Application Note

MATLAB™ Model-Based Design Using C2000™ Real-Time Microcontrollers

Aditya Dholakia, Jayakarthigeyan Prabakar, and Aditya Padmanabha

ABSTRACT

With industrial and automotive control applications becoming more complex, decoupling the high-level control algorithm development from low-level device specific driver development using code generation tools such as Embedded Coder from MathWorks is useful. With the advent of code generation tools, it becomes of paramount importance to evaluate the ease-of-use, efficiency and performance of the generated code for a real time microcontroller such as C2000™. This application note provides detailed insight on usage of MATLAB's C2000 Microcontroller Blockset for C2000 covering aspects from getting started to the best practices and performance evaluation. To showcase the entire use-case, an eCompressor reference design TIDM-02012 is chosen for evaluation.

Table of Contents

List of Figures

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.
MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc.
All trademarks are the property of their respective owners.

1 Introduction

The application note focuses on enabling performance optimization using model-based design code generation on MATLAB®. To understand the performance optimization, a reference example of TIDM-02012 eCompressor using model-based design code generation tools of MATLAB is chosen. The implementation, optimization and performance evaluation of the reference design will be discussed.

1.1 Getting Started

The required software toolset for enabling the model-based design are TI's Code Composer Studio IDE, C2000Ware Software Development Kit (SDK), MATLAB tools such as Simulink®, MATLAB Coder, Embedded Coder and C2000 Microcontroller Blockset.

Links to download and install the same are given below.

Texas Instruments tools:

- Code composer studio: CCSTUDIO IDE
- C2000Ware SDK: C2000WARE Software development kit (SDK)
- C2000Ware MotorControl SDK:C2000WARE-MOTORCONTROL-SDK Software development kit (SDK)

MathWorks tools:

- MATLAB: Download MATLAB, Simulink, Stateflow and Other MathWorks Products Installation of Simulink, MATLAB Coder, Embedded Coder and C2000 Microcontroller Blockset is required to run the model.
- C2000 Microcontroller Blockset: C2000 Microcontroller Blockset - MATLAB

Installation guidelines for the C2000 Microcontroller Blockset to be integrated with MATLAB is given in the installation page of the blockset. Training video for the C2000 Microcontroller Blockset can be found here.

Note

Since Embedded Coder, C2000 Microcontroller Blockset as well as C2000's compiler is further updated for optimal code generation with every release, it is recommended to use the latest available tools to get the best possible efficiency for code generation and performance.

2 Model-Based Design of eCompressor

TIDM-02012 model-based design example is an eCompressor hardware with target application ranging from air-conditioning, heating and traction drives. With the automotive industry transitioning to electric vehicles, the heating and cooling systems of the car use the permanent magnet synchronous motor (PMSM) to drive the eCompressor. The reference design available with the motor control SDK of C2000 demonstrates the controlling of eCompressor motor using field-oriented control (FOC) without a position sensor. The focus of this application note is not to understand the functionality of an eCompressor hardware, but to understand the flow of enabling a model-based design for given target application. To demonstrate the same, the same example available within the motor control SDK is migrated to a model-based design on MATLAB & Simulink to generate the code using the blocks available in MATLAB tools.

The current challenges in the conventional way of developing the application using manually written code is the coding expertise requirement, time taken to build the end application and debuggability. These challenges can be addressed by taking up the model-based design work flow to build the application. Model-based design allows ease in development with limited knowledge on coding the microcontroller, reduction in development time and enables a graphical approach that allows you to visualize the application code and flow. The model can be simulated at any point to get an instant view of system behavior in Simulink.

For detailed information on the high-voltage HEV/EV HVAC eCompressor motor control reference design, see the TIDM-02012 product page and the *High-Voltage HEV and EV HVAC eCompressor Motor Control Reference Design*

2.1 General Texas Instruments High Voltage Evaluation (TI HV EVM) User Safety Guidelines

Always follow TI's setup and application instructions, including use of all interface components within their recommended electrical rated voltage and power limits. Always use electrical safety precautions to help ensure your personal safety and those working around you. For more information, contact TI's Product Information Center https://support/ti./com.

Save all warnings and instructions for future reference.

WARNING

Failure to follow warnings and instructions may result in personal injury, property damage or death due to electrical shock and burn hazards.

The term TI HV EVM refers to an electronic device typically provided as an open framed, unenclosed printed circuit board assembly. It is *intended strictly for use in development laboratory environments, solely for qualified professional users having training, expertise and knowledge of electrical safety risks in development and application of high voltage electrical circuits. Any other use and/or application are strictly prohibited by Texas Instruments.* If you are not suitable qualified, you should immediately stop from further use of the HV EVM.

1. Work Area Safety
   a. Keep work area clean and orderly.
   b. Qualified observer(s) must be present anytime circuits are energized.
   c. Effective barriers and signage must be present in the area where the TI HV EVM and its interface electronics are energized, indicating operation of accessible high voltages may be present, for the purpose of protecting inadvertent access.
   d. All interface circuits, power supplies, evaluation modules, instruments, meters, scopes and other related apparatus used in a development environment exceeding 50Vrms/75VDC must be electrically located within a protected Emergency Power Off EPO protected power strip.
   e. Use stable and nonconductive work surface.
   f. Use adequately insulated clamps and wires to attach measurement probes and instruments. No freehand testing whenever possible.
2. Electrical Safety
   As a precautionary measure, it is always a good engineering practice to assume that the entire EVM may have fully accessible and active high voltages.
   a. De-energize the TI HV EVM and all its inputs, outputs and electrical loads before performing any electrical or other diagnostic measurements. Revalidate that TI HV EVM power has been safely de-energized.
   b. With the EVM confirmed de-energized, proceed with required electrical circuit configurations, wiring, measurement equipment connection, and other application needs, while still assuming the EVM circuit and measuring instruments are electrically live.

c. After EVM readiness is complete, energize the EVM as intended.

WARNING

While the EVM is energized, never touch the EVM or its electrical circuits, as they could be at high voltages capable of causing electrical shock hazard.

3. Personal Safety
   a. Wear personal protective equipment (for example, latex gloves or safety glasses with side shields) or protect EVM in an adequate lucent plastic box with interlocks to protect from accidental touch.

Limitation for safe use:

EVMs are not to be used as all or part of a production unit.

2.2 Block Diagram

Figure 2-1 shows the model-based design for the eCompressor motor control reference design.

The Simulink subsystem titled *TMS320F280039C* contains the C2000 device driver blocks and control blocks. Source code for the MCU of the subsystem can be automatically generated using C2000 Microcontroller Blockset and Embedded Coder. The block titled *Inverter and Motor – Plant Model* contains Simulation plant model for the inverter and motor with similar configured parameters as present in the actual reference design hardware. You are expected to update the model based on the hardware specification as per the application.

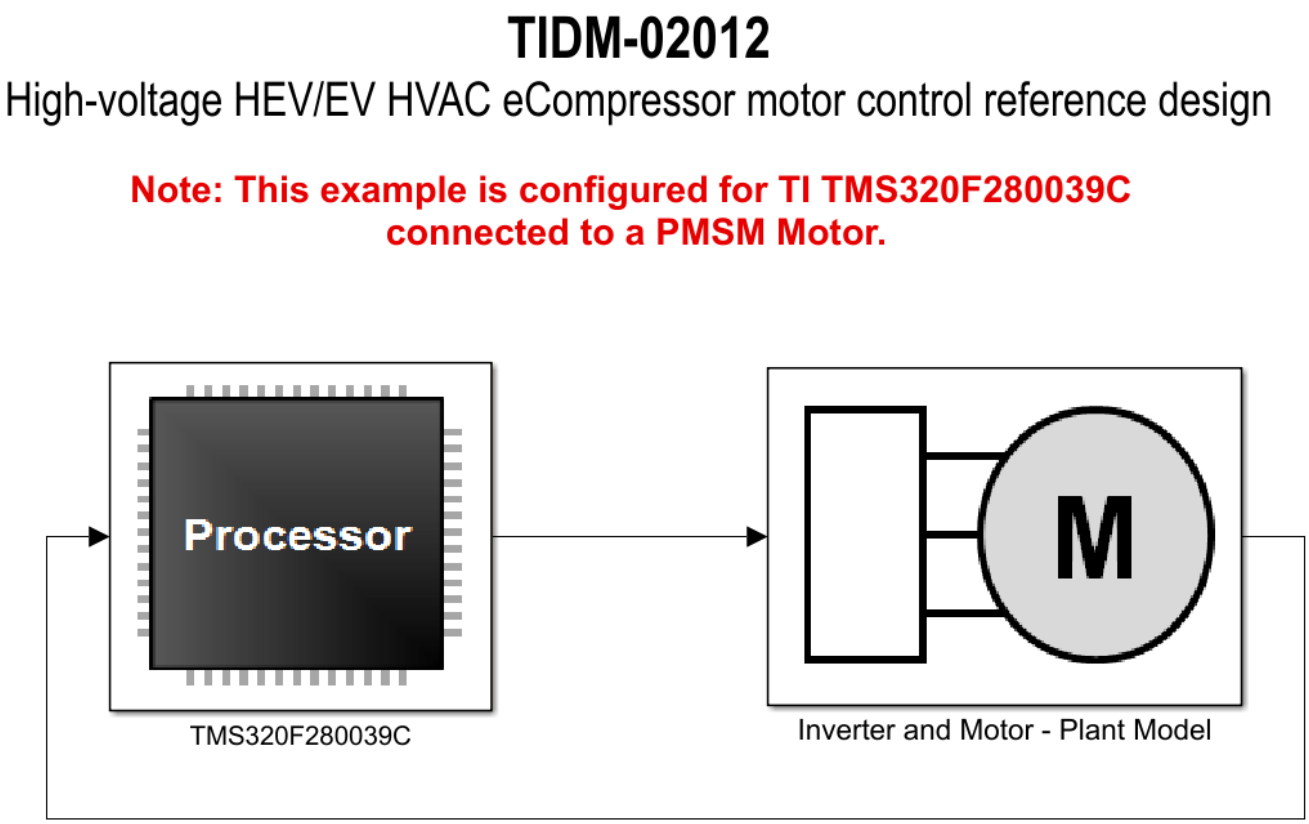

Figure 2-1. Model-Based Design of eCompressor

Going into the *TMS320F280039C* block, the complete control loop algorithm is implemented as shown in Figure 2-2.

Note

The processor TMS320F280039C model contains the blocks only for which the code is generated using Embedded Coder. Plant model is for simulation only of the Inverter and Motor that will not generate application code.

The operation is primarily divided in 3 parts.

- Sensing and communication
- Control loop
- Duty cycle control through PWMs

Sensing of motor phase currents I_a, I_b and I_c is done using the ADC available within Sensor Driver Blocks subsystem in the model as shown in the Figure 2-2. The interrupt block contains configuration for 2 interrupts – ADC and SCI. ADC interrupt is executed at a faster rate of *15 kHz* for control operation, whereas, SCI interrupt is running every *0.1 sec* at a slower rate to read changes in reference speed and on/off controls for the motor. The interrupt configurations are present in the *Sensor Driver Blocks* and *SCI Interrupt block* within the *TMS320F280039C* block.

