstData_Fe_pu:	Electrical frequency of the motor
EstData_Flux_pu:	Flux value
EstData_Fm_pu:	Mechanical frequency of the motor
EstData_ForceAngleDelta_pu:	Force angle delta value
EstData_FreqB0_lp_pu:	Low-pass filter numerator value in the frequency estimator (in IQ30)
EstData_FreqBeta_lp_pu:	Value used to set the pole location in the low-pass filter of the frequency estimator (in IQ30)
EstData_Fslip_pu:	Slip frequency of the motor
EstData_IdRated_pu:	Id rated current value
EstData_IdRated_indEst_pu:	Id current value used for inductance estimation of induction motors
EstData_IdRated_ratedFlux_pu:	Id current value used for flux estimation of induction motors
EstData_KRPM_to_PU_sf:	krpm to pu scale factor
EstData_Lr_pu:	Rotor inductance value (in IQ30)
EstData_Ls_d_pu:	Direct stator inductance value (in IQ30)
EstData_Ls_q_pu:	Stator inductance value in the quadrature coordinate direction (in IQ30)
EstData_Ls_max_pu:	Maximum stator inductance value from the stator inductance estimator
EstData_Ls_min_pu:	Minimum stator inductance value from the stator inductance estimator
EstData_Ls_coarse_max_pu:	Maximum stator inductance value during coarse estimation in the stator inductance estimator
EstData_MaxAccel_pu:	Maximum acceleration value used in the estimator

EstData_MaxAccel_est_pu:	Maximum estimation acceleration value used in the estimator
EstData_MaxCurrentSlope_pu:	Maximum current slope value used in the estimator
EstData_MaxCurrentSlope_epl_pu	Maximum EPL (Efficient Partial Load) current slope value used in the estimator
EstData_OneOverDcBus_pu:	Inverse of the dc bus voltage
EstData_PU_to_KRPM_sf:	pu to krpm scale factor
EstData_Rr_pu:	Rotor resistance value (in IQ30)
EstData_Rs_pu:	Stator resistance value (in IQ30)
EstData_RsOnLine_pu:	Online stator resistance value (in IQ30)
EstData_RsOnLineId_mag_pu:	Id magnitude value used for online stator resistance estimation
EstData_RsOnLineId_pu:	Online stator resistance value (in IQ30)
EstData_Speed_pu:	Mechanical frequency of the motor
EstData_Speed_krpm:	Speed value in krpm
EstData_Torque_lbin:	Torque value
EstData_Torque_Nm:	Torque value in N*m

The PIL InstaSPIN block requires additional parameters, and these parameters are defined in a parameter file called “InstaSPIN_Params.txt”. The file name is hard coded, and a different file name cannot be used. This file must be in the same folder as the schematic file.

A sample of the InstaSPIN parameter file is shown below. The content is divided into 4 sections: commonly changed parameters, rarely changed parameters, constants, and derived parameters. The first three sections need to be defined by users.

