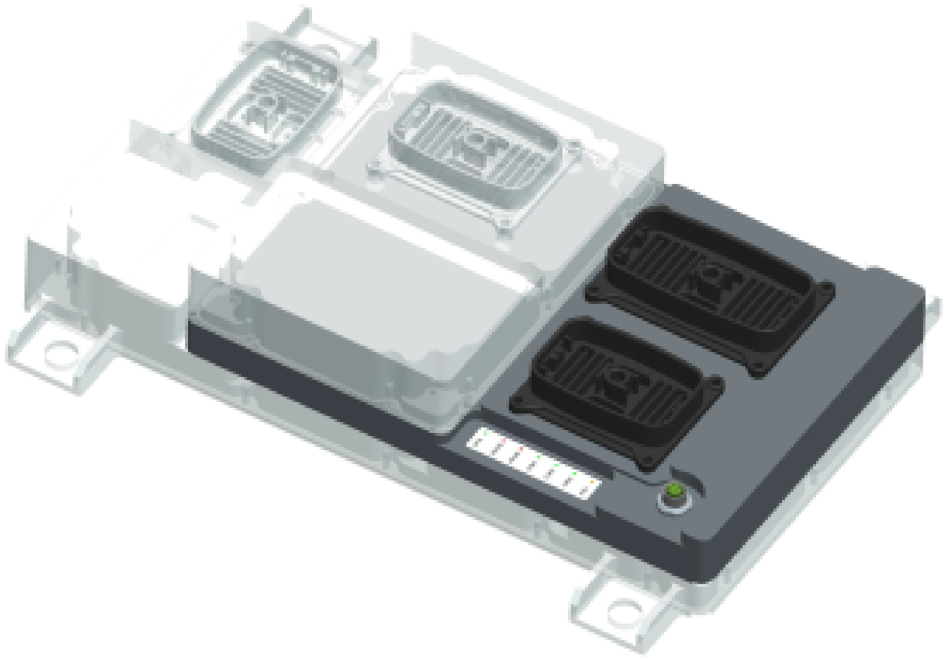
WOODWARD

Product Manual 35203V1
(Revision -, 6/2023)
Original Instructions



Large Engine System Platform

Packages

Developer Manual

Contents

Illustrations and Tables

Warnings and Notices

Important Definitions

This is the safety alert symbol used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

- **DANGER** - Indicates a hazardous situation, which if not avoided, will result in death or serious injury.
- **WARNING** - Indicates a hazardous situation, which if not avoided, could result in death or serious injury.
- **CAUTION** - Indicates a hazardous situation, which if not avoided, could result in minor or moderate injury.
- **NOTICE** - Indicates a hazard that could result in property damage only (including damage to the control).
- **IMPORTANT** - Designates an operating tip or maintenance suggestion.

WARNING

Lockout/Tagout LOTO

Ensure that personnel are fully trained on LOTO procedures prior to attempting to replace or service equipment on a "live" running engine. All safety protective systems (overspeed, over temperature, overpressure, etc.) must be in proper operational condition prior to the start or operation of a running engine. Personnel should be equipped with appropriate personal protective equipment to minimize the potential for injury due to release of hot hydraulic fluids, exposure to hot surfaces and/or moving parts, or any moving parts that may be activated and are located in the area of control of the unit.

WARNING

Overspeed / Overtemperature / Overpressure

The engine, turbine, or other type of prime mover should be equipped with an overspeed shutdown device to protect against runaway or damage to the prime mover with possible personal injury, loss of life, or property damage.
The overspeed shutdown device must be totally independent of the prime mover control system. An overtemperature or overpressure shutdown device may also be needed for safety, as appropriate.

WARNING

Personal Protective Equipment

The products described in this publication may present risks that could lead to personal injury, loss of life, or property damage. Always wear the appropriate personal protective equipment (PPE) for the job at hand. Equipment that should be considered includes but is not limited to:

- **Eye Protection**
- **Hearing Protection**
- **Hard Hat**
- **Gloves**
- **Safety Boots**
- **Respirator**

Always read the proper Material Safety Data Sheet (MSDS) for any working fluid(s) and comply with recommended safety equipment.

⚠WARNING **Start-up**	**Be prepared to make an emergency shutdown when starting the engine, turbine, or other type of prime mover, to protect against runaway or overspeed with possible personal injury, loss of life, or property damage.**

⚠WARNING **Automotive Applications**	**On- and off-highway Mobile Applications: Unless Woodward's control functions as the supervisory control, customer should install a system totally independent of the prime mover control system that monitors for supervisory control of engine (and takes appropriate action if supervisory control is lost) to protect against loss of engine control with possible personal injury, loss of life, or property damage.**

⚠WARNING

IOLOCK

IOLOCK: driving I/O into a known state condition. When a control fails to have all the conditions for normal operation, watchdog logic drives it into an IOLOCK condition where all output circuits and signals will default to their de-energized state as described below. ***The system MUST be applied such that IOLOCK and power OFF states will result in a SAFE condition of the controlled device.***

- **Microprocessor failures will send the module into an IOLOCK state.**
- **Discrete outputs / relay drivers will be non-active and de-energized.**
- **Analog and actuator outputs will be non-active and de-energized with zero voltage or zero current.**

Network connections like CAN stay active during IOLOCK. This is up to the application to drive actuators controlled over network into a safe state.

The IOLOCK state is asserted under various conditions, including:

- **Watchdog detected failures**
- **Microprocessor failure**
- **PowerUp and PowerDown conditions**
- **System reset and hardware/software initialization**
- **PC tool initiated**

NOTE—Additional watchdog details and any exceptions to these failure states are specified in the related section of the product manual.

NOTICE **Battery Charging Device**	**To prevent damage to a control system that uses an alternator or battery-charging device, make sure the charging device is turned off before disconnecting the battery from the system.**

Regulatory Compliance

The LESP is partially completed Motohawk blocks meant to assist end users in designing their applications. As such, the LESP is a partially complete subsystem of individual parts. The device/system it is used in is only complete when the Motohawk application is integrated with reciprocating engines on or in completed “Engine Packages”, “Engine Skids”, “Construction, Earth Moving, Agricultural, Forestry, & Offroad Vehicles & Equipment”, “Semi-Mobile Industrial” (small installations that are fixed during operation), and stationary industrial “Fixed Installations”.

The LESP is a system of individually qualified devices; most do not have a function unless integrated into the fully operating package or vehicle.

Regulatory Compliance for each country that the finished device will ship into or be retrofit is the responsibility of the OEM or Aftermarket Retrofit Contractor. This includes the three essential items of marking the finished product, creating regulatory documentation, and any evaluation of the finished device/product.

The LESP has no regulatory marking requirements since it is a part of the system and performs no function on its own. **The OEM or Aftermarket Retrofit Contractor are responsible for any marking required by the governing body for the location or locations of operation.**

For the various parts mentioned in the following manual, please refer to the appropriate hardware manuals for regulatory compliance.

Manual #	**Description**
03425	**LECM Product Spec**
26757	**Large Engine Control Module**
35014	**LECM Platform EID MixedMode Driver**
35079	**LECM Aux RTCDC**
35079	**RT-CDC**
35100	**LECM Platform EID Ignition Driver**
35112	**LECM Aux Thermocouple**
35138	**LECM Aux Platform Knock**
35175	**LECM Platform EID Injection Driver**
35188	**LECM Aux Protection IO**
35200	**MotoHawk Developer Guide**
35203V1	**LESP Packages Developer Manual**
35203V2	**LESP Software Features Developer Manual**

Chapter 1. Overview

Introduction

The following application developer's manual provides detailed instructions and an in-depth understanding of developing a MotoHawk-based application from an initial Large Engine System Platform (LESP) starter package. Woodward MotoHawk training is a pre-requisite to comprehending the material within this manual. Please refer to MotoHawk Development Guide 35200 or contact your provider for MotoHawk training. The content in this section contains a high-level description of different LESP-based starter packages and a general overview of each offering.

LECM IO Starter Package

The LECM IO package contains a basic Woodward LECM IO model, LESP-standard model architecture, and LESP framework dependencies for getting started with development on the Woodward Large Engine Control Module (LECM). As illustrated below, the basic model architecture/layout, hardware input reads, and hardware output writes are complete. The input processing (hardware → engineering conversion), control logic, and output processing (engineering commands → hardware conversion) containers are left blank for customer development.

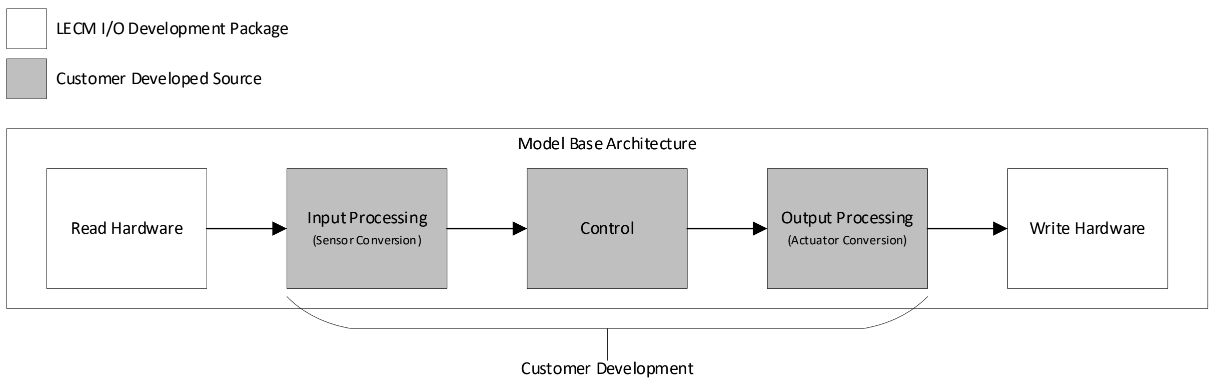


Figure 1-1. LECM IO Starter Package Overview

The package includes:

- LESP standard OS Tasking/Events
- LESP standard diagnostics management
- LESP standard HMI communications
- LECM Analog + Discrete + Temperature inputs
- LECM Analog + Discrete + Trigger outputs
- Woodward LECM Aux/EID standard interfacing
- LECM Standard Encoder System
    - 1-, 2-, or 3-sensor encoder system support
    - Standard pattern support (NX, SingleTooth, N-M, N+1)
    - Failover to Cam capabilities
    - Encoder emulation

LECM DF CRS µP Starter Package

The LECM DF CRS µP package contains the basic LECM IO package as described above; however, in addition, it includes pre-built software features (see 35203V2 for individual feature details). This package provides a more comprehensive engine package that allows customer development to focus on customizations and other rapid prototyping development rather than basic engine controls and features.