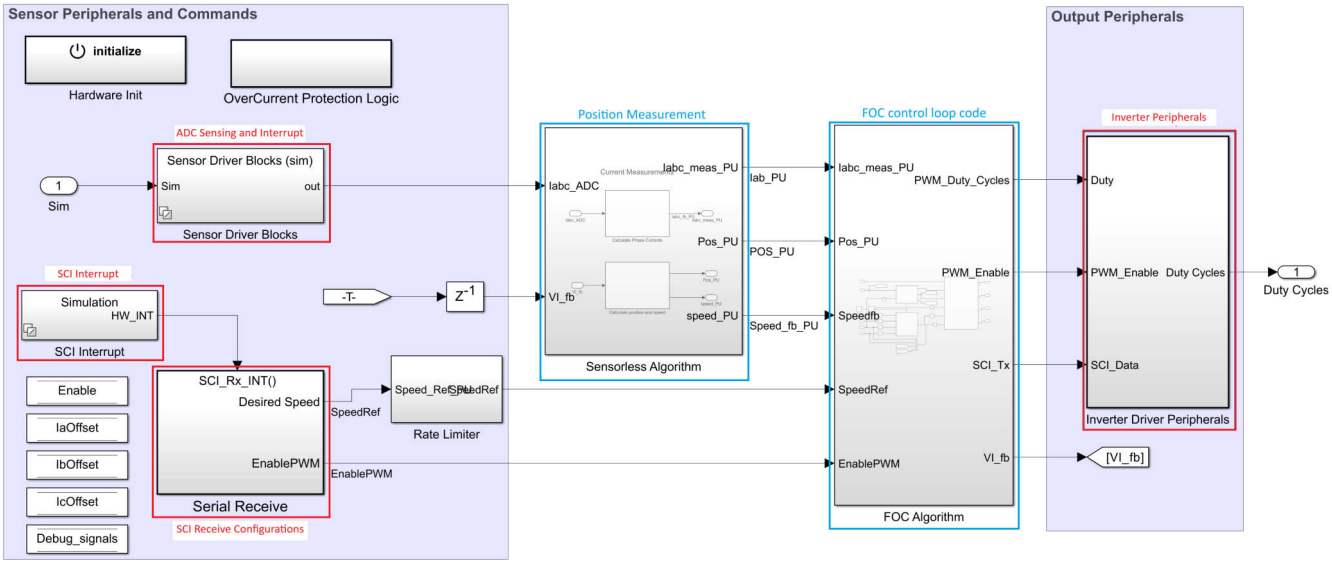

Figure 2-2. TMS320F280039C Model Block

On top of the code to be executed on the device, to ensure that the control is correctly implemented, there are additional requirements such as to enable / disable the motor, configure the speed of the motor and to read back the parameters from the actual system such as the speed, direct and quadrature axis currents - I_d and I_q, motor phase currents I_a and I_b and the position calculated. For the values to be read, the SCI module on board is used to send the selected data to the control host. User has complete control over the operations of motor using the control host model. The SCI module for the receiver is implemented within the *Serial* Receive block as shown in Figure 2-2.

The SCI module on the control host is implemented within the *Serial Communication* block in the file TIDM_02012_control_host.six file as shown in Figure 2-3.

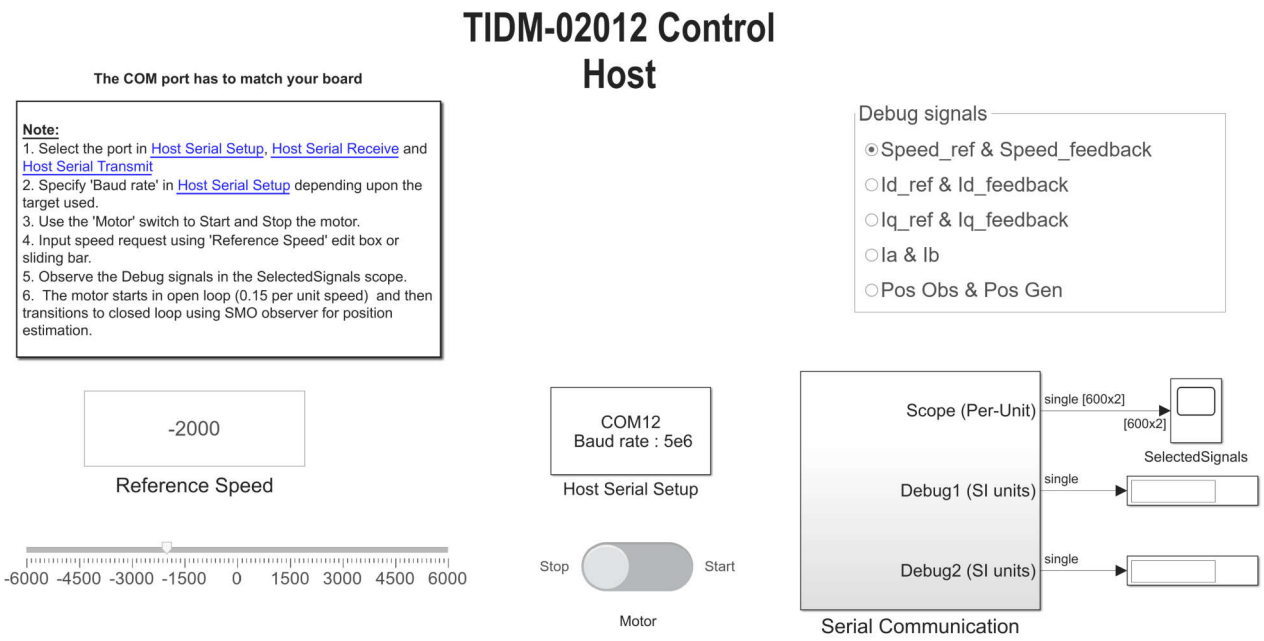

Figure 2-3. TIDM-02012 Control Host Block

The pictorial representation of how the models work is shown in Figure 2-4. The model runs on the hardware, whereas the host system runs the control host application.

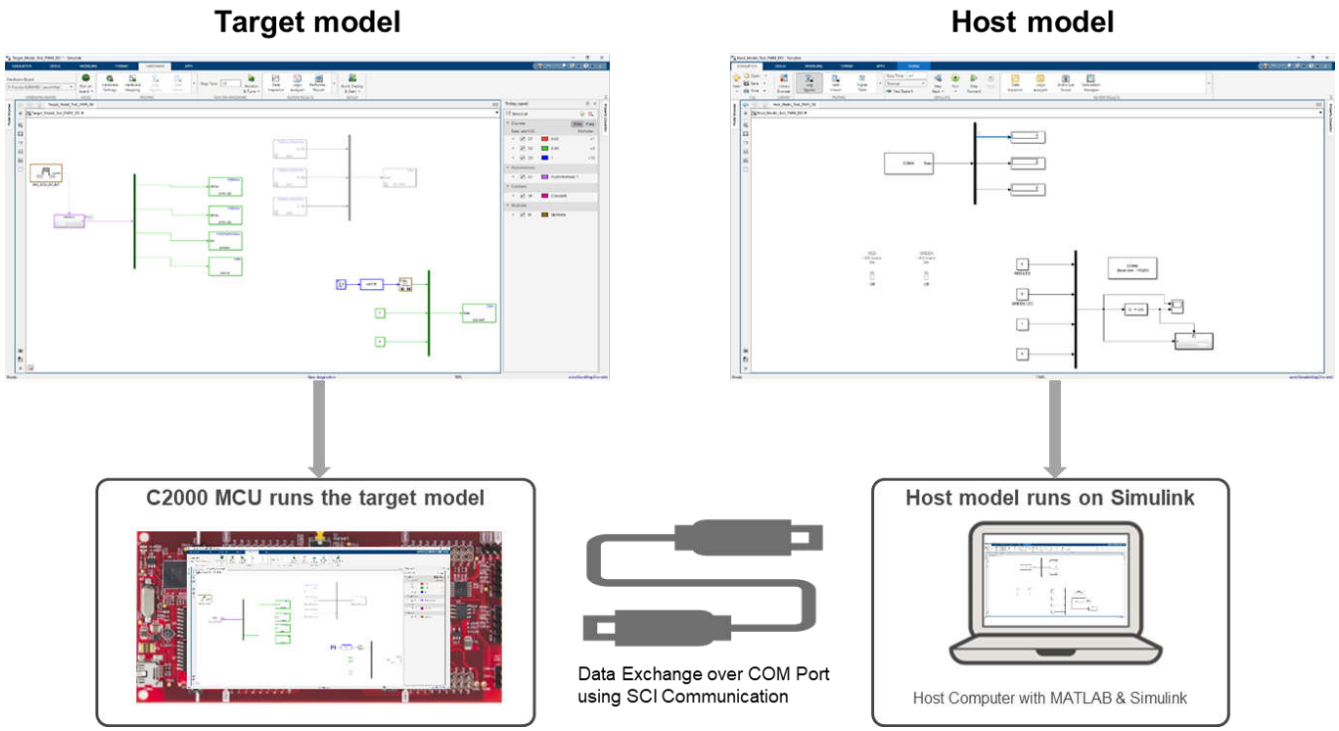

Figure 2-4. System Overview

2.3 Hardware, Software and Testing Requirements

This sections highlights the requirements for running the model-based design on the TIDM-02012 eCompressor hardware. For detailed information on the guidelines, see the *Hardware, Software, Testing Requirements, and Test Results* section in the *High-Voltage HEV and EV HVAC eCompressor Motor Control Reference Design*.

2.3.1 Hardware setup

1. Connect a USB cable to TMDSCNCD280039C controlCARD for JTAG connection.
2. Connect the eCompressor motor wires to terminals J13U, J13V, J13W.
3. Connect measuring instruments – multimeter, oscilloscope probes and other measurement instruments to probe or analyze various signals and parameters as desired. Apply 12V auxiliary DC power supply to terminal J5.
4. Apply a DC bus power supply to terminal J2 and J3. The maximum input to the design is $800V_{DC}$.

Note

Add ferrite beads on JTAG signals and USB cable, if the external emulator has connectivity issues while testing. And make the connection lines as short as possible.

WARNING

The ground planes of both the power domains can be same or different depending on hardware configuration. Meet proper isolation requirements before connecting any test equipment with the board to ensure the safety of yourself and your equipment. Review the GND connections before powering the board. An isolator is required if measurement equipment is connected to the board.

2.3.2 Software setup

Download and install MATLAB, Code Composer Studio™ (CCS) and C2000Ware. Details for the version for these software is given in Section 1.1.