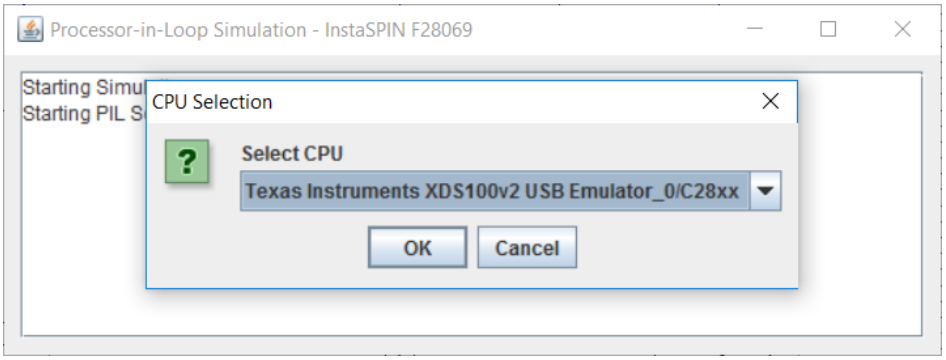
```
// InstaSPIN Parameters
// Commonly changed
//---------------------------------------------------------------------------
USER_PWM_FREQ_kHz                       = 10        // User defined
USER_NUM_PWM_TICKS_PER_ISR_TICK         = 1         // User defined
USER_MOTOR_TYPE                         = 1         // Motor dependent
USER_MOTOR_NUM_POLE_PAIRS               = 4         // Motor dependent
USER_MOTOR_RATED_FLUX                   = 0.03416464        // Motor dependent
USER_MOTOR_Rr                           = 0         // Motor dependent
USER_MOTOR_Rs                           = 0.4051206         // Motor dependent
USER_MOTOR_Ls_d                         = 0.000639871       // Motor dependent
USER_MOTOR_Ls_q                         = 0.000639879       // Motor dependent
USER_MOTOR_MAX_CURRENT                  = 5         // Motor dependent
USER_MOTOR_RES_EST_CURRENT              = 1         // Motor dependent
USER_MOTOR_IND_EST_CURRENT              = -1        // Motor dependent
USER_MOTOR_MAGNETIZING_CURRENT          = 0         // Motor dependent
USER_MOTOR_FLUX_EST_FREQ_Hz             = 20        // Motor dependent
//---------------------------------------------------------------------------
// Rarely changed
//---------------------------------------------------------------------------
USER_IQ_FULL_SCALE_CURRENT_A            = 10        // Kit dependent
USER_IQ_FULL_SCALE_VOLTAGE_V            = 24        // Kit dependent
USER_MAX_ACCEL_EST_Hzps                 = 5         // Kit dependent
USER_ADC_FULL_SCALE_VOLTAGE_V           = 66.32     // Kit dependent
USER_ADC_FULL_SCALE_CURRENT_A           = 17.3      // Kit dependent
USER_VOLTAGE_FILTER_POLE_Hz             = 714.14    // Kit dependent
```

```
NUM_ISR_TICKS_PER_CTRL_TICK                = 1        // User defined
NUM_CTRL_TICKS_PER_CURRENT_TICK            = 1        // User defined
NUM_CTRL_TICKS_PER_EST_TICK                = 1        // User defined
NUM_CTRL_TICKS_PER_SPEED_TICK              = 15       // User defined
NUM_CTRL_TICKS_PER_TRAJ_TICK               = 15       // User defined
USER_SYSTEM_FREQ_MHz                       = 90       // DSP dependent
//USER_MAX_VS_MAG_PU                       = 0.5      // Hardware defined
USER_NUM_CURRENT_SENSORS                   = 3        // Hardware defined
USER_NUM_VOLTAGE_SENSORS                   = 3        // Hardware defined
USER_IQ_FULL_SCALE_FREQ_Hz                 = 800      // Motor dependent
USER_IDRATED_DELTA                         = 0.00002            // Hardware dependent
USER_R_OVER_L_EST_FREQ_Hz                  = 300      // Motor dependent
//---------------------------------------------------------------------------
// Constants
//---------------------------------------------------------------------------
MATH_PI_VALUE                              = 3.14159265358979323846264338327 95
USER_OFFSET_POLE_rps                       = 20
USER_FLUX_POLE_rps                         = 100
USER_MAX_ACCEL_Hzps                        = 20
USER_MAX_ACCEL_EST_Hzps                    = 5.0
USER_DIRECTION_POLE_rps                    = 6.0
USER_SPEED_POLE_rps                        = 100
USER_DCBUS_POLE_rps                        = 100
USER_FLUX_FRACTION                         = 1.0
SPEEDMAX_FRACTION_FOR_L_IDENT              = 1.0
USER_POWERWARP_GAIN                        = 1.0
USER_EST_KAPPAQ                            = 1.5
IDRATED_FRACTION_FOR_L_IDENT               = 1.0
IDRATED_FRACTION_FOR_RATED_FLUX            = 1.0
//---------------------------------------------------------------------------
// Derived variables
//---------------------------------------------------------------------------
USER_ZEROSPEEDLIMIT                = 0.5 / USER_IQ_FULL_SCALE_FREQ_Hz
USER_FORCE_ANGLE_FREQ_Hz           = 2.0 * USER_ZEROSPEEDLIMIT * USER_IQ_FULL_SCALE_FREQ_Hz
USER_PWM_PERIOD_usec               = 1000.0/USER_PWM_FREQ_kHz
USER_VOLTAGE_SF                    = USER_ADC_FULL_SCALE_VOLTAGE_V/USER_IQ_FULL_SCALE_VOLTAGE_V
USER_CURRENT_SF                    = USER_ADC_FULL_SCALE_CURRENT_A/USER_IQ_FULL_SCALE_CURRENT_A
USER_VOLTAGE_FILTER_POLE_rps        = 2.0 * MATH_PI_VALUE * USER_VOLTAGE_FILTER_POLE_Hz
USER_MAX_VS_MAG_PU                 = 2.0/3.0
USER_ISR_FREQ_Hz                   = USER_PWM_FREQ_kHz * 1000.0 / USER_NUM_PWM_TICKS_PER_ISR_TICK
USER_CTRL_FREQ_Hz                  = USER_ISR_FREQ_Hz/NUM_ISR_TICKS_PER_CTRL_TICK
USER_TRAJ_FREQ_Hz                  = USER_CTRL_FREQ_Hz/NUM_CTRL_TICKS_PER_TRAJ_TICK
USER_EST_FREQ_Hz                   = USER_CTRL_FREQ_Hz/NUM_CTRL_TICKS_PER_EST_TICK
USER_MAX_CURRENT_SLOPE = USER_MOTOR_RES_EST_CURRENT/USER_IQ_FULL_SCALE_CURRENT_A/USER_TRAJ_FREQ_Hz
MAX_CURRENT_SLOPE_POWERWARP =
0.3*USER_MOTOR_RES_EST_CURRENT/USER_IQ_FULL_SCALE_CURRENT_A/USER_TRAJ_FREQ_Hz
USER_ISR_PERIOD_usec               = USER_PWM_PERIOD_usec * USER_NUM_PWM_TICKS_PER_ISR_TICK
USER_CTRL_PERIOD_usec              = USER_ISR_PERIOD_usec * NUM_ISR_TICKS_PER_CTRL_TICK
USER_CTRL_PERIOD_sec               = USER_CTRL_PERIOD_usec/1000000.0
MAX_NEGATIVE_ID_REF_CURRENT_A = -0.5 * USER_MOTOR_MAX_CURRENT
```