- LECM IO Package Contents
- Standard DF Sensors with Input Processing
- Engine State Management
- Speed Controller
- Common Rail Pressure Control + WLO FMV Controller
- LECM Aux Platform Acoustic Mitigation

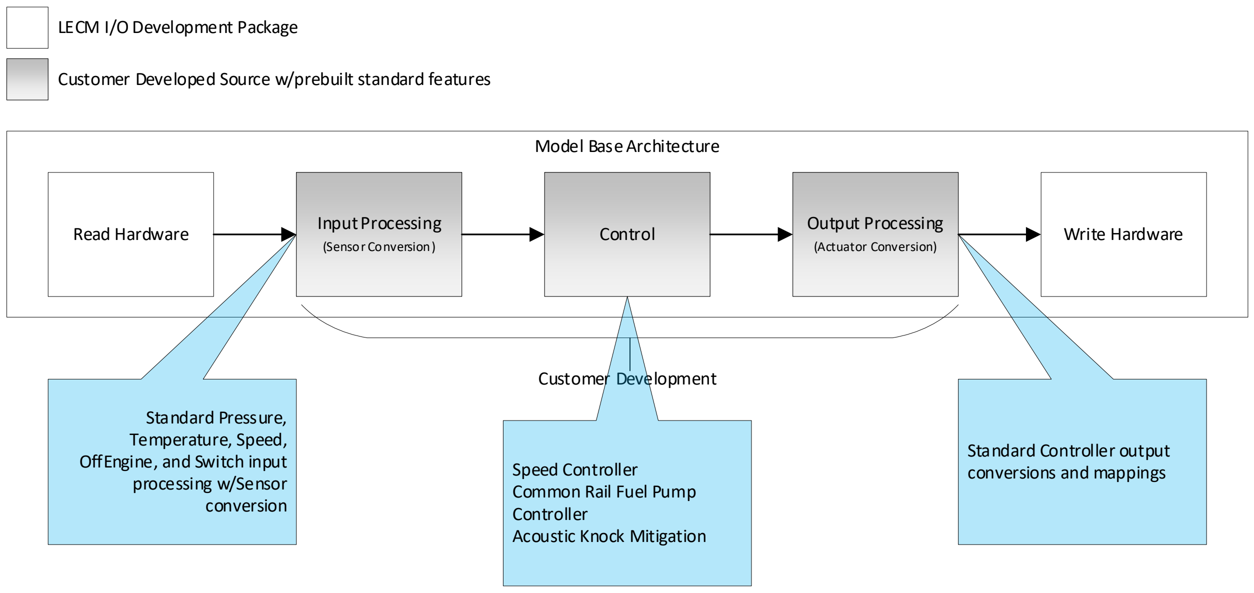

LECM I/O Development Package
Customer Developed Source w/prebuilt standard features
Model Base Architecture
Read Hardware
Input Processing (Sensor Conversion)
Control
Output Processing (Actuator Conversion)
Write Hardware
Customer Development
Standard Pressure, Temperature, Speed, OffEngine, and Switch input processing w/Sensor conversion
Speed Controller
Common Rail Fuel Pump Controller
Acoustic Knock Mitigation
Standard Controller output conversions and mappings

Figure 1-2. LECM CRS DF µP Package Overview

Application Development Environment

Before getting started, it’s important to ensure the application development environment and all dependencies are properly installed and licensed.

- MathWorks MATLAB/Simulink
  - See individual package release notes for version requirements
- MathWorks StateFlow Toolbox
- MathWorks Simulink Coder Toolbox
- MathWorks Embedded Coder Toolbox
- Woodward MotoHawk
  - See individual package release notes for version requirements
- Woodward Application Support Blocks (ASB) License
  - Contact provider for licensing. Licensing authorization can be accessed at https://www.woodward.com
- Woodward ToolKit with Developer License
  - Contact provider for licensing. Licensing authorization can be accessed at https://www.woodward.com

IMPORTANT **MATLAB/Simulink and Woodward MotoHawk training is a pre-requisite to understanding the subsequent material of this manual. Please ensure a fundamental understanding of the MATLAB/Simulink ADE and application development with Woodward MotoHawk has been taken into consideration before proceeding. Contact your provider for more information on MotoHawk training.**

Licensing Checklist

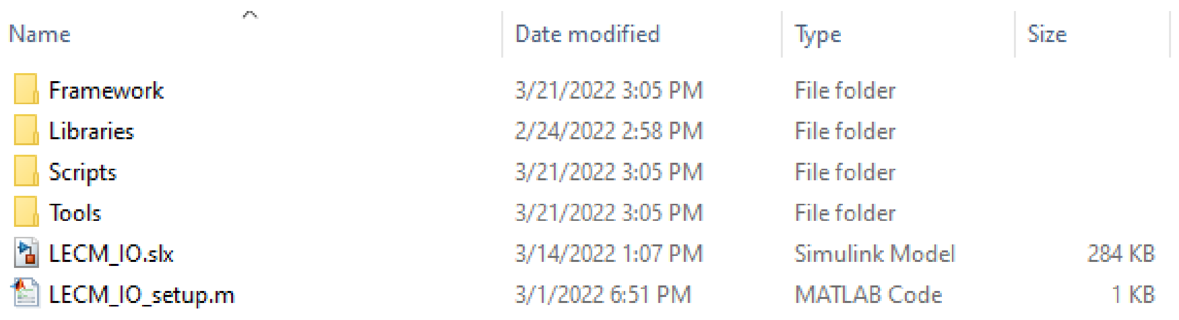
- ☐ Refer to the package release notes for Matlab/Simulink version requirements and ensure all required toolboxes are licensed and installed as indicated above.
- ☐ Refer to package release notes for MotoHawk version requirements. Use Windows Start Menu → MotoTools → License Viewer to verify MotoHawk licensing. See Manual 35200 for more details on MotoHawk licensing.
- ☐ Open Matlab and the package model. Execute ">> asb_license_manager" in the Matlab command line or navigate to the Model/Build Configuration/ASB License block and enter the ASB License key for each respective licensed PC node.
- ☐ Open ToolKit and navigate to the Tools Ribbon → License Authorization button and enter the ToolKit License key for each respective licensed PC node.

Understanding Project Directory Structure

First, it is important to understand the initial project directory layout. An LESP package is distributed in two archive files:

- <packagename>_framework_<version>.zip
    - Example: LECM_IO_framework_1_1_1.zip
- <packagename>_package_<version>.zip
    - Example: LECM_IO_package_1_1_1.zip

The version number is of the form Major_Minor_Build. The **framework** archive is used for upgrading the LESP framework libraries and dependencies within a local project and is only required if performing an upgrade. The **package** archive contains ALL of the files necessary to get started, including the framework files, the initial model, and local model supporting files. If getting started with a new project, extract the **package** archive; otherwise, if upgrading an existing project, use the **framework** archive. Once the package is extracted to a desired directory, the output should contain the following structure:



Name	Date modified	Type	Size
Framework	3/21/2022 3:05 PM	File folder	
Libraries	2/24/2022 2:58 PM	File folder	
Scripts	3/21/2022 3:05 PM	File folder	
Tools	3/21/2022 3:05 PM	File folder	
LECM_IO.slx	3/14/2022 1:07 PM	Simulink Model	284 KB
LECM_IO_setup.m	3/1/2022 6:51 PM	MATLAB Code	1 KB

Figure 1-3. LESP Package Initial Directory Structure

- **[Folder – ReadOnly] Framework**
    - The **Framework** folder contains all of the LESP framework libraries and supporting scripts. This folder contents should be considered readonly and NEVER modified unless upgraded with newer versions of the <packagename>_framework_<version> archive. Modifications to contents within this folder will be overridden when upgrading; therefore, if

there is a need to make a change, careful tracking of changes is required OR customizations should be moved to local supporting files instead.

- **[Folder – ReadWrite] Libraries**
    - The **Libraries** folder is used to host any local project-specific Simulink libraries outside the scope of the LESP framework. This directory will only be updated at the user's discretion and not during upgrades.
- **[Folder – ReadWrite] Scripts**
    - The **Scripts** folder is used to host any local project-specific MATLAB scripts or supporting files outside the scope of the LESP framework. This directory will only be updated at the user's discretion and not during upgrades.
- **[Folder – ReadWrite] Tools**
    - The **Tools** folder is used to host any local project-specific HMI or tool files outside the scope of the LESP framework. This directory will only be updated at the user's discretion and not during upgrades.
- **[File – ReadWrite] <ModelName>.slx**
    - The <ModelName>.slx file contains the initial starting point for the MotoHawk application model. This file should be renamed to a desired application name.
- **[File – ReadWrite] <ModelName>_setup.m**
    - The <ModelName>_setup.m file is executed from the <ModelName>.slx preload model callback function and is used to configure the environment by adding MATLAB search paths and initializing the workspace when the model is first loaded. The filename should match the *<ModelName>_setup.m* format. This file can be modified to further expand on initialization steps required during model development.

IMPORTANT

Before making any project-specific changes, it is highly recommended to confirm the initial package opens, updates, and builds successfully:

1. **Open model**
2. **Confirm model updates successfully (Ctrl+D)**
3. **Confirm model builds successfully (Ctrl+B) or provided use local_build script**

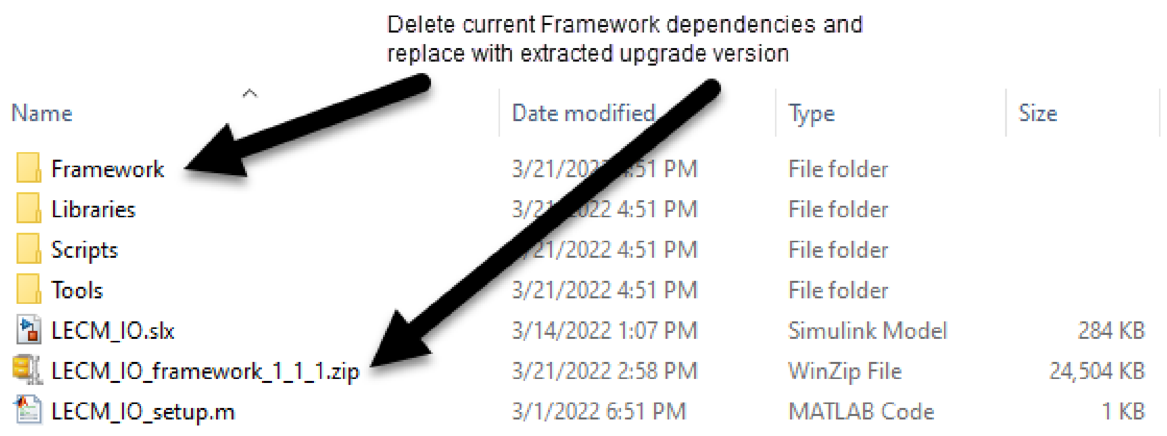
It's noteworthy to mention that the format of a project directory is often based on developer preferences and styles. There is no limitation to using a different directory format if desired; however, it's recommended to use the structure suggested in this document in order to follow further outlined instructions. The key instruction to follow is to not modify the LESP **Framework** folder in order to support upgrades in the future.