The model-based design for this reference design is available as a part of C2000 MotorControl SDK. Once downloaded and installed, browse to the folder for this design by going to *C2000Ware_MotorControl_SDK_X_XX\solutions\tidm_02012_ecompressor\matlab*. The MATLAB folder just contains the .slx files to start with. Once the project is built, the generated files will also be available in the same folder location.

The example contains two primary files - *TIDM_02012_F280039_MBD.slx* and *TIDM_02012_control_host.slx*. TIDM_02012_F280039_MBD.slx file contains the main control loop implementation whereas the TIDM_02012_control_host.slx contains the universal receiver receiver/transmitter (UART) code, which is used to send data relating to desired speed and control parameters.

2.3.3 Test Procedure

To run the algorithm on hardware, configure the hardware parameters corresponding to inverter and motor by selecting *Edit motor & inverter parameters* option in the TIDM_02012_F280039C_MBD example. This opens up the tidm_2012_param_init_script.m that contains the motor and inverter parameter initializations such as the PWM frequency, data type, motor model number and electrical parameters. It also contains C2000 device initialization parameters such as device part number, frequency, PWM, ADC configuration.

User is expected to update the parameter based on the application.

Before running the algorithm on hardware, the control loop algorithm can be tested using simulation in Simulink. To run the simulation, once all the parameters for the motor and inverter are configured, click on *Simulation' tab in the Simulink window and click on 'Run*. The execution of simulation can be checked by putting scope or data display blocks wherever desired in the example. By opening the Simulink Data Inspector from the *Apps* tab in Simulink, the preset parameters are available to watch. You are free to add more parameters that might be needed for validation of the algorithm.

Keep in mind, that while running the simulation, no code is being generated. It is just the control algorithm that can be validated for operation. The C2000 device F280039C is not needed to be connected until this point.

Once the simulation is validated, to deploy the code on the hardware, select the *Hardware* tab and choose *Build, Deploy and Start* as shown in Figure 2-5. Ensure that the hardware is already connected for the code to be deployed. The code starts running as soon as it is deployed on the hardware.

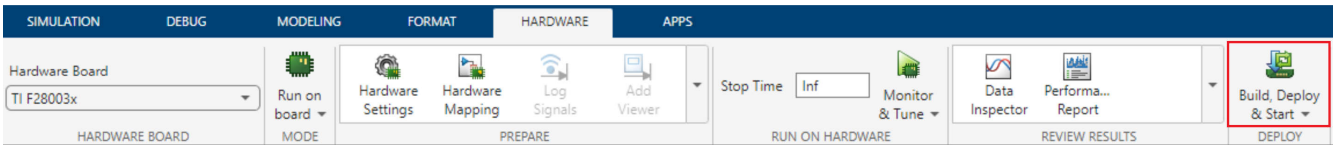

Figure 2-5. Build, Deploy and Start

To control the motor operation such as start-stop, motor speed and watch the run time parameters such as speed, current, etc., a separate example file TIDM_02012_control_host model has to be used. Open the file and choose the correct COM port for UART communication. Ensure the same COM port is enabled in *Host Serial Setup*, *Host Serial Receive* and *Host Serial Transmit* as highlighted in Figure 2-6.

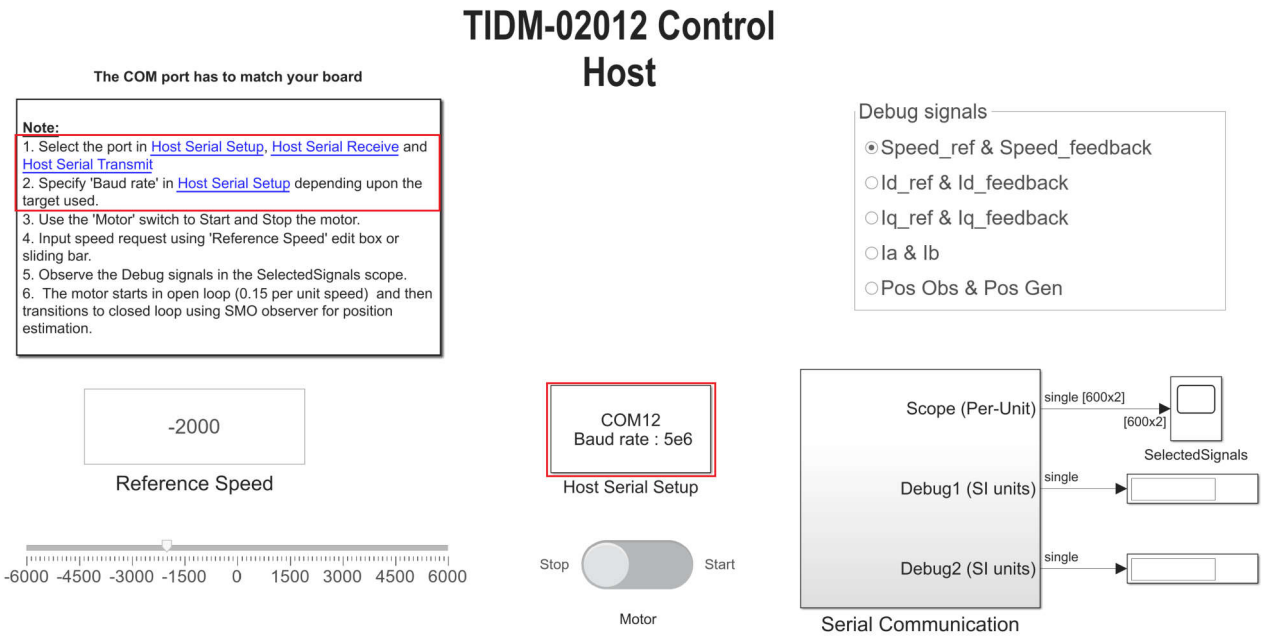

Figure 2-6. Host Serial Configuration

The baud rate for communication with the device is to be configured the same as what is configured in control algorithm model that was deployed on the hardware, the default value on the example is 5e6. To check in the main example, open model explorer (Ctrl + E), go to *Hardware Implementation*, expand the drop down *Target hardware resources* and check the configured baud rate under SCIA as shown in Figure 2-7.

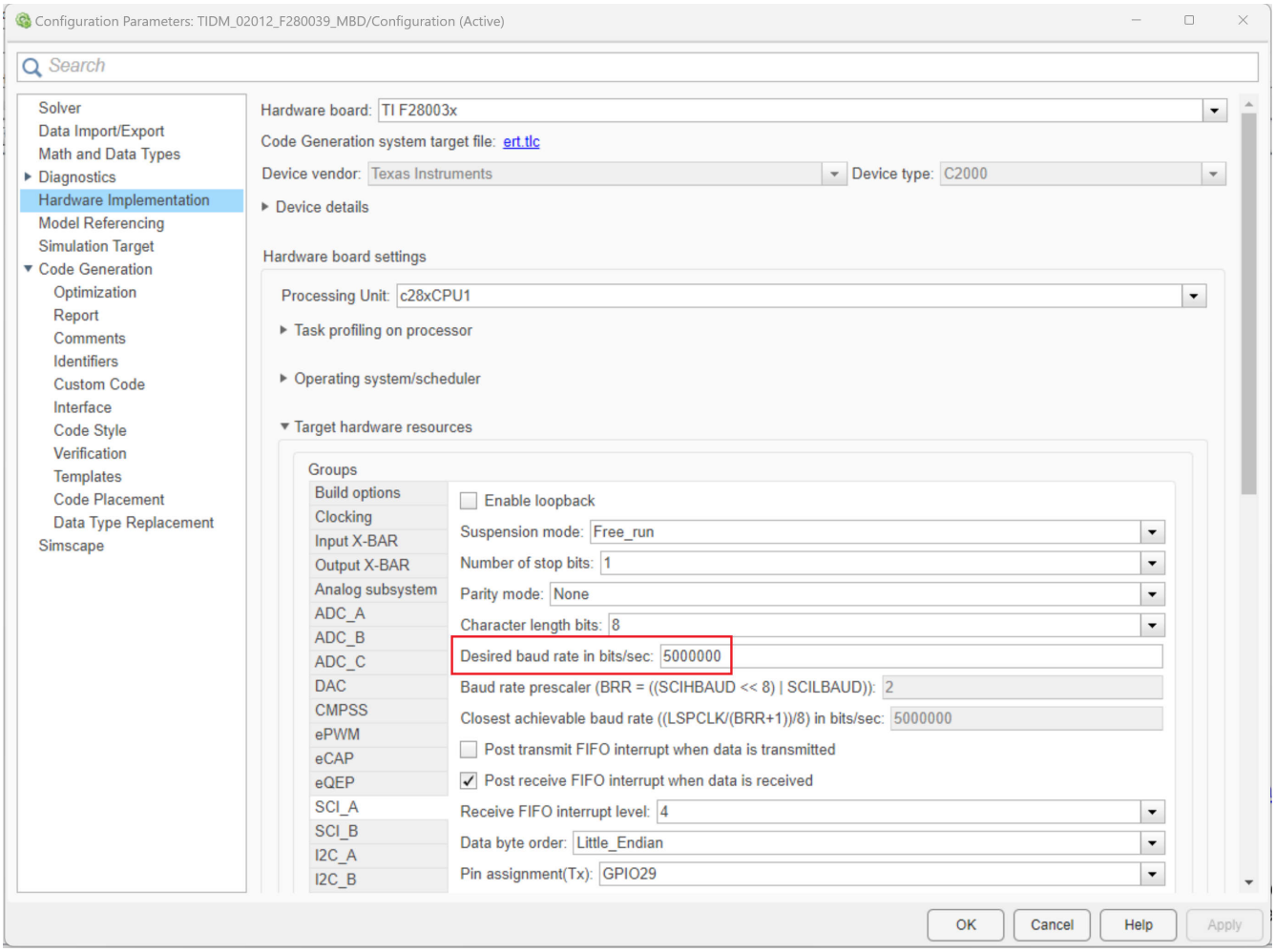

Figure 2-7. Baud Rate Configuration

Run the control host example by clicking on *Run* in the simulation tab. Keep the simulation time as 'Inf' for continuous execution. The selected parameter from the options will now be continuously read and plotted on the scope when the motor is started. Additionally, the motor speed can also be controlled by moving the marker or by providing input in the box.

3 Simulink Configuration Settings

One of the key challenges while using model-based design is to generate optimized code for the C2000 MCU. While generating the code using C2000 Microcontroller Blockset, there are two stages of optimizations in use – Simulink tool optimization and C2000 specific optimizations. To generate the most optimal code for the application, each of the mentioned optimization needs to be configured. It is to be noted that, while we discussed about the TIDM-02012 eCompressor application in the previous section, the optimization settings that are discussed here are generic and can be applied for efficient code generation for all applications.

3.1 Simulink Tool Optimization

The C2000 Microcontroller Blockset along with Embedded Coder is responsible to generate the code for C2000 devices. To ensure that the code generated is optimized for C2000, the configurations in the tool are to be specifically chosen. All configurations corresponding to model-based code generation are available in the model configuration tab in Simulink.