2. Simulation with PIL InstaSPIN Block

Below are the steps to simulate a circuit with the PIL InstaSPIN block:

- Place the block in the circuit and connect it with the rest of the circuit.
- Define the InstaSPIN parameters and other parameters.
- Connect the computer to a DSP hardware with the F28069M DSP. Note that the DSP must be the M version that supports InstaSPIN.
- Run the simulation. A dialog window as shown below will appear.



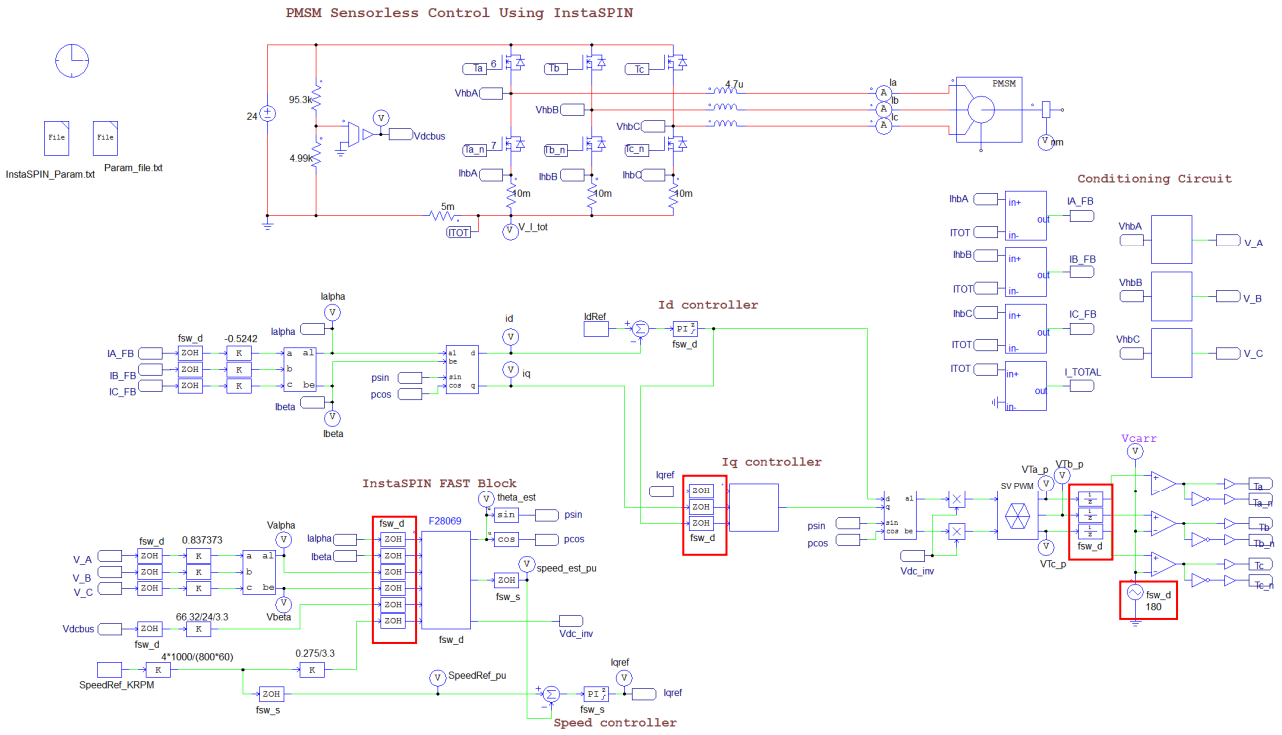
Select the correct CPU type, and click on OK to continue. The simulation will run. Note that in the middle of the simulation, do not close the "Processor-in-Loop Simulation" popup window.