Upgrading Project Framework Dependencies

As previously mentioned, within a project directory, a folder named **Framework** contains all LESP-related package dependencies, including Simulink libraries, Matlab scripts, enumerations, etc. As part of a package distribution, the framework dependencies are archived independently in a *<packagename>_framework_<version>* archive file. To update a local project's framework dependencies, delete the existing **Framework** folder within the project's directory, and extract the newer *<packagename>_framework_<version>* archive file contents to a new local **Framework** project folder.

IMPORTANT

Note: Be sure the default framework folder is named Framework so it will be added to the Matlab search path using the default Model_setup.m script.


Delete current Framework dependencies and replace with extracted upgrade version

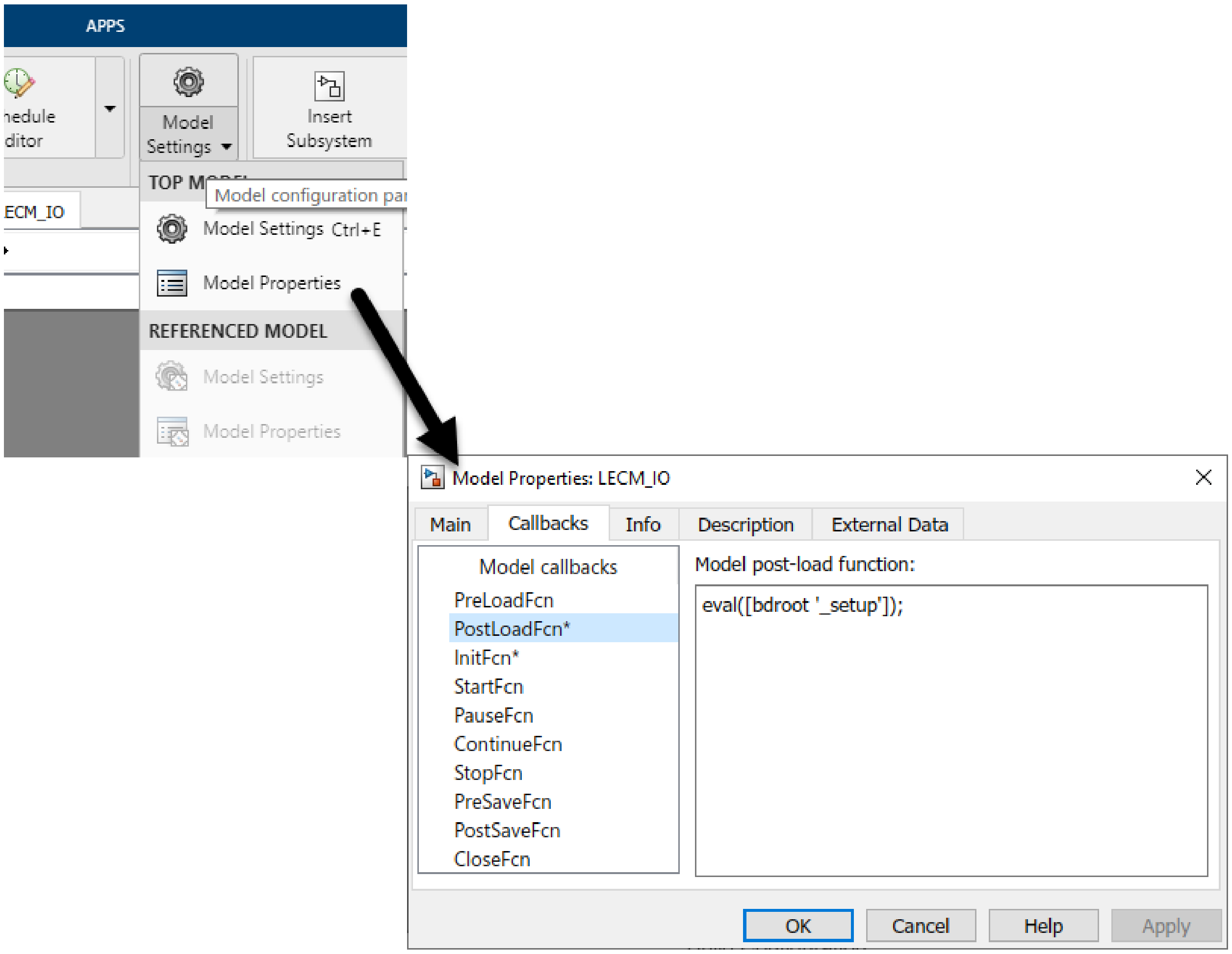
Name	Date modified	Type	Size
Framework	3/21/2022 4:51 PM	File folder	
Libraries	3/21/2022 4:51 PM	File folder	
Scripts	3/21/2022 4:51 PM	File folder	
Tools	3/21/2022 4:51 PM	File folder	
LECM_IO.slx	3/14/2022 1:07 PM	Simulink Model	284 KB
LECM_IO_framework_1_1_1.zip	3/21/2022 2:58 PM	WinZip File	24,504 KB
LECM_IO_setup.m	3/1/2022 6:51 PM	MATLAB Code	1 KB

Figure 1-4. Upgrading Framework Dependencies

- What if one of the framework files contains a software issue?
    - If there is an issue related to a framework dependency file, a patched or modified file with the same name that is higher on the Matlab path (e.g., local project directory ./Scripts or ./Libraries path) can be used to override (shadow) the framework file until the framework can be upgraded.

Closer Look: Model_setup.m

Upon opening the model, the model postload callback function will execute any script with matching <ModelName>_setup.m that is on the Matlab path.



Model post-load function:

```
eval([bdroot '_setup']);
```

Figure 1-5. Model_setup.m Script

By default, this script should be located alongside the model and is used to configure the application development environment and load any project-specific content. The script performs three important actions. First, it adds search paths of the local Framework, Libraries, and Scripts folders. Second, it populates the workspace with four project-specific parameters: OSConfig, HardwareConfig, SpeedConfig, and EncoderConfig structures. These structs are used to control global parameter settings as well as Simulink variant selections.

- OSConfig
    - Defines the OS fast, medium, and slow task rates.
- HardwareConfig
    - Defines the hardware target that is used to drive specific LESP blockset variants. This is currently extracted directly from the selection set in the MotoHawk Target Definition block using the lesp_get_hardware_target script.
- SpeedConfig
    - This struct is used to define the current LECM system variant for which frequency/duty cycle speed definitions should be defined. For example, even though the target may be an LECM 3-Stack, the encoder may consume the Main and EID speed resources; therefore, only the Aux speed resource variant is selected.
- EncoderConfig
    - This struct is used to define the encoder system stroke (2 or 4) and the selected LECM encoder variant (if applicable).
- BuildConfig

  - This struct is used to drive the build versioning with a Major/Minor/Build number versioning scheme. This struct must be manually created by the user.

Lastly, all framework workspace variables are loaded to the Matlab workspace with the lesp_load_all_lib_data function. Any additional search paths to be added, data to be loaded to the base workspace, or any other environment initialization actions should be added to this script.

```
%%
localdir = fileparts(mfilename('fullpath'));

%% Add Paths
addpath(genpath(fullfile(localdir, 'Libraries')));
addpath(genpath(fullfile(localdir, 'Scripts')));
addpath(genpath(fullfile(localdir, 'Framework')));

%% Create Configurations
OSConfig = struct('FastRateSec', 0.005, ...
    'MediumRateSec', 0.010, ...
    'SlowRateSec', 0.050);
if ~exist('HardwareConfig', 'var')
    HardwareConfig = struct('Target', lesp_get_hardware_target);
end
if ~exist('SpeedConfig', 'var')
    SpeedConfig = struct('System', LecmSpeedInDefn.AUX);
end
if ~exist('EncoderConfig', 'var')
    EncoderConfig = struct('Stroke', uint8(4), ...
        'System', LecmFlexEncoderSystemsDefn.LECM_MAINEID_2STACK);
end
if ~exist('BuildConfig', 'var')
    BuildConfig = struct('Major', 0, ...
        'Minor', 0, ...
        'Build', 0);
end

%% Load Library Data
lesp_load_all_lib_data;

clear localdir;
```

Enumerations

The LE system platform implements enumeration definitions in scripts (functions). Each script can be called with the string value of the enumeration and returns the corresponding numeric value or vice versa. This enables enumerations to be referenced directly in the model without hardcoded numeric values.