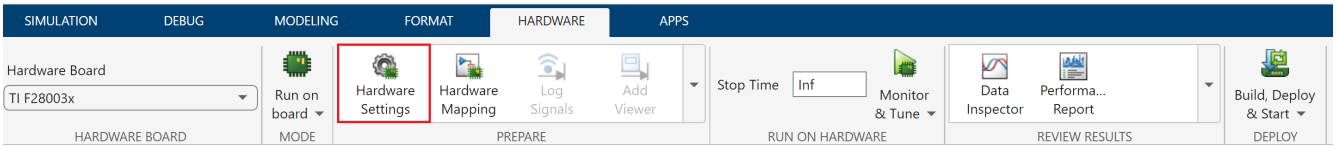

Figure 3-1. Hardware Settings

Open model configuration parameters by selecting the *Hardware settings* option under the *Hardware* tab of Simulink window. Code optimization configurations are available in the *Optimization* tab available in the *Code Generation* drop down as shown in Figure 3-2.

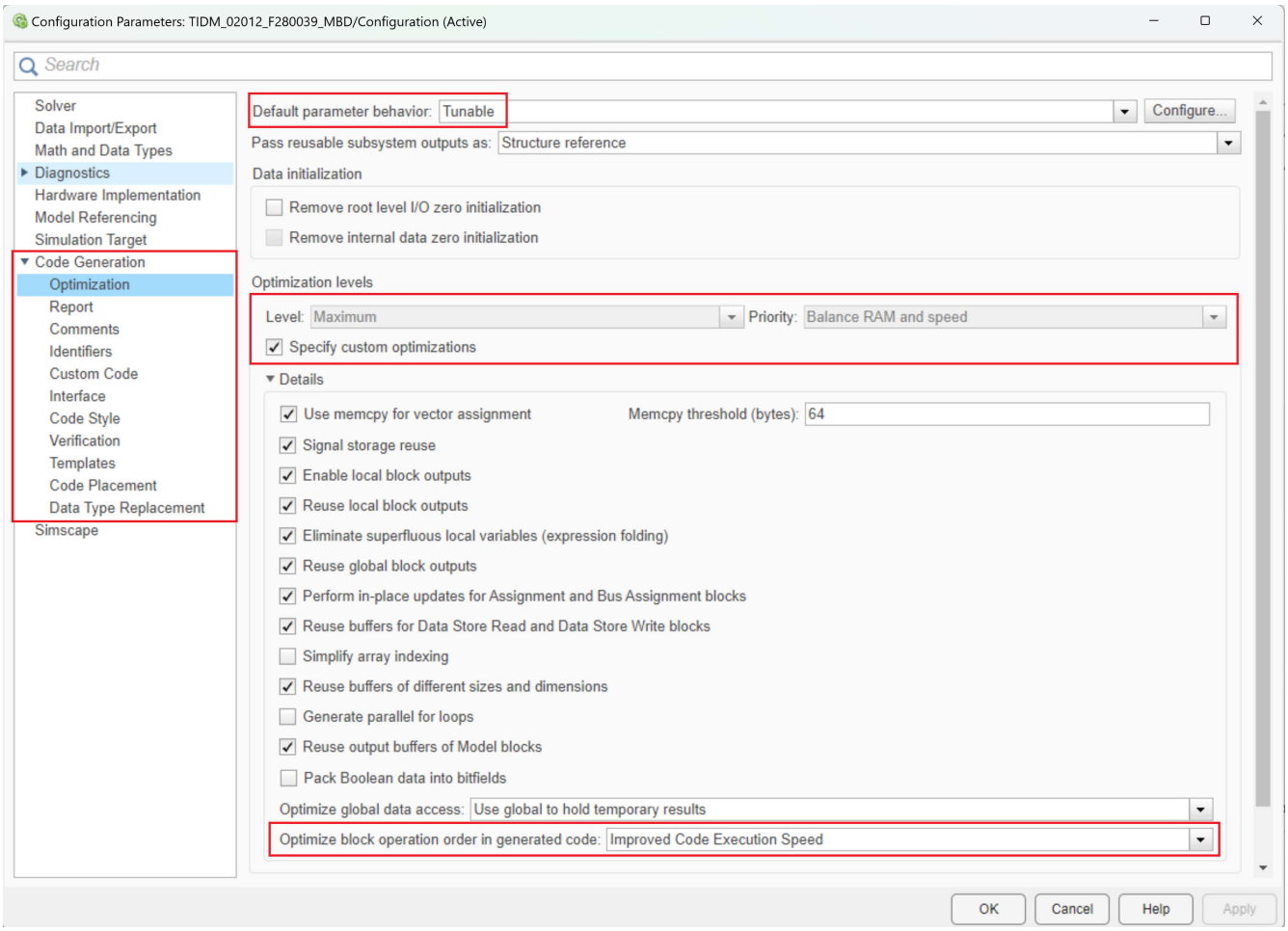

Figure 3-2. Configuration Parameters

Configurations to be made to ensure optimal code generation for C2000 are as shown in Table 3-1.

Table 3-1. MATLAB Optimization Settings

Simulink Tab	Specific Setting	Default Configuration	Optimized Configuration
Code Generation	Build configuration	Faster Builds	Faster Runs
	Prioritized objectives	Unspecified	Execution efficiency
Optimization	Priority	Balance RAM and speed	Speed
	Specify custom optimization	On	Off
	Default parameter behavior	Tunable	Inlined
	Efficient Map of Float to Int	Off	On
Interface	Support absolute time	On	Off
	Support complex	On	Off
	Support non finite	On	Off
Math and Data Types	Life span	auto	1

All the mentioned configurations can be configured manually by selecting appropriate options in the *Model configuration* section as shown in Figure 3-2. Alternatively, a MATLAB .m script containing the above configuration can be integrated with the application example to ensure that the optimized configurations are invoked.

The script to integrate with the application is available with the TIDM_2012_F280039_MBD example with the name - *TIDM_02012_F280039C_MBD_optimconfigs.m.*

Note

The script provided with the example *TIDM_02012_F380039C_MBD_optimconfigs* can be used as-is for other application examples by changing just the model name and running before generating the code to ensure all the optimal settings are configured in the Simulink tool.

Code Generation:

1. Build Configuration: Faster runs

   The build configuration of faster runs ensures the C2000 tool chain configuration is updated to use compiler optimization -O2 in place of default -O0.
2. Objective priorities: Execution efficiency

   Keeping execution efficiency as the highest priority in the objective priority ensures the code generation from MATLAB is focused on the execution speed. Other parameters such as RAM, ROM efficiency are also available as a part of objective priorities that can be invoked based on the requirement, but may impact the performance. Default configuration is none.

Optimization:

1. Optimization priority: Maximize speed

   User is allowed to configure the optimization priority to either maximize the speed, minimize the memory (RAM) or to balance between minimizing RAM and maximizing speed. To generate code focused on better performance in terms of execution speed, the optimization priority is to be set to maximize the speed. The default behavior is to optimize keeping a balance between RAM and speed.

   Change **Configuration Parameters** > **Code Generation** > **Optimization** > **Optimization levels > Priority** to **Maximize Execution Speed**. This applies code generation settings to maximize execution speed.

2. Optimization customization: Off

   If the optimization customization is unchecked, the default configurations under the *Details* tab is auto-populated with the block operation optimized for improving code execution speed.

   Uncheck the **Specify custom optimizations** in **Configuration Parameters** > **Code Generation** > **Optimization** > **Optimization levels.** This disables the parameters in the **Details** section, so that you cannot individually select or clear these parameters.
3. Default parameter behavior: Inlined

   Boost performance by changing **Configuration Parameters** > **Code Generation** > **Optimization** > **Default parameter behavior** to **Inlined**, optimizing default parameter behavior.

   Also, check the **Inline invariant signals** checkbox in **Configuration Parameters** > **Code Generation** > **Optimization** > **Advanced parameters.** This uses the numerical values of model parameters, instead of their symbolic names, in generated code.
4. Efficient Map of Float to Int: On

   Datatype mapping from float to int is to be enabled. This configuration removes code that handles floating-point to integer conversion results for NaN values.

   Check the **Remove code from floating-point to integer conversions with saturation that maps NaN to zero** checkbox in **Configuration Parameters** > **Code Generation** > **Optimization** > **Advanced parameters.** This removes code that handles floating-point to integer conversion results for NaN values.

Interface:

Under the interface tab, the support for all the data type configurations that are noto going to be in use can be disabled. For generating efficient code, the configuration support for absolute time, non-finite numbers and complex numbers can be removed. Additional checks for the selected configurations in the *Support* tab is enabled, which may add additional number of cycles in run time application code.

Uncheck the **Support: absolute time, non-finite numbers and complex numbers** checkbox in **Configuration Parameters** > **Interface** > **Software environment.** This causes the support for blocks that depend on absolute time, non-finite numbers and complex numbers. Deselecting unused configurations under the *Support* tab ensures better run-time code efficiency.

Application lifetime:

Set the **Application lifespan (days)** field in **Configuration Parameters** > **Math and Data Types** > **Advanced parameters.** This specifies how long, in days, an application that depends on elapsed time can execute before timer overflow occurs.

3.1.1 Optimum Code Generation

The configurations showcased in the table can be manually configured by configuring each parameter in the *Hardware Settings* in Simulink. To avoid manual effort, MATLAB also allows to configure settings in Simulink through MATLAB script.

The optimized code generation configuration can also be simply parsed through a script by either including the script in the existing model or just by running the script once to configure all settings before running the application code. The parameter 'mdl' in the script needs to reflect the name of the model in use to correctly configure the settings. It can be validated, if the settings are configured, by manually checking the configurations in the *Hardware Settings* in Simulink window.

```
%% Load the model
mdl = 'TIDM_02012_F280039_MBD';%Model Name
load_system(mdl);

%% Set Build Configurations and Prioritized Objectives in Code Generation tab
set_param(mdl,'BuildConfiguration','Faster Runs') ; %Build Configurations
set_param(mdl,'ObjectivePriorities','Execution efficiency');  %Prioritized Objectives
%% Set Level, Priority in Optimization levels and enable some advanced Parameters in Optimization
tab
set_param(mdl,'OptimizationPriority','speed');%Optimization Priority
```

```
set_param(mdl,"OptimizationCustomize","off");%Customize Optimizations Checkbox
%% TODO: Check -- updated Code Config > Optimization > default parameter behaviour to inlined
instead of tunable to make the below config work
set_param(mdl,"InlineInvariantSignals","on");%Inline Invariant Signals
set_param(mdl,"EfficientMapNaN2IntZero","on");% Removes code from float to int with saturation
mapping NaN to zero.
%%  Remove support for non-finite, complex and absolute time Interface tab
set_param(mdl,"SupportAbsoluteTime","off");%Remove Absolute time support
set_param(mdl,"SupportComplex","off");%Remove Complex Number support
set_param(mdl,"SupportNonFinite","off");%Remove Non-Finite Number support
%% Application Lifetime setting
set_param(mdl,"LifeSpan","1");%Remove Absolute time support
```

Note

In the code block above, replace the model name with the application model name in use and run the script before executing the code generation for the application model to incorporate all the optimized configuration.