An InstaSPIN example is provided in the folder "examples\PIL\PMSM InstaSPIN Lab11 (F28069)". This example is based on Lab 11 of TI's Motorware InstaSPIN examples for F28069 for the DRV8312 kit. The PSIM schematic of the example is shown below.

Notice that beside the parameter file "InstaSPIN_param.txt", another parameter file "Param_file.txt" is used in the schematic to define parameters used in other part of the circuit. The file "InstaSPIN_param.txt" has a higher priority than the file "Param_file.txt", which means that parameters defined in "InstaSPIN_param.txt" can be used in "Param_file.txt". To define the file priority, in the parameter file dialog window, select **Edit** >> **Priority**.

In this example, 3-phase ac voltages and currents as well as dc bus voltage are measured. The ac voltages and currents are converted to the alpha/beta frame through the Clarke transformation. The alpha/beta quantities, together with the dc bus voltage and speed reference, are sent to the PIL InstaSPIN block, which generates estimated rotor angle theta, estimated speed, and the inverse of the dc bus voltage.

The estimated speed is used in the speed control loop to generate the Iq reference. The outputs of the Id and Iq control loops are sent to the inverse Park transformation block, and the outputs are then used to generate PWM gating signals.

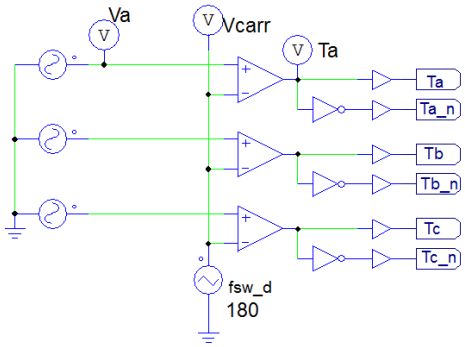


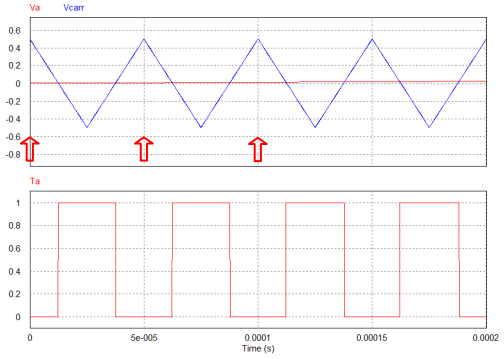
There are two sampling rates in this circuit, fsw_d for the inner current loop (set to 20kHz in "Param_file.txt"), and fsw_s for the outer speed loop (fsw_s = fsw_d/15).

Note that in this schematic, zero-order-hold blocks of fsw_d are connected to all inputs of the PIL InstaSPIN block and the Iq controller C block. This is needed to force these two blocks to compute in the sampling rate of fsw_d.

Three unit delay blocks at the input of the PWM generation circuit (highlighted in red) are used to model the one cycle delay inherent in digital control.