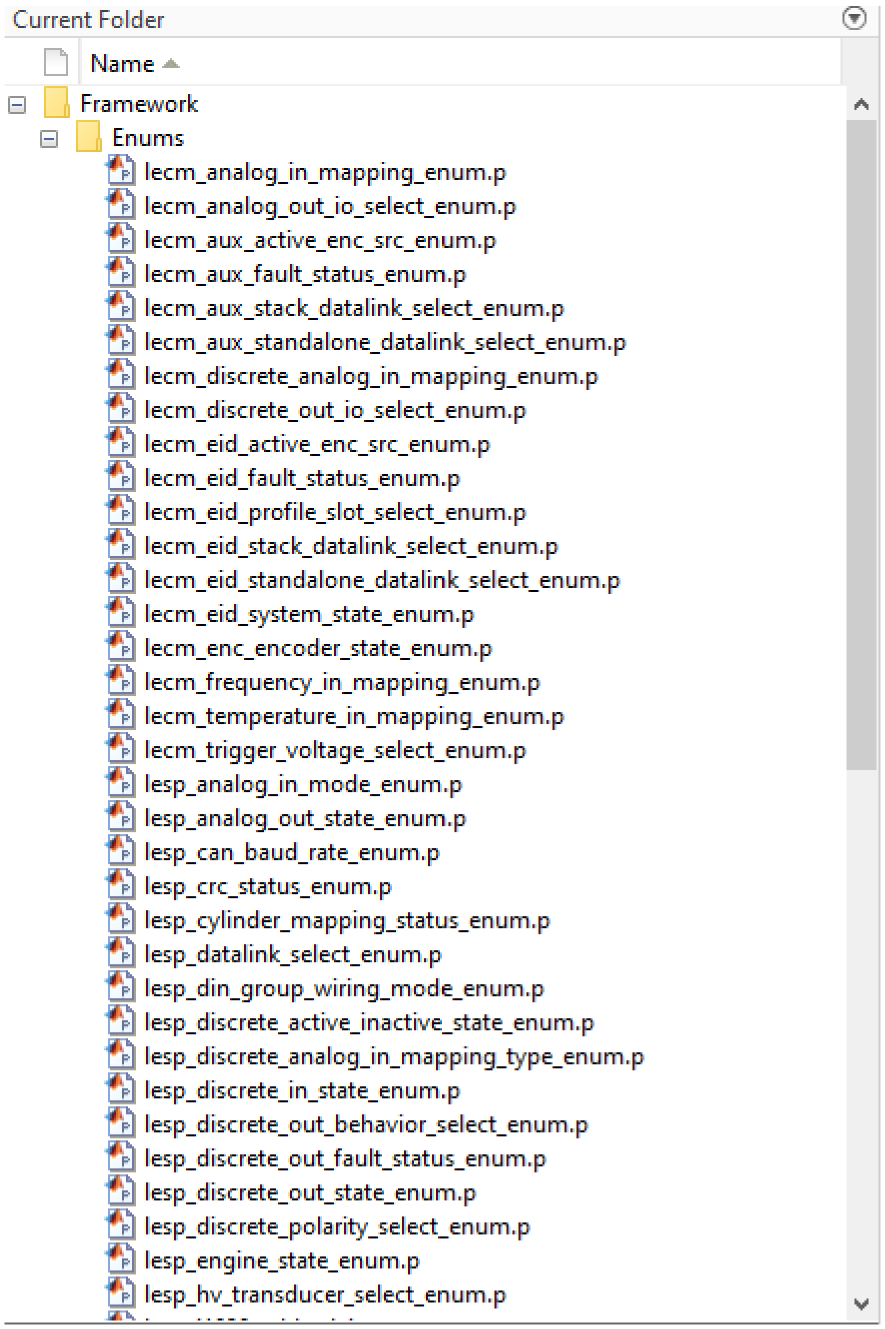
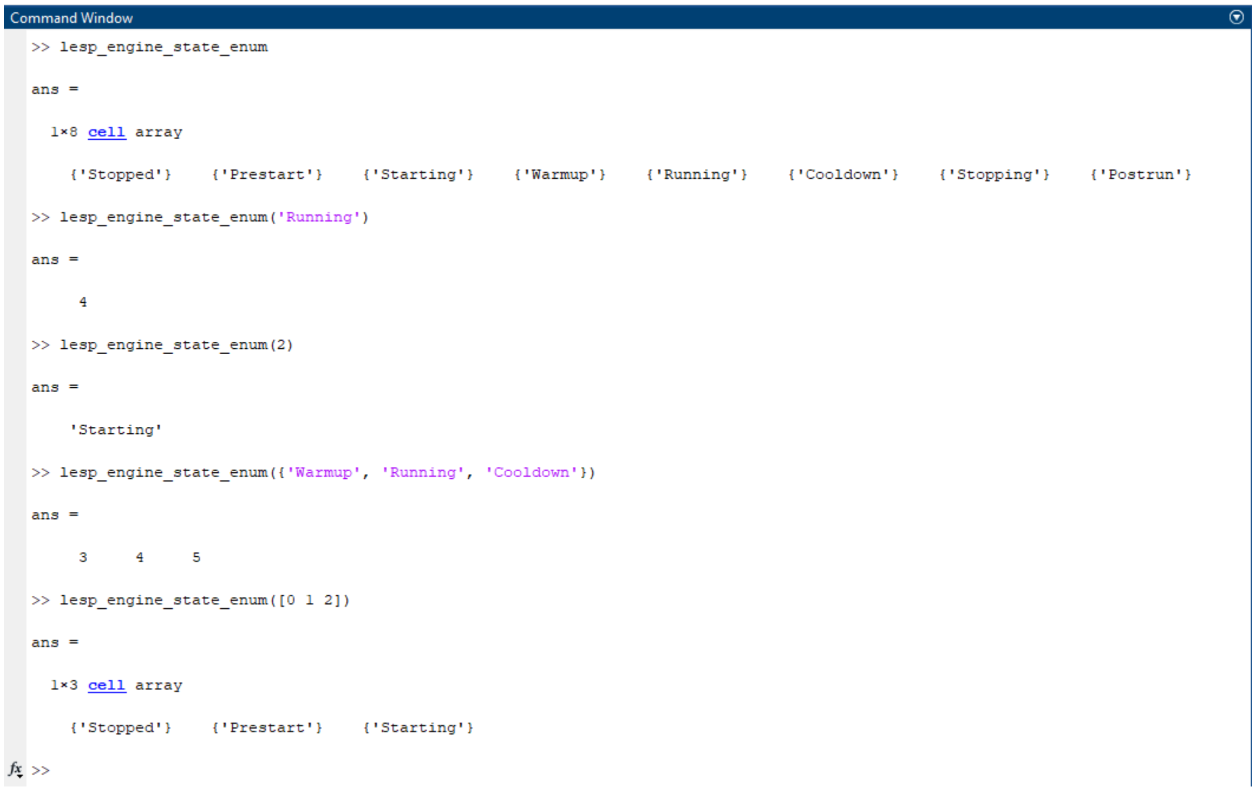

Current Folder
Name
Framework
Enums
lecm_analog_in_mapping_enum.p
lecm_analog_out_io_select_enum.p
lecm_aux_active_enc_src_enum.p
lecm_aux_fault_status_enum.p
lecm_aux_stack_datalink_select_enum.p
lecm_aux_standalone_datalink_select_enum.p
lecm_discrete_analog_in_mapping_enum.p
lecm_discrete_out_io_select_enum.p
lecm_eid_active_enc_src_enum.p
lecm_eid_fault_status_enum.p
lecm_eid_profile_slot_select_enum.p
lecm_eid_stack_datalink_select_enum.p
lecm_eid_standalone_datalink_select_enum.p
lecm_eid_system_state_enum.p
lecm_enc_encoder_state_enum.p
lecm_frequency_in_mapping_enum.p
lecm_temperature_in_mapping_enum.p
lecm_trigger_voltage_select_enum.p
lesp_analog_in_mode_enum.p
lesp_analog_out_state_enum.p
lesp_can_baud_rate_enum.p
lesp_crc_status_enum.p
lesp_cylinder_mapping_status_enum.p
lesp_datalink_select_enum.p
lesp_din_group_wiring_mode_enum.p
lesp_discrete_active_inactive_state_enum.p
lesp_discrete_analog_in_mapping_type_enum.p
lesp_discrete_in_state_enum.p
lesp_discrete_out_behavior_select_enum.p
lesp_discrete_out_fault_status_enum.p
lesp_discrete_out_state_enum.p
lesp_discrete_polarity_select_enum.p
lesp_engine_state_enum.p
lesp_hv_transducer_select_enum.p

Figure 1-6. Framework Enumerations Example

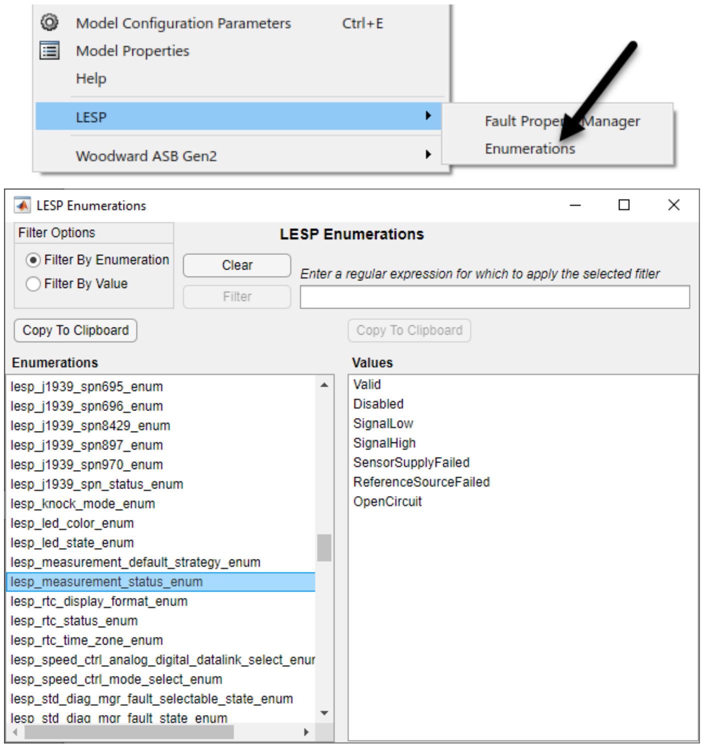
The below example illustrates the lesp_engine_state_enum script. When no arguments are supplied, the enumeration as originally defined is returned. If a string or cell array of strings is input, the numeric value(s) are returned respectively. If the numeric value is input, the string(s) are returned respectively.



```
Command Window
>> lesp_engine_state_enum

ans =

  1×8 cell array

    {'Stopped'}    {'Prestart'}    {'Starting'}    {'Warmup'}    {'Running'}    {'Cooldown'}    {'Stopping'}    {'Postrun'}

>> lesp_engine_state_enum('Running')

ans =

     4

>> lesp_engine_state_enum(2)

ans =

    'Starting'

>> lesp_engine_state_enum({'Warmup', 'Running', 'Cooldown'})

ans =

     3     4     5

>> lesp_engine_state_enum([0 1 2])

ans =

  1×3 cell array

    {'Stopped'}    {'Prestart'}    {'Starting'}

fx >>
```

Figure 1-7. Enumeration Example

To review all available enumerations, LESP supports an “Enumeration” context menu, which will open a dialog that searches for all scripts currently on the Matlab path that end with _enum.m[p] and return an expected enumeration. This tool can be used to review and search available enumerations.