3.2 C2000 Specific Optimization

To ensure that the generated code is optimally executed on the C2000 MCU, compiler settings needs to be correctly configured beyond the Simulink specific optimization, which are hardware specific to the C2000 MCUs. As discussed in Section 3.1, if the build configuration setting is correctly configured as *Faster Runs*, the optimization level (-O2) will be invoked while running the code on the hardware. Additionally, if manually the optimization level is to be changed, the *Toolchain details* section in the Code generation tab of hardware settings allows you to configure the compiler settings by selecting build configuration as *Specify*. Compiler optimization settings are available in the C compiler settings as shown in Figure 3-3.

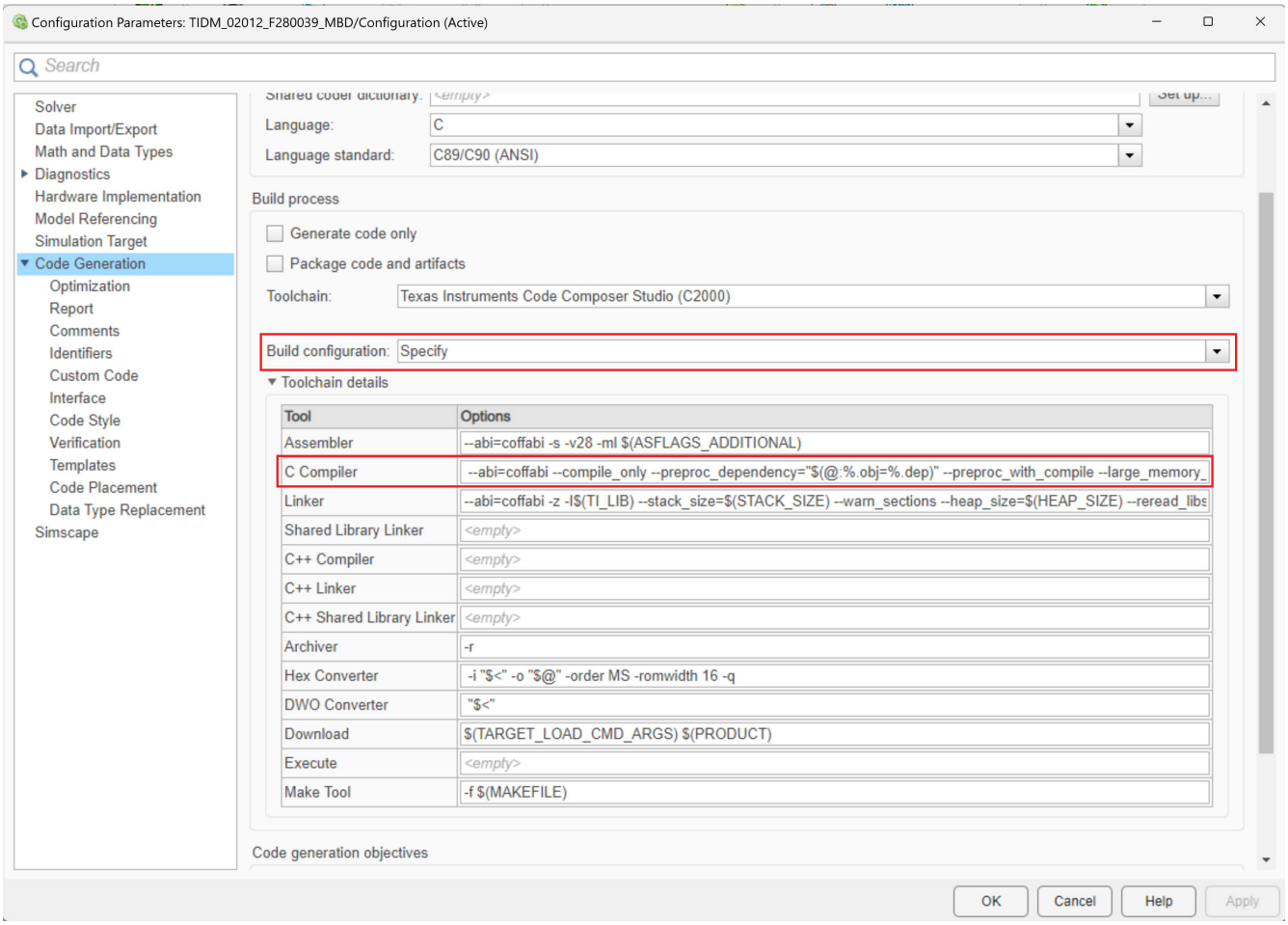

Figure 3-3. Custom Compiler Optimization Configuration

3.2.1 Using TMU Through Simulink

C2000 devices also contain accelerators such as Trigonometric Math Unit (TMU) to perform floating point trigonometric operations faster. To ensure that the generated code is using the TMU instructions, the hardware implementation tab in the hardware settings contains the option to enable TMU. This ensures that wherever trigonometric operations are to be performed, the TMU instructions are invoked. By default, TMU is enabled for the devices having TMU accelerator unless manually unchecked.

While TMU can specifically be invoked by calling the intrinsic function as described in the C2000 compiler user guide, by enabling the TMU function in the Simulink window still uses the conventional trigonometric operations and not the TMU intrinsics. The TMU is enabled by using the appropriate compiler flag in the build configuration and ensures that the hardware accelerator is correctly invoked for performing trigonometric operations. The look up table blocks such as sin-cosine LUT is still using the LUT-based approach even if TMU is enabled. The sin-cosine blocks from the Simulink Library needs to be used to generate the trigonometric operations in place of the look up table-based approach.

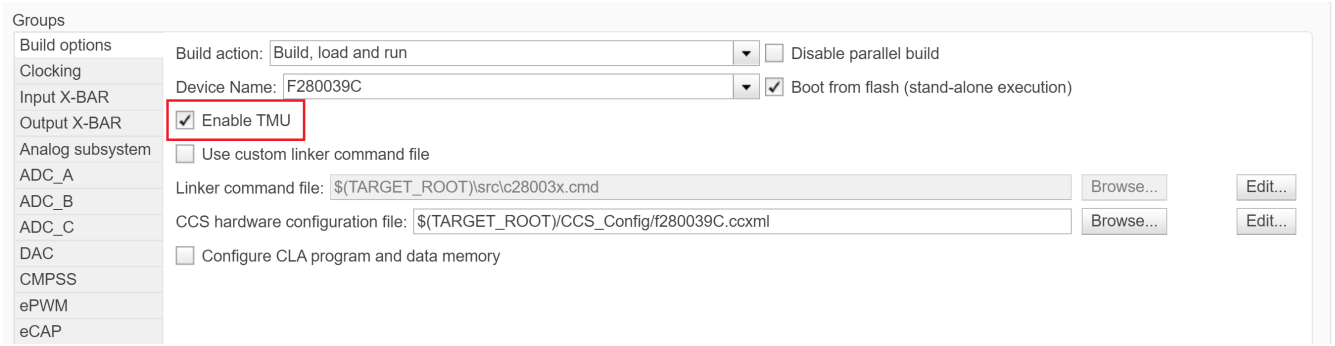

Figure 3-4. TMU Configuration

3.2.2 Using Software Libraries Through Simulink

To invoke additional software libraries for faster computation, embedded coder allows to import code replacement libraries (CRL), which replaces the conventional code generation with the library functions wherever suitable. One such library for faster computations on fixed point arithmetic is C28x IQMath library. If the application uses Q numbers, it is recommended to use the IQMath CRL. To invoke IQMath CRL, go to *Interface* section under the *Code Generation* tab in the Hardware settings. Select *TI C28x* library as shown in the image, which contains the code replacement libraries for IQ Math, FastIntDiv, CLA, and so forth.

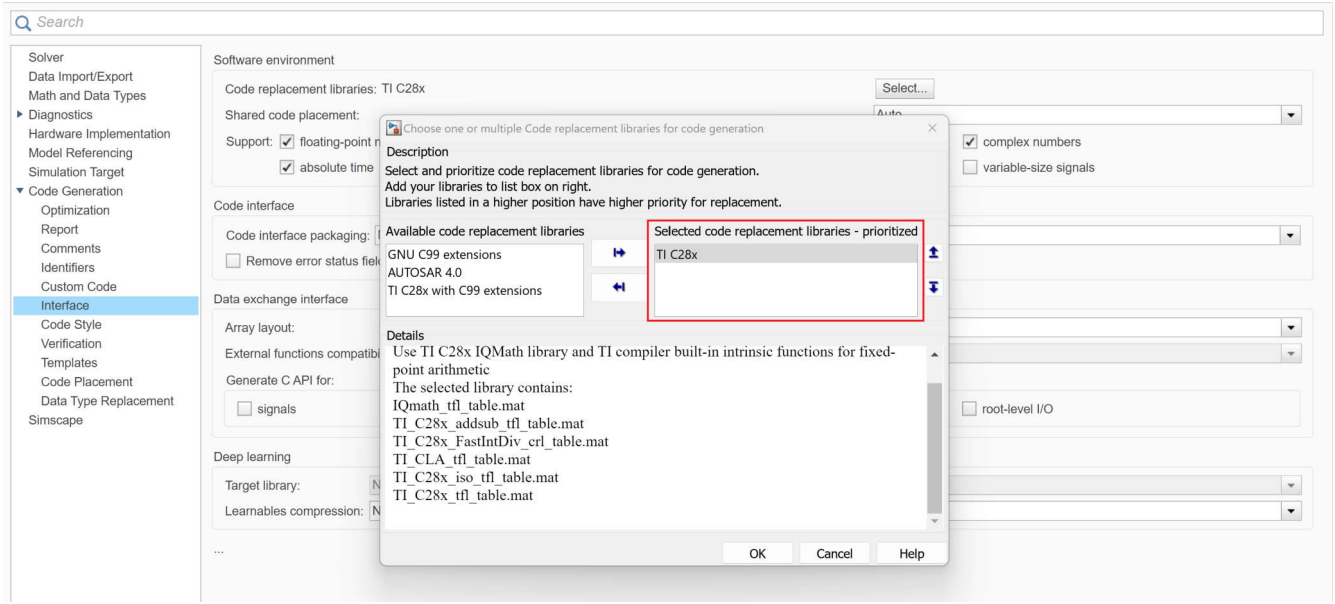

Figure 3-5. Code Replacement Library configuration

Note

MATLAB versions older than 2018 do not support the TI C28x library.