Also, the PWM carrier waveform phase delay must be set to 180 deg. This is very important as otherwise the circuit will not work. This is because the bottom switch currents are measured and sampled by the DSP ADC. This means that, at the moment of sampling, the bottom switch must be on and the top switch must be off. In order to achieve this, the carrier waveform must be phase shifted by 180 deg. To understand this, let's look at the PWM circuit by itself, as shown below on the left:





The carrier waveform is a triangular waveform from -0.5 to 0.5, with the phase delay set to 180 deg. and a switching frequency of 20kHz. The simulation waveforms are shown on the right. In PSIM simulation, the beginning of sampling periods starts naturally at the time of zero and at each integer number of the sampling period. In this case, the beginning of the periods starts at 0, 50us, 100us, etc., as marked by the red arrow.

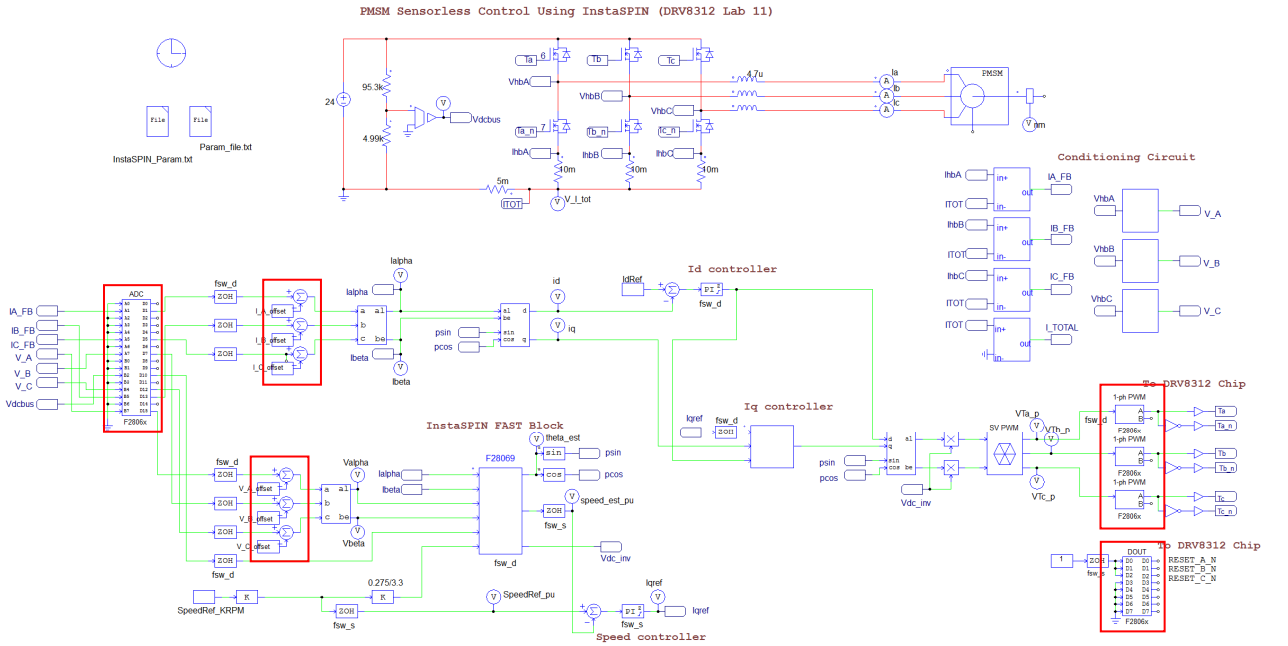
If A/D conversion is performed at the beginning of each period, with the settings above, the top switch gating signal Ta will be low, and the bottom switch gating signal Ta_n will be high. This is exactly what is needed for DSP implementation. If the phase delay of the carrier wave is set to 0 as is done typically, at the beginning of each period, Ta will be high and Ta_n will be low. The A/D conversion result for the current will be all 0 since the bottom switch is not conducting.

To summarize, the carrier wave phase delay should be set to 180 deg. if the bottom switch currents are sampled, and the phase delay should be set to 0 if the top switch currents are sampled.

3. Auto Code Generation with the PIL InstaSPIN Block

With SimCoder and F2806x/F2802x Target, PSIM can automatically generate code that is ready to run on an InstaSPIN-enabled DSP hardware.

An example is provided in the folder “examples\SimCoder\F2806x Target\TI PMSM InstaSPIN Lab11”. This example is based on Lab 11 of TI’s Motorware InstaSPIN examples for F28069 for the DRV8312 kit. The PSIM schematic is shown below.