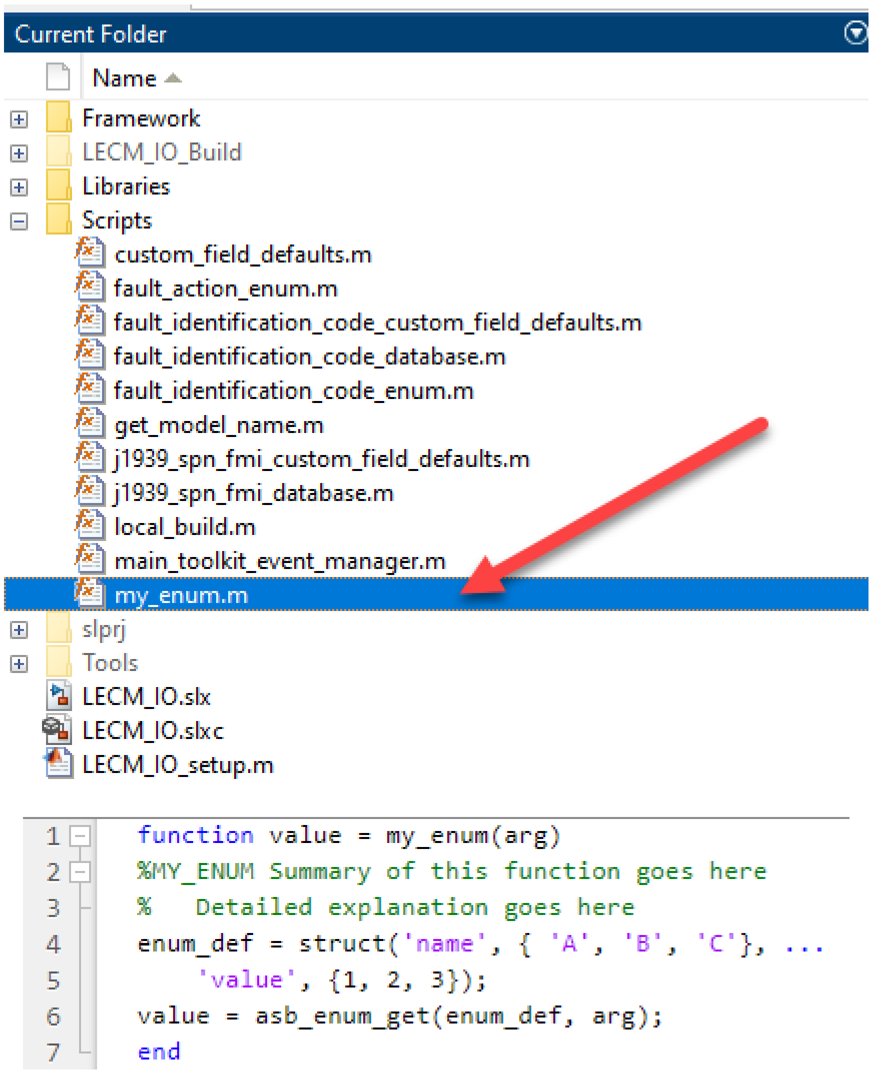
lesp_j1939_spn695_enum
lesp_j1939_spn696_enum
lesp_j1939_spn8429_enum
lesp_j1939_spn897_enum
lesp_j1939_spn970_enum
lesp_j1939_spn_status_enum
lesp_knock_mode_enum
lesp_led_color_enum
lesp_led_state_enum
lesp_measurement_default_strategy_enum
lesp_measurement_status_enum
lesp_rtc_display_format_enum
lesp_rtc_status_enum
lesp_rtc_time_zone_enum
lesp_speed_ctrl_analog_digital_datalink_select_enur
lesp_speed_ctrl_mode_select_enum
lesp_std_diag_mgr_fault_selectable_state_enum
lesp_std_diag_mgr_fault_state_enum

Valid
Disabled
SignalLow
SignalHigh
SensorSupplyFailed
ReferenceSourceFailed
OpenCircuit

Figure 1-8. LESP Enumerations Dialog

Defining An Enumeration

To define your own custom enumeration that utilizes the same structure, define a function using a name for the enumeration. Define your enumeration using either a cell string array for zero-based numeric values, a struct with name and value fields for explicit name-value pairs, or by utilizing a motohawk_enum object.



```
function value = my_enum(arg)
%MY_ENUM Summary of this function goes here
%   Detailed explanation goes here
enum_def = struct('name', { 'A', 'B', 'C'}, ...
    'value', {1, 2, 3});
value = asb_enum_get(enum_def, arg);
end
```

Figure 1-9. My Enumeration Definition

Subsequently, in the model, you can use ASB Enum Constant, Enum Comparison blocks, and other standard Simulink blocks to access the enumeration values by name, allowing for the enumeration to be modified in a single location if needed in addition to easier model source code to interpret.

LESP Custom DataTypes

As part of the LE system platform, a number of custom datatypes are defined and used within the model. These compound types are used as interface definitions for standard LESP blocks and are defined as busses that are loaded in the Matlab workspace during the model setup (*lesp_load_all_lib_data*). Note: Struct datatypes that end in Bus (e.g., AlwaysActiveDiscreteInBus, AlwaysInactiveDiscreteInBus, DefaultDiscreteInBus, DefaultAnalogSensorBus, etc...) are bus constants that match the corresponding bus type's structure and can be used with a Simulink Constant block to output a nonvirtual constant valued bus.

Examples:

```
DefaultDiscreteInBus = struct('DiscreteState', false, ...
    'Frequency_Hz', single(0), ...
    'DutyCycle_Pct', single(0));

AlwaysActiveDiscreteInBus = struct('DiscreteState', true, ...
    'Frequency_Hz', single(0), ...
    'DutyCycle_Pct', single(0));

AlwaysInactiveDiscreteInBus = struct('DiscreteState', false, ...
    'Frequency_Hz', single(0), ...
    'DutyCycle_Pct', single(0));

DefaultAnalogSensorBus = struct('FilteredValue', single(0), ...
    'MeasurementStatus_enum', uint8(lesp_measurement_status_enum('Disabled')));
```

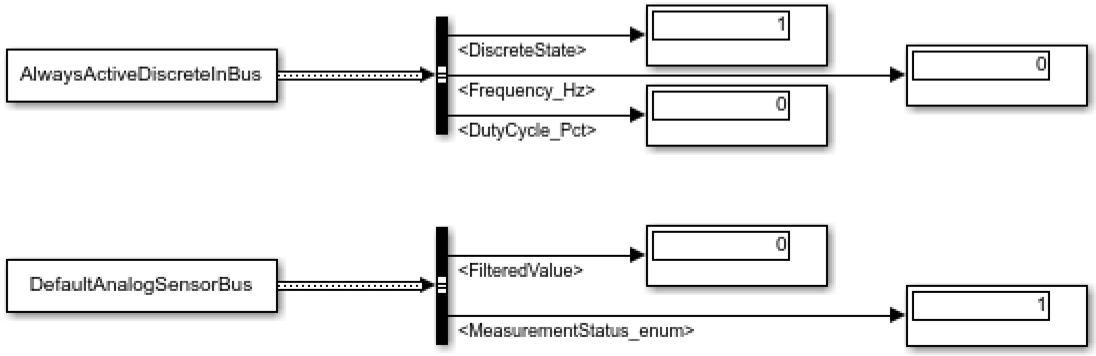
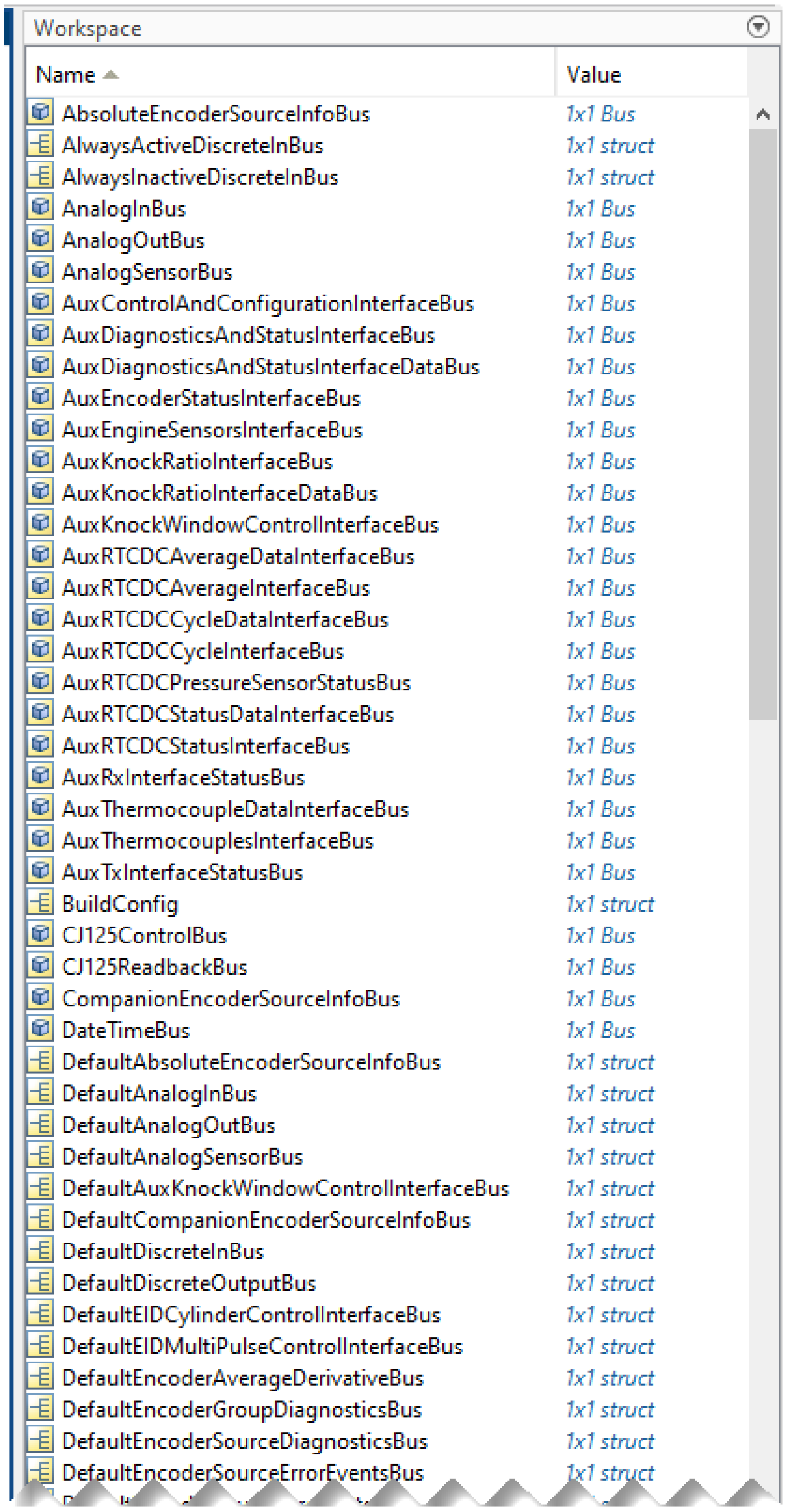