Note

It is to be noted that hardware accelerator Trigonometric Math Unit (TMU) is accelerator for performing floating point operations faster, whereas, the IQ Math library is a software library that accelerates the fixed point computations faster.

3.2.3 Running Code From RAM

Any application code can be either placed in RAM or Flash for storage and run time execution. Placing the code in flash memory allows to store a larger application size since flash is typically bigger than RAM in terms of memory, but executing the code from flash invokes additional wait states while performing read-write operations. Since executing the code from RAM becomes the faster alternative, a solution involving storage of code in flash memory and executing the code from RAM becomes the optimal solution.

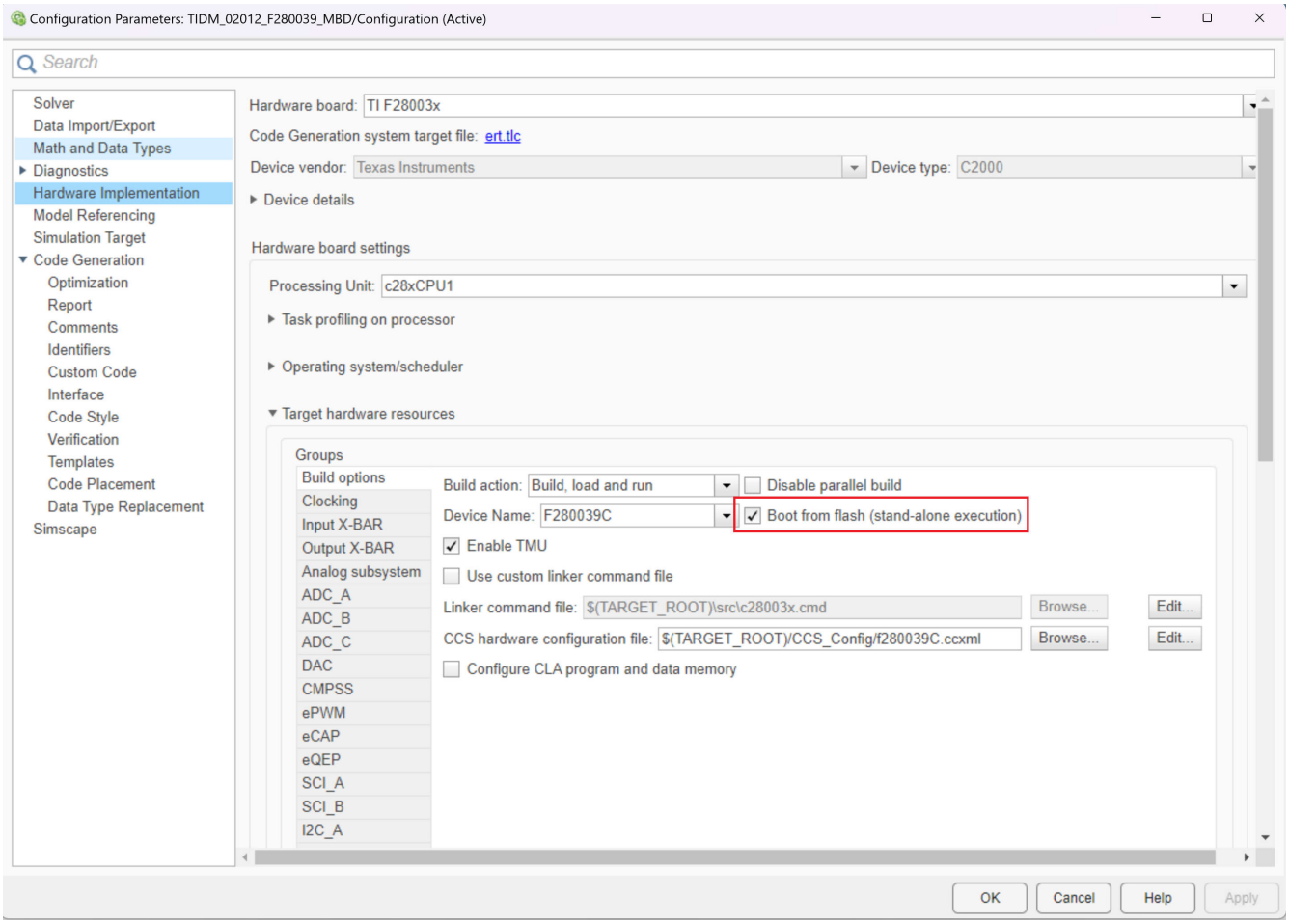

Figure 3-6. Booting From Flash

If the option to boot from flash is selected as shown in Figure 3-6, the code is stored in flash. To copy and run from RAM, by right clicking on the subsystem block and selecting *Block Parameters (Subsystem)*. Select the *Code Generation* tab and configure the *Memory section for execution functions:* as *code_ramfuncs* from the drop-down menu as shown in Figure 3-7. It is to be noted that if the subsystem is larger than the available memory on the device, compile time error is shown showcasing not enough memory.

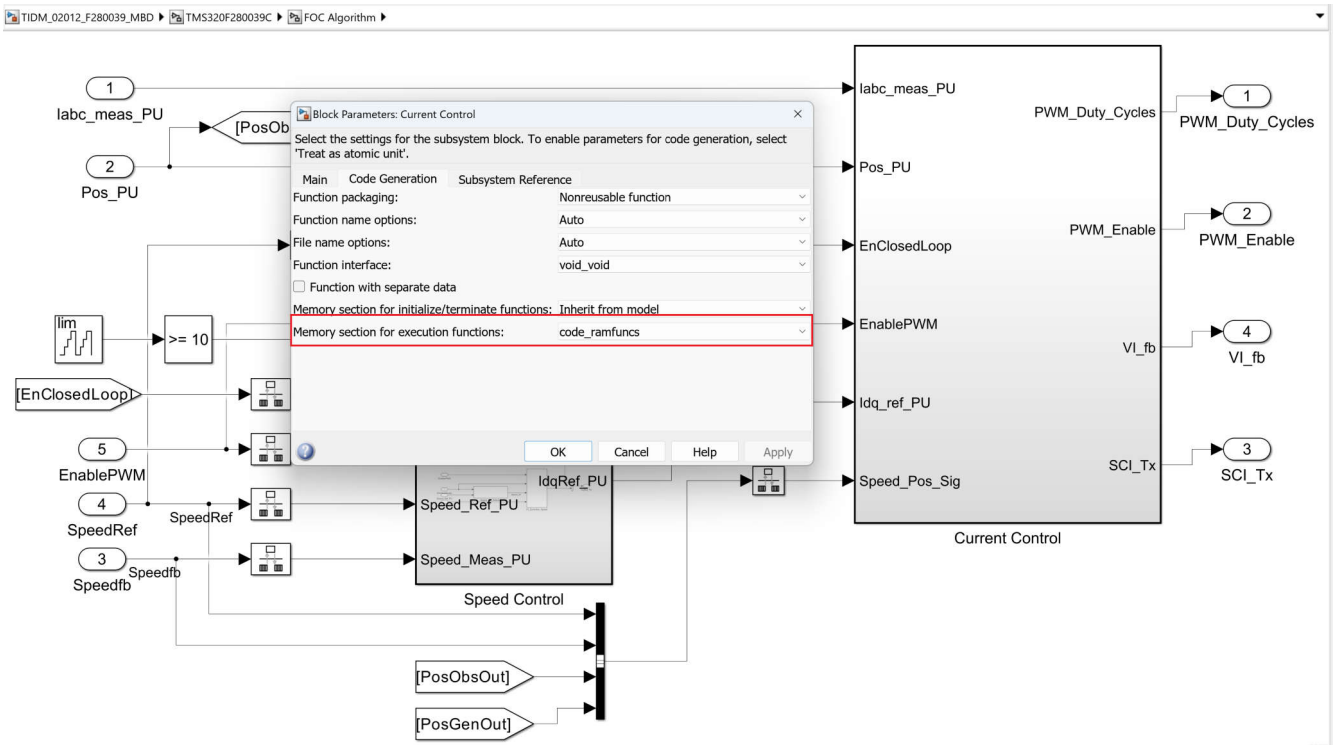

Figure 3-7. Running Code From RAM

3.3 Performance Comparison

After enabling the optimizations as discussed in Section 3, performance of TIDM-02012 eCompressor model is validated against the default configuration. Overall improvement for the fastest execution loop of FOC is around 57% whereas for the speed loop the improvement in execution performance is around 38%.

The numbers discussed in Performance Comparison are profiled using the processor-in-loop (PIL) technique, which is discussed in Section 4.1. The default configuration still uses the C2000 hardware accelerator, that is, TMU. The difference in the optimized configuration column is the enabling of configuration settings optimized in MATLAB through the script shown in Section 3.1.1.

Table 3-2. Performance Comparison

Function (*rate of execution*)	Default Configuration Execution Time (*ns*)	Optimized Configuration Execution Time (*ns*	Performance Improvement Over Default Configuration	Default Configuration Average CPU Utilization	Optimized Configuration Average CPU Utilization
Current control loop (66.67µs)	15261	6286	57.45%	22.89%	9.43%
Speed control loop (0.67ms)	2961	1840	37.87%	0.44%	0.28%
Background housekeeping loop (0.1s)	260	200	23.07%	0.0003%	0.0002%

4 Profiling Using Simulink

The Embedded Coder tool allows the flexibility to profile a part of the application code, multiple blocks in the application code or the complete application code using Simulink. There can be multiple ways to profile the code generated by MATLAB – Processor-in-loop method on Simulink, C2000 Timer, general-purpose input/output (GPIO)-based profiling and the CCS clock profiler tool. While there is no notable difference in terms of the profiling data, there might be small differences due to different approach of profiling the application code.

4.1 Processor-in-loop (PIL) Method

Simulink offers a processor-in-loop (PIL) simulation tool that allows to verify the code and profile it using the SIL/PIL manager tool. To profile using PIL tool, the hardware settings needs to be configured. Open hardware settings (Ctrl + E), go to *Verification* tab under *Code Generation* and enable *Measure task execution time* with settings as shown in Figure 4-1 and select *PIL* in the *Create Block* section under *Advanced Parameters*.