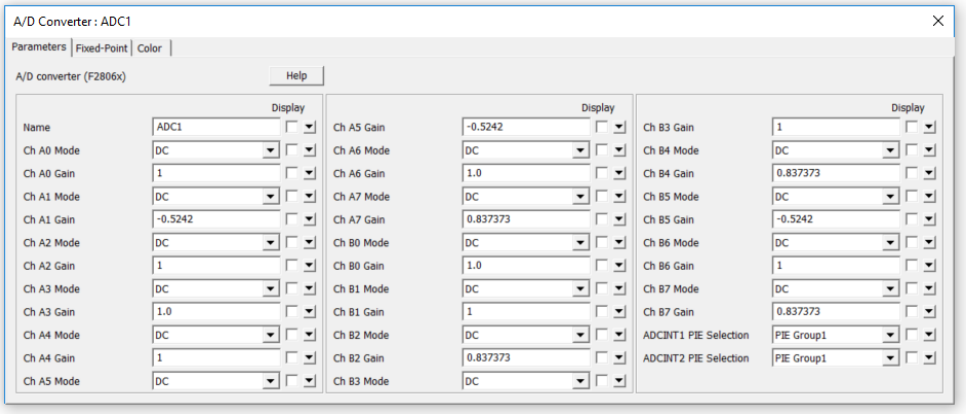
In the schematic, the zero-order-hold blocks at the inputs of the PIL InstaSPIN block and the Iq controller block (except the Iqref input) are no longer needed as SimCoder will detect the sampling rates of these blocks automatically.

Also, since ac voltages and currents at the ADC inputs have dc offset due to conditioning circuits, to restore the voltages/currents back to ac, the dc offsets are removed after the ADC and ZOH blocks, as highlighted in red.

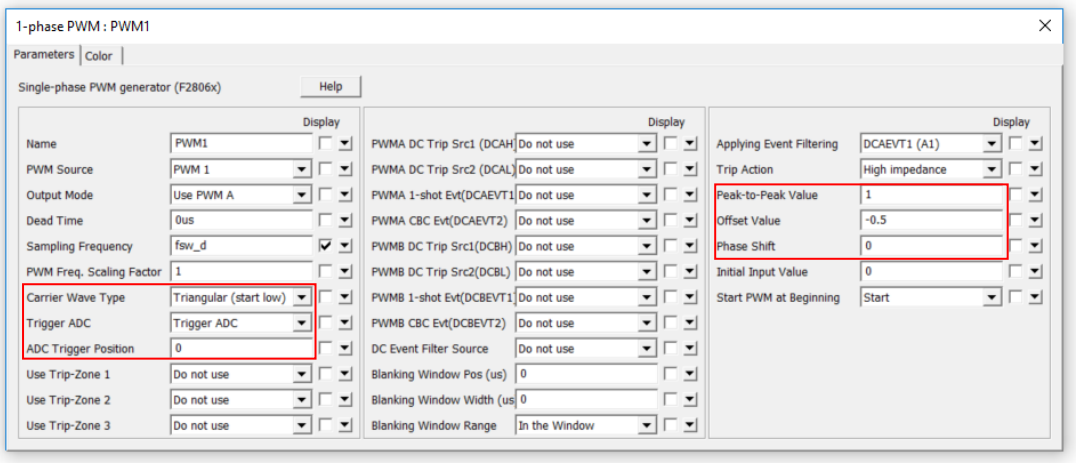
The offset values from the TI Lab 11 code are used here. Note that the offsets are not exactly equal among the three phases in the hardware circuit, but they are equal in simulation. To address the discrepancy, a flag called “flag_simulation” is defined in the parameter file “param_file.txt”. When the flag is set to 1, identical offset values are used. When the flag is 0, the actual hardware offset values are used.

To perform auto code generation, ADC, PWM, and digital output blocks, as highlighted in red, are used in the circuit. They simulate the functions of the actual F28069 ADC, PWM, and digital output hardware peripheral blocks.

The figure below shows the ADC block definition. The modes of all the ADC channels are set to DC as all incoming voltage and current signals are dc quantities.



The figure below shows the PWM block definition for the Phase A PWM signal Ta. The definitions of other two phases are the same except that the parameter “Trigger ADC” is set to *Do not trigger ADC*.