Figure 1-10. Bus Constants



Workspace

Name	Value
AbsoluteEncoderSourceInfoBus	1x1 Bus
AlwaysActiveDiscreteInBus	1x1 struct
AlwaysInactiveDiscreteInBus	1x1 struct
AnalogInBus	1x1 Bus
AnalogOutBus	1x1 Bus
AnalogSensorBus	1x1 Bus
AuxControlAndConfigurationInterfaceBus	1x1 Bus
AuxDiagnosticsAndStatusInterfaceBus	1x1 Bus
AuxDiagnosticsAndStatusInterfaceDataBus	1x1 Bus
AuxEncoderStatusInterfaceBus	1x1 Bus
AuxEngineSensorsInterfaceBus	1x1 Bus
AuxKnockRatioInterfaceBus	1x1 Bus
AuxKnockRatioInterfaceDataBus	1x1 Bus
AuxKnockWindowControlInterfaceBus	1x1 Bus
AuxRTCDCAverageDataInterfaceBus	1x1 Bus
AuxRTCDCAverageInterfaceBus	1x1 Bus
AuxRTCDCCycleDataInterfaceBus	1x1 Bus
AuxRTCDCCycleInterfaceBus	1x1 Bus
AuxRTCDCPressureSensorStatusBus	1x1 Bus
AuxRTCDCStatusDataInterfaceBus	1x1 Bus
AuxRTCDCStatusInterfaceBus	1x1 Bus
AuxRxInterfaceStatusBus	1x1 Bus
AuxThermocoupleDataInterfaceBus	1x1 Bus
AuxThermocouplesInterfaceBus	1x1 Bus
AuxTxInterfaceStatusBus	1x1 Bus
BuildConfig	1x1 struct
CJ125ControlBus	1x1 Bus
CJ125ReadbackBus	1x1 Bus
CompanionEncoderSourceInfoBus	1x1 Bus
DateTimeBus	1x1 Bus
DefaultAbsoluteEncoderSourceInfoBus	1x1 struct
DefaultAnalogInBus	1x1 struct
DefaultAnalogOutBus	1x1 struct
DefaultAnalogSensorBus	1x1 struct
DefaultAuxKnockWindowControlInterfaceBus	1x1 struct
DefaultCompanionEncoderSourceInfoBus	1x1 struct
DefaultDiscreteInBus	1x1 struct
DefaultDiscreteOutputBus	1x1 struct
DefaultEIDCylinderControlInterfaceBus	1x1 struct
DefaultEIDMultiPulseControlInterfaceBus	1x1 struct
DefaultEncoderAverageDerivativeBus	1x1 struct
DefaultEncoderGroupDiagnosticsBus	1x1 struct
DefaultEncoderSourceDiagnosticsBus	1x1 struct
DefaultEncoderSourceErrorEventsBus	1x1 struct

Figure 1-11. LESP Custom DataType Examples

Below are a few commonly used bus structures:

AnalogInBus

The AnalogInBus is used for all AIN input signals with the following bus structure:

- AnalogInBus

- Mode_enum (uint8)
  - The input mode of the AIN input (e.g., 4-20 mA input, 0-5V input, +- 2.5V input, etc.
  - See *lesp_analog_in_mode_enum* script for supported modes and enumeration definition
- Value (single)
  - The hardware units value of the input (V, mA, ...)
- ADC (uint16)
  - The hardware ADC counts of the input
- ADCToHwUnitsGain (single)
  - The gain used to convert from ADC to hardware units (V, mA, ...)
- ADCToHWUnitsOffset (single)
  - The offset used to convert from ADC to hardware units (V, mA)
- PullupResistorValue_ohms (single)
  - The value of the pullup resistor of the input if applicable

AnalogOutBus

The AnalogOutBus is used to command supported AOUTs on the module with the following bus structure:

- AnalogOutBus
  - IOSelect_enum (uint8)
    - Dynamic IO mapping support. Enumeration is hardware-dependent.
    - See *lecm_analog_out_io_select_enum* enumeration for LECM hardware.
  - State_enum (uint8)
    - Value of Not Used, Inactive, or Active. See *lesp_analog_out_state_enum*.
  - CurrentDemand_uA (uint16)
    - The current demand for the analog output in uA.

AnalogSensorBus

The AnalogSensorBus is used for analog sensors in the input processing logic to report the following compound signal value for each applicable sensor:

- AnalogSensorBus
  - FilteredValue (single)
    - The final engineering value of the analog sensor (e.g., Ambient Air Pressure sensor filtered value in units of kPa).
  - MeasurementStatus_enum (uint8)
    - An enumerated value indicating the status of the FilteredValue signal. See *lesp_measurement_status_enum*.

DateTimeBus

The DateTime is used as a compound signal to report the current DateTime (e.g., from real-time clock logic, fault timestamps, etc.):

- DateTimeBus
  - Year (uint16)
  - Month (uint8)
  - Day (uint8)
  - Hour (uint8)
  - Minute (uint8)

    - Second (uint8)
    - Microsecond (uint32)

DiscreteAnalogInBus

The DiscreteAnalogInBus is used to combine AnalogInBus and DiscreteInBus input signals to allow for analog and discrete input bus types to be used for discrete logic with following bus structure:

- DiscreteAnalogInBus
    - Type_enum (uint8)
        - None, Discrete, or Analog. Used to indicate the type of hardware input this signal represents.
    - AnalogADC (uint16)
        - The analog input ADC count if the type is an Analog.
    - DiscreteState (boolean)
        - 0 = Inactive, 1 = Active if the type is Discrete

DiscreteInBus

The DiscreteInBus is used for all DIN input signals with the following bus structure:

- DiscreteInBus
    - DiscreteState (boolean)
        - 0 = Inactive, 1 = Active. This is used for state-based discrete inputs.
    - Frequency_Hz (single)
        - The frequency in Hz of the discrete input. This is used for frequency/duty-cycle based discrete inputs.
    - DutyCycle_Pct (single)
        - The duty cycle of the discrete input in units of percent. This is used for frequency/duty-cycle based discrete inputs.

DiscreteOutputBus

The DiscreteOutputBus is used for all digital output-based command signals with the following bus structure:

- DiscreteOutputBus
    - IOSelect_enum (uint8)
        - Dynamic IO mapping support. Enumeration is hardware-dependent.
        - See *lecm_discrete_out_io_select_enum* enumeration for LECM hardware
    - BehaviorSelect_enum (uint8)
        - Defines the behavior of the output as either Discrete or Pulse-Width-Modulated (PWM). See *lesp_discrete_out_behavior_select_enum* enumeration.
    - State_enum (uint8)
        - Value of Not Used, Inactive, or Active. See *lesp_discrete_out_state_enum*.
    - DutyCycle_Pct (single)
        - Duty cycle command of the output if configured with PWM behavior.
    - Frequency_Hz (single)
        - Frequency command of the output if configured with PWM behavior.

FaultBus

The FaultBus is used for LESP standard diagnostic management fault interfacing with the following bus structure:

- FaultBus
  - AssertionStatus (boolean)
    - True signal to indicate if the respective fault should be assert if not already asserted.
  - ConditionalStatus (boolean)
    - True signal to indicate the conditions for the fault to assert are true, independent of the actual fault state.
  - InhibitStatus (boolean)
    - True signal to indicate if conditions for the fault to be inhibited are valid.

LEDControlBus

The LEDControlBus is used controlling an LED with a standard LESP LED controller block with the following bus structure:

- LEDControlBus
  - Color_enum (uint8)
    - Indicates the desired color of the LED. See *lesp_led_color_enum* enumeration.
  - State_enum (uint8)
    - Defines the desired control state of the LED. See *lesp_led_state_enum* enumeration.
  - BlinkCode (uint8)
    - Indicates the desired blink code of the LED if the state is equal to “Code.”

LEDOutputBus

The LEDOutputBus is used controlling the state of an LED:

- LEDOutputBus
  - RedLEDState (boolean)
    - True if the Red color of the LED should be active.
  - GreenLEDState (boolean)
    - True if the Green color of the LED should be active.

TcRtdInBus

The TcRtdInBus is used for all Thermocouple/RTD input signals with the following bus structure:

- TcRtdInBus
  - Mode_enum (uint8)
    - The input mode of the TcRtd input (e.g., 0-5V input, Thermocouple input, RTD input, Datalink degC, etc.
    - See *lesp_tcrtd_in_mode_enum* script for supported modes and enumeration definition.
  - Status (uint8)
    - The status of the input as either Not Available, Error, Open, or Valid. See *lesp_tcrtd_in_status_enum*.
  - PullupResistorValue_ohms (single)
    - The value of the pullup resistor of the input if applicable.

  - Value (single)
    - The hardware units value of the input (degC).