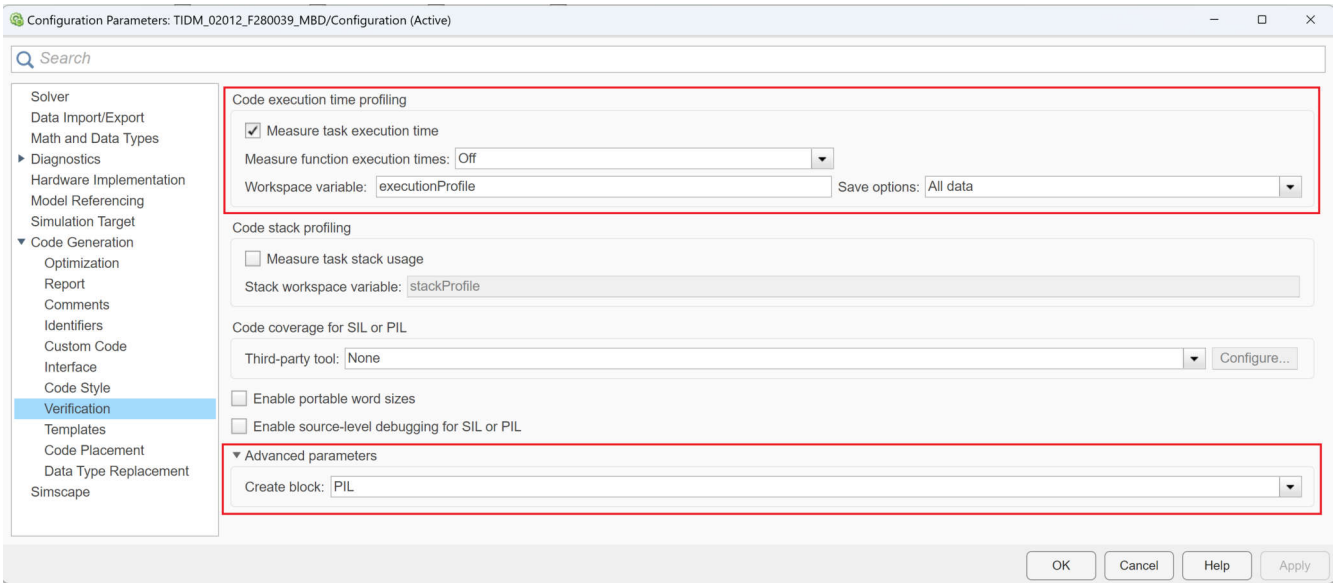

Figure 4-1. Processor-in-loop Profiling

Configure the COM port in the PIL section in *Target hardware resources* under the *Hardware Implementation* tab as shown in Figure 4-2. Validate the serial port connection in MATLAB as per the device UART port.

To create a PIL block from a subsystem / system, right click the block to be chosen for profiling and select *Deploy subsystem to hardware* in *C/C++ Code*. This generates the code for the chosen block and generate a PIL block for the same. Replace the generated PIL block with the actual block. Open the SIL/PIL manager from the *APPS* tab. Select *SIL/PIL Simulation Only* as the mode, *Model blocks in SIL/PIL mode* as the System Under Test and give a reference stop time for the simulation before running the automated verification as shown in the Figure 4-2. Once the execution is successfully completed, the code execution parameters are available in a report available under the *Results* section in MATLAB. The report contains CPU utilization, execution time in nano-seconds (avg, min and max), which can be used to compute the cycle counts based on the device in use.

While the PIL is only used for profiling a single block of code, the C2000 timer and GPIO based profiling methods can be used to profile multiple blocks of code at a time. The advantage of PIL-based profiling is that the profiling data is clearly stated with respect to the rate of execution. If multiple loops are operating at different rates, PIL separates out the execution time for each of the blocks based on its rate of execution. The PIL-based method comes with a fixed small overhead in the measurement, hence it is recommended to use the PIL-based profiling method for profiling a block, which uses larger code size to minimize the effect of overhead.

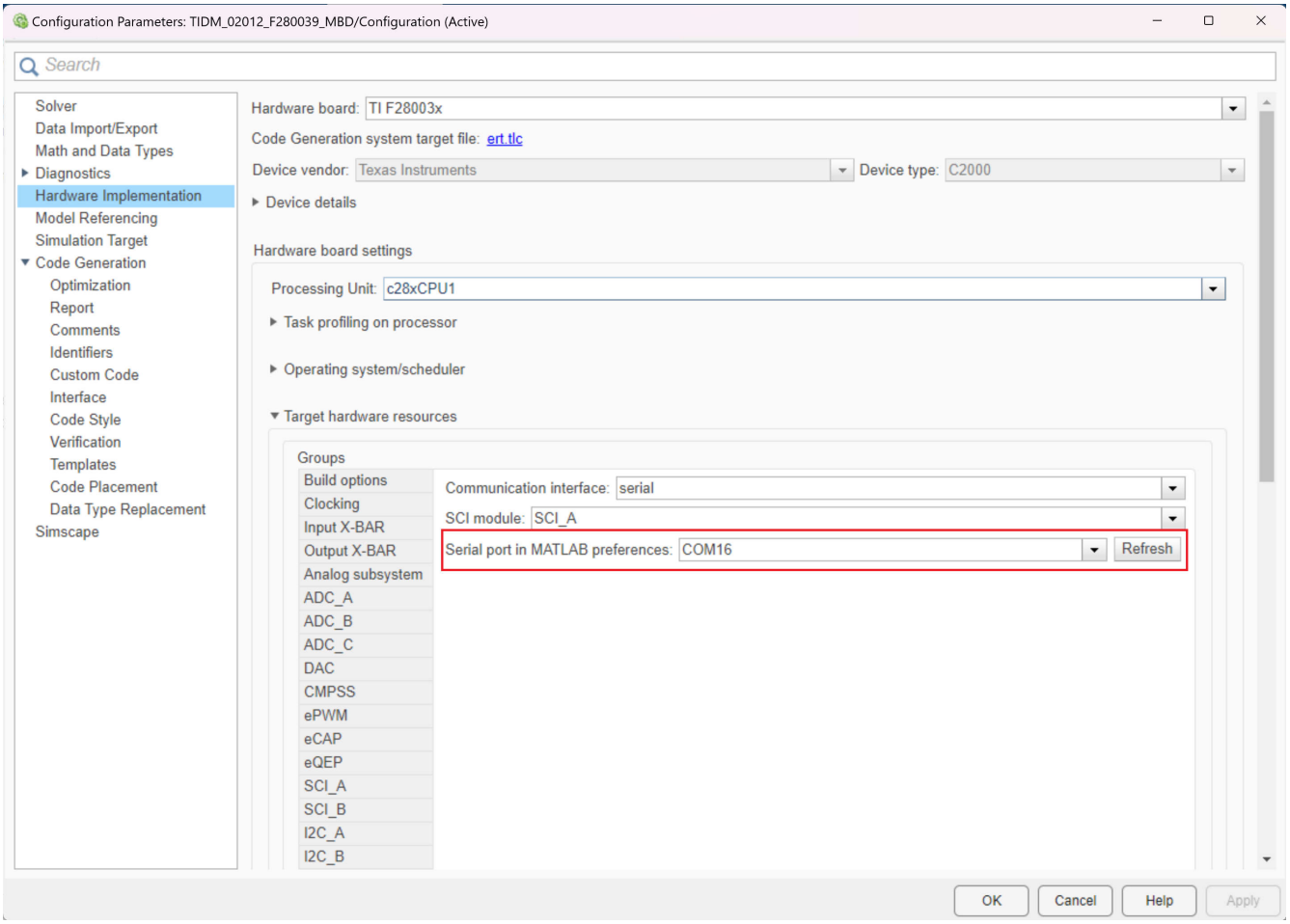

Figure 4-2. PIL COM Port Configuration

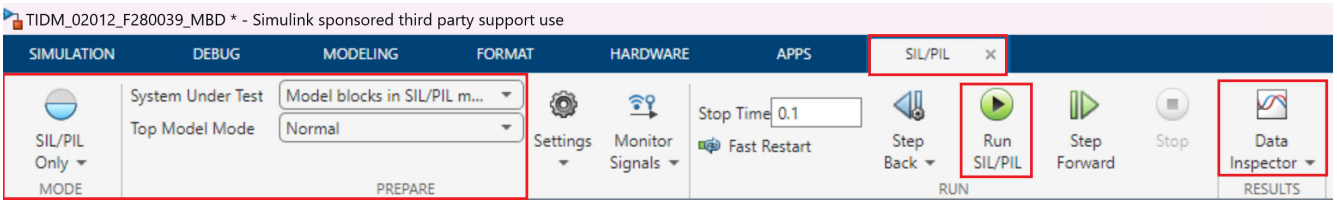

Figure 4-3. PIL Configuration settings

For more information on PIL Simulation, see the PIL Simulation article.

4.2 C2000 Timer-Based Profiling

Other than the processor-in-loop simulation profiling available with the Simulink, the C2000 timer peripheral can also be used to get the execution time data. The timer code needs to be integrated with the existing model for profiling making this method an intrusive profiling method. The overhead incurred by the timer-based method can be removed by separately measuring the timer execution time. While running the code for the application, the timer code can be removed to save on additional cycles for timer configuration and running. Steps to be taken to profile a section or complete application code using C2000 timer peripheral are discussed below. The current example uses Timer 2 as the profiling tool, but if the timer is already in use by the application, then it is recommended to choose a timer that is not in use.

At the top level of the model, add the *System Initialize* block available in the Simulink library under Simulink Coder > Custom Code. This block contains the timer initialization code.

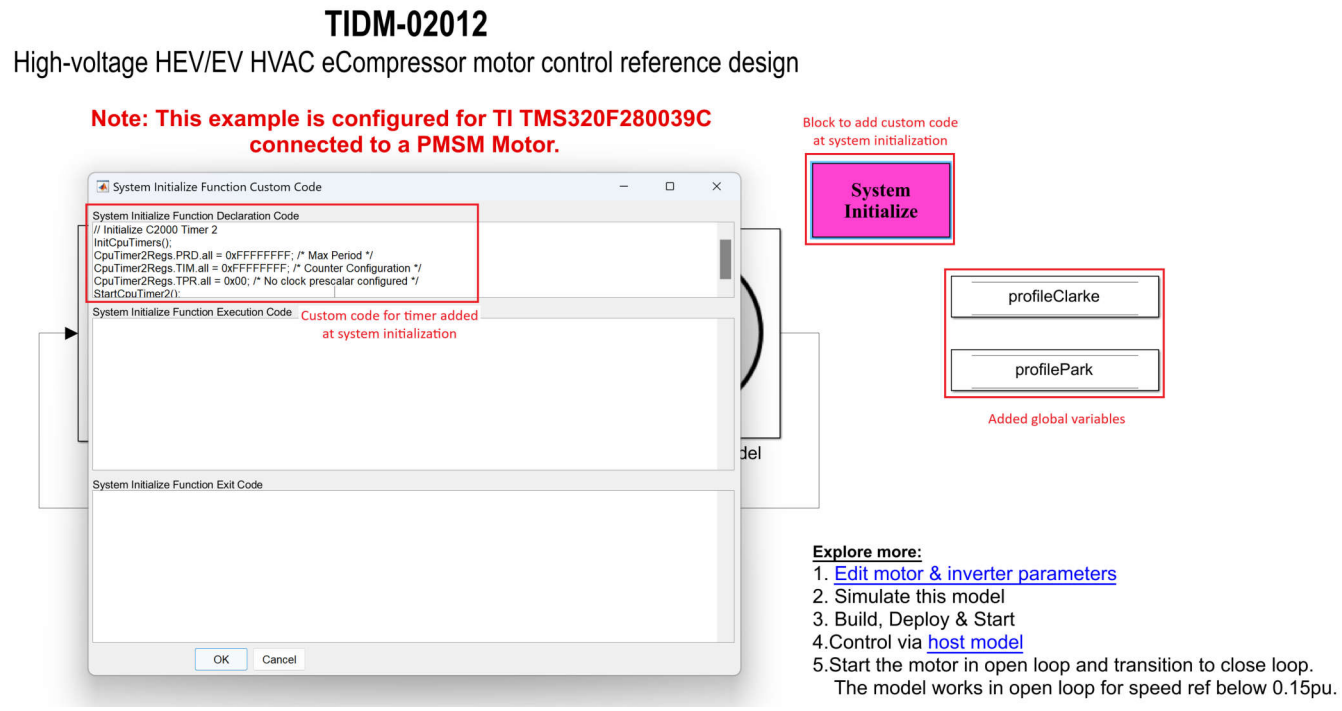

Figure 4-4. Timer Initialization Code

```
// Initialize C2000 Timer 2
InitCpuTimers();
CpuTimer2Regs.PRD.all = 0xFFFFFFFF; /* Max Period */
CpuTimer2Regs.TIM.all = 0xFFFFFFFF; /* Counter Configuration */
CpuTimer2Regs.TPR.all = 0x00; /* No clock prescalar configured */
StartCpuTimer2();
```