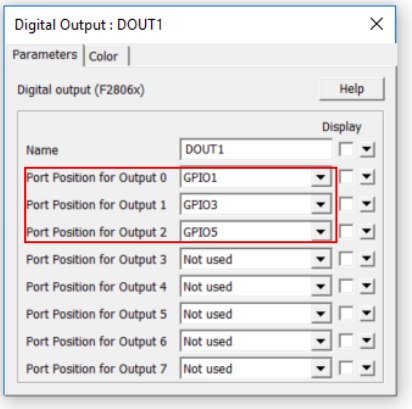
In the definition, the parameters “Carrier Wave Type” is set to *Triangular (start low)*, “Trigger ADC” is set to *Trigger ADC*, and “ADC Trigger Position” is set to 0. This means that the PWM

generator will trigger ADC, and ADC will perform A/D conversion at the beginning of the sampling period. The setting *Triangular (start low)* means that when the modulation wave is greater than the carrier wave, the PWM signal will be low. This will give a low signal to the top switch and a high signal to the bottom switch, exactly what is required for bottom switch current sampling.

The carrier waveform is defined to be a triangular waveform from -0.5 to +0.5.

For the DRV8312 chip, it needs not only 3 PWM signals PWMA, PWMB, and PWMC for the top switches, but also 3 reset signals RESET_A_N, RESET_B_N, and RESET_C_N which should be set to high all the time. These three reset signals are generated through the digital output block. Since it does not have to run at the frequency of fsw_d, it is set to run at a slower rate of fsw_s.

The definition of the digital output block is shown below. GPIO ports GPIO1, 3, and 5 are used as required by the DRV8312 hardware.

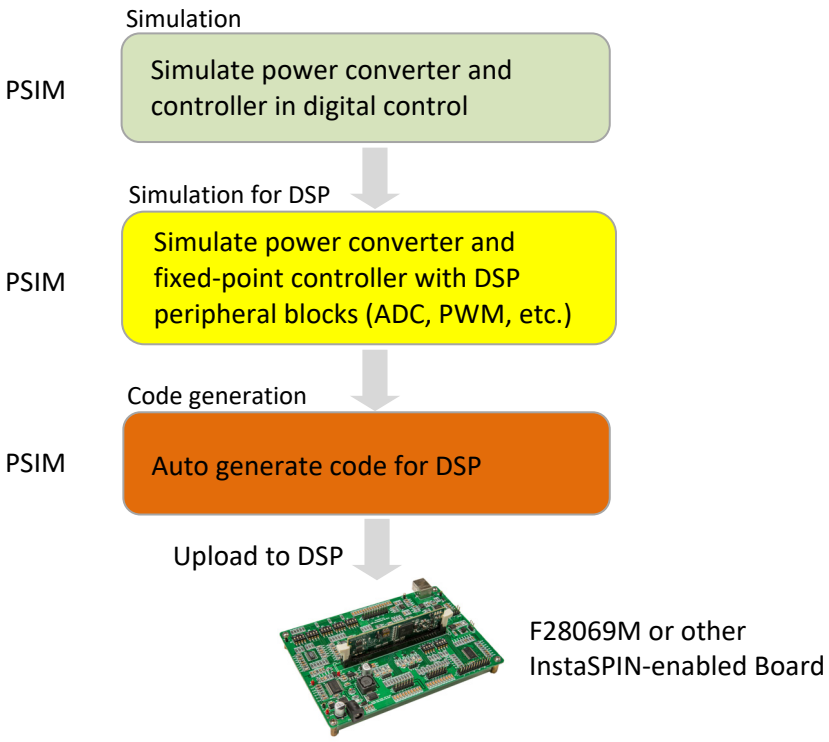


To run simulation, select **Simulate** >> **Run Simulation** in the same way as with other circuits.

To generate code automatically, select **Simulate** >> **Generate Code**. This will generate the code that is ready to run on the DRV8312 hardware kit. Launch Code Composer Studio, and navigate to the schematic folder and enter the subfolder "… …(C code)". Load the project, and compile and run the code.

For more information on how to generate code and run on F2806x DSP, please refer to the tutorial "Tutorial – Auto code generation for F2806x Target.pdf".

With PSIM's automatic code generation capability, one can perform both simulation and auto code generation for rapid control prototyping and hardware implementation in one seamless workflow, as shown below:



The integrated environment greatly speeds up the development process and helps to reduce development cost and time-to-market.