TimeSpanBus

The TimeSpanBus is used as a compound signal to report the Timestamps (e.g., engine hour meter):

- TimeSpanBus
  - Days (int32)
  - Hours (int8)
  - Minutes (int8)
  - Seconds (int8)
  - Microseconds (int32)

Transducer5VoltBus

The Transducer5VoltBus is used for the transducer 5V power supply of the module:

- Transducer5VoltBus
  - FilteredValue_V (single)
    - The filtered transducer power supply voltage.
  - MeasurementStatus_enum (uint8)
    - An enumerated value indicating the status of the FilteredValue signal. See *lesp_measurement_status_enum*.
  - RatiometricCorrectionFactor (single)
    - A gain multiplier indicating the transducer voltage relative to the nominal expected voltage. This value can be used as a multiplier for ratiometric sensor types.
    - This multiplier is calculated based on the formula:

$$RatiometricFactor = \left\{\frac{Nominal\ Voltage}{Actual\ Voltage}\right\}_{0.95}^{1.05}$$

TransducerHighVoltBus

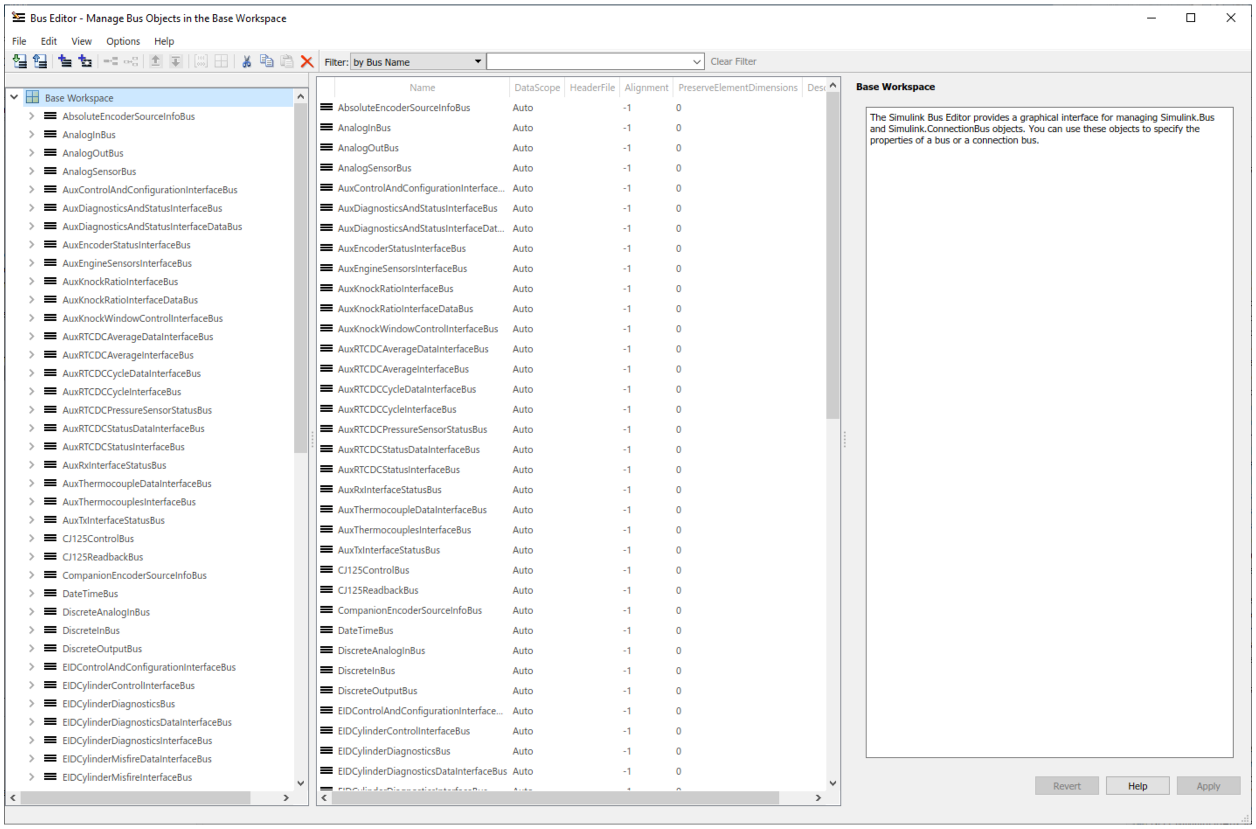
The TransducerHighVoltBus is used for the transducer HV power supply of the module:

- TransducerHighVoltBus
  - FilteredValue_V (single)
    - The filtered transducer power supply voltage
  - MeasurementStatus_enum (uint8)
    - An enumerated value indicating the status of the FilteredValue signal. See *lesp_measurement_status_enum*.
  - RatiometricCorrectionFactor (single)
    - A gain multiplier indicating the transducer voltage relative to the nominal expected voltage. This value can be used as a multiplier for ratiometric sensor types.
    - This multiplier is calculated based on the formula:

$$RatiometricFactor = \left\{\frac{Nominal\ Voltage}{Actual\ Voltage}\right\}_{0.95}^{1.05}$$

  - VoltageSelect_enum (uint8)
    - Indicates the current voltage selection of the transducer power supply, see *lesp_hv_transducer_select_enum*.

There are numerous other LECM platform EID/Aux bus definitions that describe the interfaces to EID/Aux platform application communications as described in more detail in subsequent chapters. To view the details of each bus definition, execute “>> buseditor” in the Matlab command line. New custom compound datatypes can be defined; however, do not modify the LESP-defined types as these will be restored when reloading the framework data. If defining custom bus types, be sure to save them to a file and reload by adding to the Model_setup.m script.



- Base Workspace
  - AbsoluteEncoderSourceInfoBus
  - AnalogInBus
  - AnalogOutBus
  - AnalogSensorBus
  - AuxControlAndConfigurationInterfaceBus
  - AuxDiagnosticsAndStatusInterfaceBus
  - AuxDiagnosticsAndStatusInterfaceDataBus
  - AuxEncoderStatusInterfaceBus
  - AuxEngineSensorsInterfaceBus
  - AuxKnockRatioInterfaceBus
  - AuxKnockRatioInterfaceDataBus
  - AuxKnockWindowControlInterfaceBus
  - AuxRTCDCAverageDataInterfaceBus
  - AuxRTCDCAverageInterfaceBus
  - AuxRTCDCCycleDataInterfaceBus
  - AuxRTCDCCycleInterfaceBus
  - AuxRTCDCPressureSensorStatusBus
  - AuxRTCDCStatusDataInterfaceBus
  - AuxRTCDCStatusInterfaceBus
  - AuxRxInterfaceStatusBus
  - AuxThermocoupleDataInterfaceBus
  - AuxThermocouplesInterfaceBus
  - AuxTxInterfaceStatusBus
  - CJ125ControlBus
  - CJ125ReadbackBus
  - CompanionEncoderSourceInfoBus
  - DateTimeBus
  - DiscreteAnalogInBus
  - DiscreteInBus
  - DiscreteOutputBus
  - EIDControlAndConfigurationInterfaceBus
  - EIDCylinderControlInterfaceBus
  - EIDCylinderDiagnosticsBus
  - EIDCylinderDiagnosticsDataInterfaceBus
  - EIDCylinderDiagnosticsInterfaceBus
  - EIDCylinderMisfireDataInterfaceBus
  - EIDCylinderMisfireInterfaceBus

Name	DataScope	HeaderFile	Alignment	PreserveElementDimensions	Desc
AbsoluteEncoderSourceInfoBus	Auto		-1	0	
AnalogInBus	Auto		-1	0	
AnalogOutBus	Auto		-1	0	
AnalogSensorBus	Auto		-1	0	
AuxControlAndConfigurationInterface...	Auto		-1	0	
AuxDiagnosticsAndStatusInterfaceBus	Auto		-1	0	
AuxDiagnosticsAndStatusInterfaceDat...	Auto		-1	0	
AuxEncoderStatusInterfaceBus	Auto		-1	0	
AuxEngineSensorsInterfaceBus	Auto		-1	0	
AuxKnockRatioInterfaceBus	Auto		-1	0	
AuxKnockRatioInterfaceDataBus	Auto		-1	0	
AuxKnockWindowControlInterfaceBus	Auto		-1	0	
AuxRTCDCAverageDataInterfaceBus	Auto		-1	0	
AuxRTCDCAverageInterfaceBus	Auto		-1	0	
AuxRTCDCCycleDataInterfaceBus	Auto		-1	0	
AuxRTCDCCycleInterfaceBus	Auto		-1	0	
AuxRTCDCPressureSensorStatusBus	Auto		-1	0	
AuxRTCDCStatusDataInterfaceBus	Auto		-1	0	
AuxRTCDCStatusInterfaceBus	Auto		-1	0	
AuxRxInterfaceStatusBus	Auto		-1	0	
AuxThermocoupleDataInterfaceBus	Auto		-1	0	
AuxThermocouplesInterfaceBus	Auto		-1	0	
AuxTxInterfaceStatusBus	Auto		-1	0	
CJ125ControlBus	Auto		-1	0	
CJ125ReadbackBus	Auto		-1	0	
CompanionEncoderSourceInfoBus	Auto		-1	0	
DateTimeBus	Auto		-1	0	
DiscreteAnalogInBus	Auto		-1	0	
DiscreteInBus	Auto		-1	0	
DiscreteOutputBus	Auto		-1	0	
EIDControlAndConfigurationInterface...	Auto		-1	0	
EIDCylinderControlInterfaceBus	Auto		-1	0	
EIDCylinderDiagnosticsBus	Auto		-1	0	
EIDCylinderDiagnosticsDataInterfaceBus	Auto		-1	0	

Base Workspace

The Simulink Bus Editor provides a graphical interface for managing Simulink.Bus and Simulink.ConnectionBus objects. You can use these objects to specify the properties of a bus or a connection bus.

Figure 1-12. Simulink Bus Editor

LESP User Blocks

Before continuing, it’s important to ensure the Simulink Library Browser has been refreshed and is actively showing all Woodward ASB and LESP libraries. This is accomplished by opening the Simulink Browser. If the LESP and ASB libraries are not all visible, hit the “F5” key to refresh the browser view, then select “Fix” and “Generate repositories in memory” option. This should expose all LESP and ASB libraries in the browser.