Add *Data Store Memory* blocks to the top level along with System initialize block for the blocks that need to be profiled. Name the *Data Store Memory* blocks appropriately as shown in Figure 4-5. Open the Embedded Coder application from the *APPS* menu. Select Code Mappings > Component Interface, navigate to *Data Stores* tab. Change the storage class of variables created using *Data Store Memory* blocks to *ExportedGlobal*.

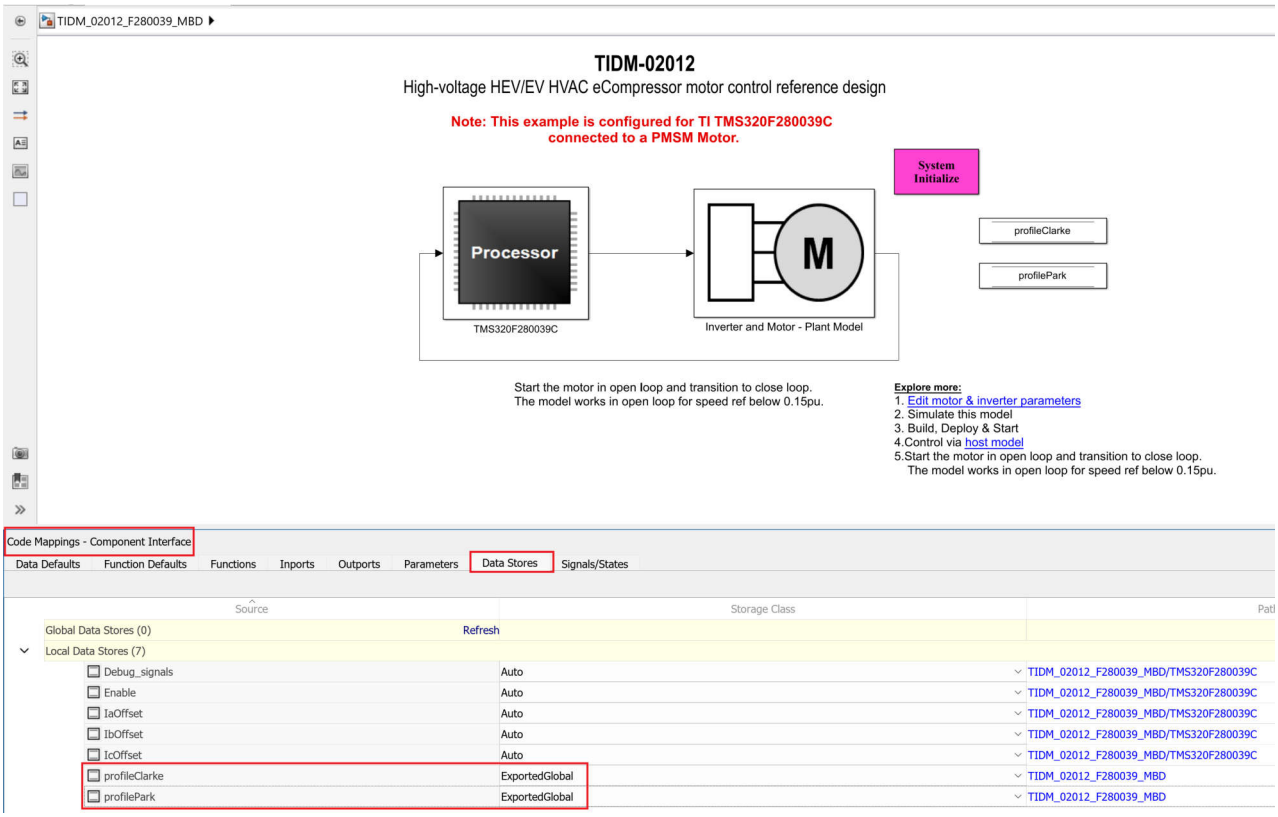

Figure 4-5. Storage Class for Variable

Now, within the subsystem to be profiled, add a *System Outputs* block from the Simulink library. If it is a standalone block, for example, park transform block, then the block can be made a subsystem after which the System Outputs block can be added. In the *System Outputs* block, add the read timer value code and store in a temporary variable in the Function Declaration Code section and read the timer value again in the Function Exit Code section. Compute the difference in the exit code section and store the value in global variable defined earlier as shown in the code block.

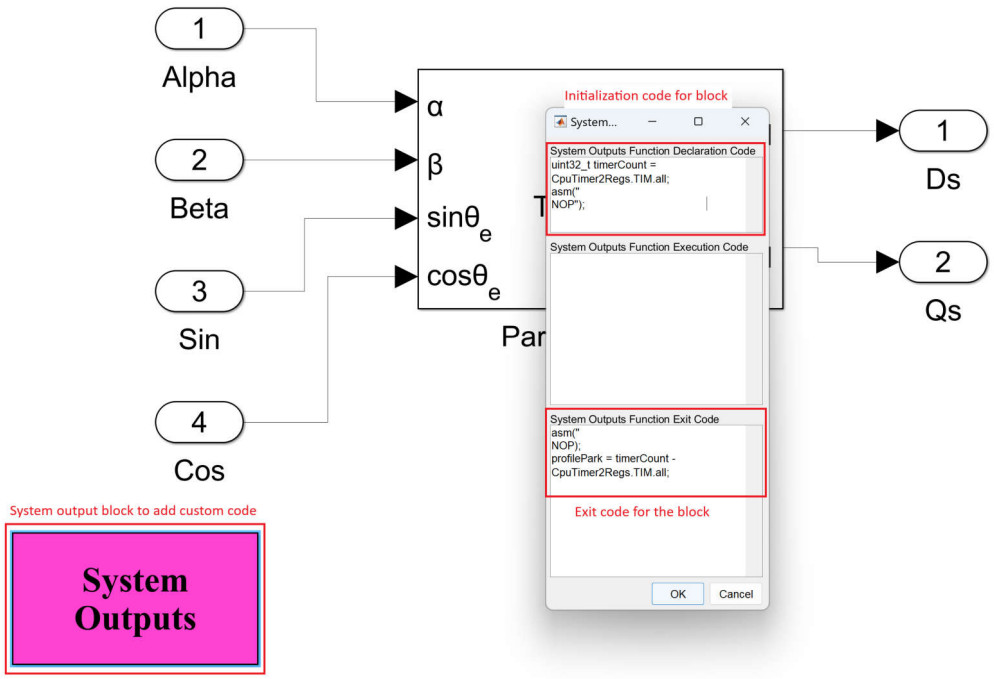

Figure 4-6. System Outputs Timer Code

To profile a block, once the subsystem is created, right click on it and navigate to Block Parameters(subsystem). Check the treat as atomic unit checkbox. Build the model and load the generated .out file in the device using CCS. Add the added global variables in the expressions window of CCS to watch and run the code. Open the host control model, select the appropriate COM port and set the desired speed. Run the motor, using the toggle switch to start the motor. This sends an enable PWM signal to the device to run the algorithm, after which the variables in the watch window are updated with the cycle count. Observe the variation in the variables using the *Continuous refresh* option in the expressions window.

```
/* Code block for System Output Function Declaration Code */
uint32_T timerCount1 = CpuTimer2Regs.TIM.all;
asm(" NOP");

/* Code block for System Output Function Exit Code */
asm(" NOP");
profilePark=  timerCount1 - CpuTimer2Regs.TIM.all;
```

To remove the timer measurement overhead, a separate block of code can be added in any of the blocks with the code mentioned in the block below. The idea to calculate the timer overhead is to do a back to back timer read and difference between the reads will be the overhead. On execution of the code block, the overhead can be quantified and subtracted out from the profiling data that is calculated for the other blocks. It is to be noted that, similar to the global variables defined previously for each of the blocks, the variable corresponding to timer overhead also has to be added.

```
/* Code block for calculation of timer overhead */
uint32_T overheadCount = CpuTimer2Regs.TIM.all;
asm(" NOP");

/* Code block for System Output Function Exit Code */
asm(" NOP");
profileOverhead=  overheadCount - CpuTimer2Regs.TIM.all;
```

Other than C2000 timer-based profiling, GPIO-based profiling method can also be used if an oscilloscope is available. It is to be noted that GPIO-based profiling needs external scope to view the waveform and measure the timings. Instead of the timer reads as discussed in this section, the GPIO can be toggled at the start and end of the routine, which needs to be profiled. The overhead in the GPIO-based profiling method is limited to the time taken to write to the GPIO registers.

4.3 Code Composer Studio tools

The Code Composer Studio (CCS) IDE also allows to profile the execution cycles for application code. The clock tool available with CCS allows to get the point-to-point cycle count information. To use the CCS for profiling, import the MATLAB generated project in the CCS environment by opening CCS and selecting *Import CCS project* under the *Project* tab. Browse to the MATLAB project folder location and import the project. Once imported, the project should be available in the *Project Explorer* in CCS window. Connect the hardware and debug the project to load the .out file on the C2000 device.

Once the project is loaded, the clock tool can be enabled for use. Detailed description on how to use the Code composer studio for profiling is given in the link - Profiling on C28x Targets.

5 Summary

The industrial and automotive control becoming more complex makes a clear way for the usage of minimum code tools such as Embedded Coder by MATLAB for C2000 real time controllers. It is possible to achieve the ease-of-use while achieving the performance entitlement for such complex time-critical control applications.

Implementation of optimized configuration over the default configurations as showcased in Table 3-1 is enough to achieve the faster computation performance with optimized code generation. The simpler way to implement the optimized configuration is by incorporating the code script in the MATLAB model.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265