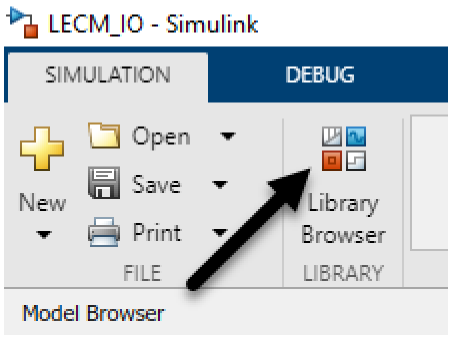
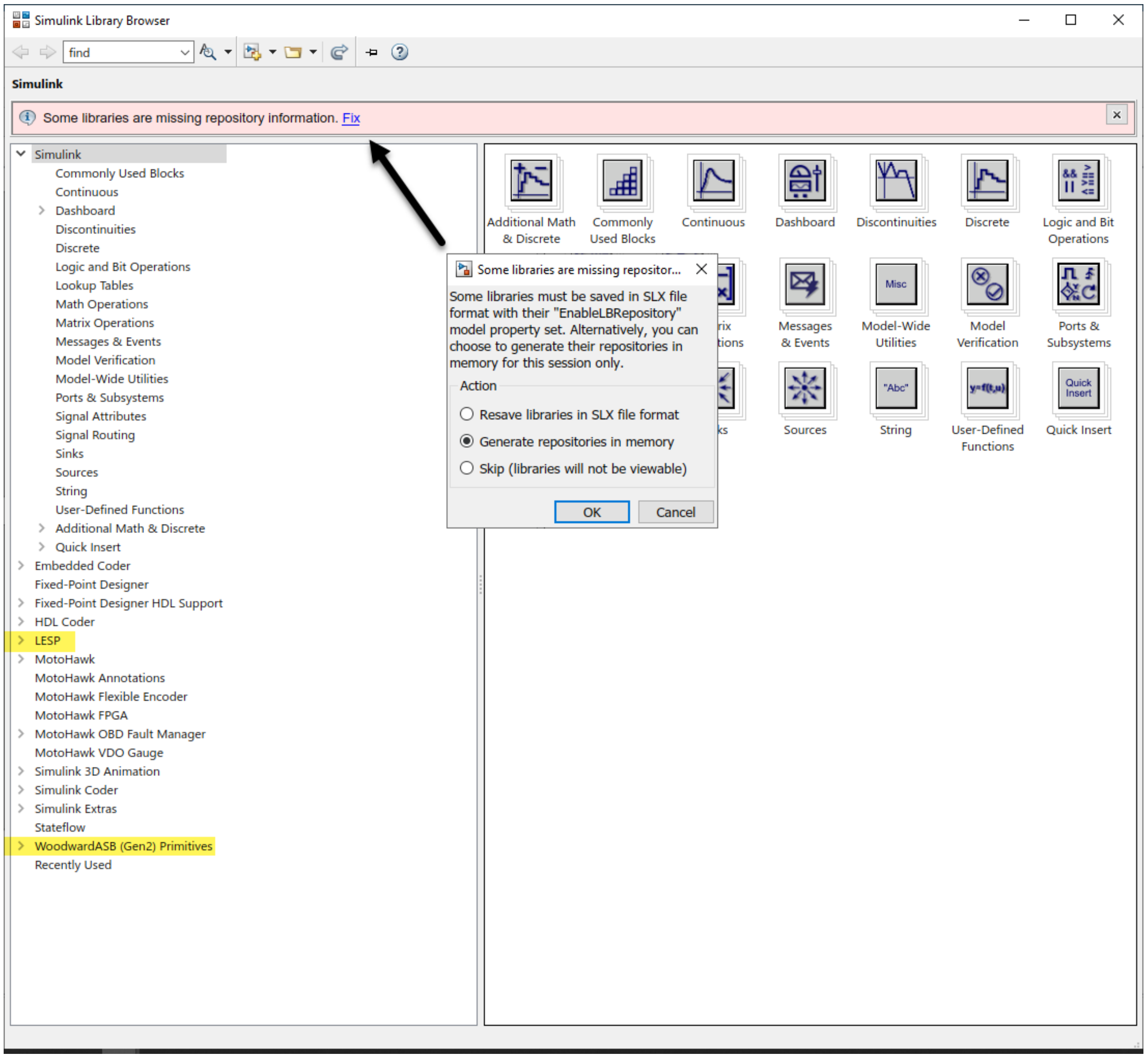


Figure 1-13. Open Simulink Library Browser



Simulink Library Browser

Simulink

Some libraries are missing repository information. Fix

Some libraries are missing repositor...

Some libraries must be saved in SLX file format with their "EnableLBRepository" model property set. Alternatively, you can choose to generate their repositories in memory for this session only.

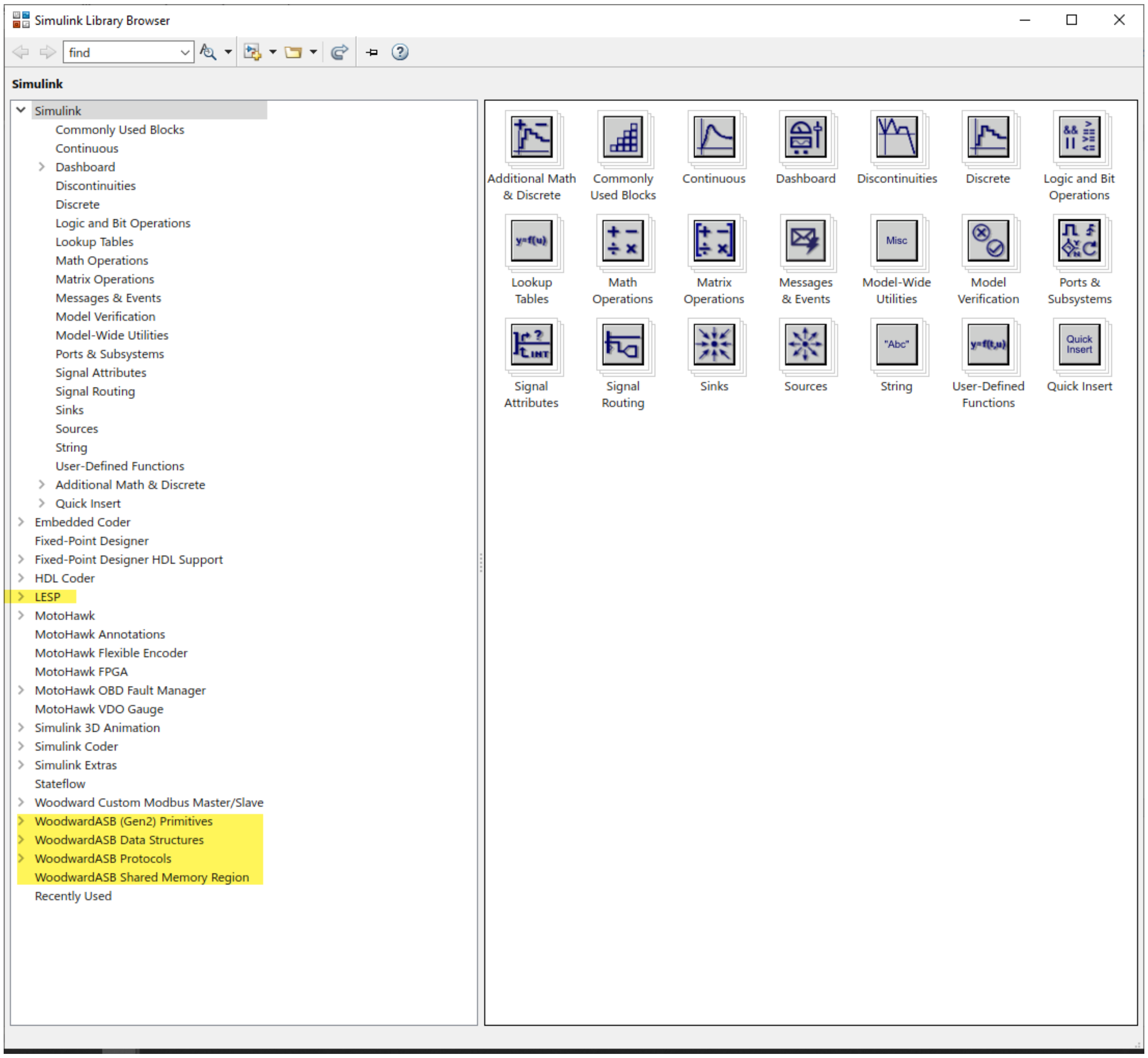
Action

- ○ Resave libraries in SLX file format
- ● Generate repositories in memory
- ○ Skip (libraries will not be viewable)

OK Cancel

- Simulink
  - Commonly Used Blocks
  - Continuous
  - Dashboard
  - Discontinuities
  - Discrete
  - Logic and Bit Operations
  - Lookup Tables
  - Math Operations
  - Matrix Operations
  - Messages & Events
  - Model Verification
  - Model-Wide Utilities
  - Ports & Subsystems
  - Signal Attributes
  - Signal Routing
  - Sinks
  - Sources
  - String
  - User-Defined Functions
  - Additional Math & Discrete
  - Quick Insert
- Embedded Coder
- Fixed-Point Designer
- Fixed-Point Designer HDL Support
- HDL Coder
- LESP
- MotoHawk
- MotoHawk Annotations
- MotoHawk Flexible Encoder
- MotoHawk FPGA
- MotoHawk OBD Fault Manager
- MotoHawk VDO Gauge
- Simulink 3D Animation
- Simulink Coder
- Simulink Extras
- Stateflow
- WoodwardASB (Gen2) Primitives
- Recently Used

Figure 1-14. [Before] Refreshing Simulink Library Browser



Simulink Library Browser

Simulink

- Simulink
  - Commonly Used Blocks
  - Continuous
  - Dashboard
  - Discontinuities
  - Discrete
  - Logic and Bit Operations
  - Lookup Tables
  - Math Operations
  - Matrix Operations
  - Messages & Events
  - Model Verification
  - Model-Wide Utilities
  - Ports & Subsystems
  - Signal Attributes
  - Signal Routing
  - Sinks
  - Sources
  - String
  - User-Defined Functions
  - Additional Math & Discrete
  - Quick Insert
- Embedded Coder
- Fixed-Point Designer
- Fixed-Point Designer HDL Support
- HDL Coder
- LESP
- MotoHawk
- MotoHawk Annotations
- MotoHawk Flexible Encoder
- MotoHawk FPGA
- MotoHawk OBD Fault Manager
- MotoHawk VDO Gauge
- Simulink 3D Animation
- Simulink Coder
- Simulink Extras
- Stateflow
- Woodward Custom Modbus Master/Slave
- WoodwardASB (Gen2) Primitives
- WoodwardASB Data Structures
- WoodwardASB Protocols
- WoodwardASB Shared Memory Region
- Recently Used

Figure 1-15. [After] Refreshing Simulink Library Browser

Chapter 2.
Model Architecture

Introduction

The following chapter provides a deeper understanding of an LESP-based standard application model layout for engine control software development. The sections will work *downward* through the model hierarchy, starting at the model root.

Model Root

The following illustration below shows a simple package example which includes the standard directory structure with the Libraries (1) subfolder used to hold any non-framework dependency libraries or project-specific libraries. The Scripts (2) subfolder contains local project-specific scripts. The Matlab workspace (3) contains the global definitions of Simulink Busses, Structs, System Variants, and other model accessible definitions. Most of these definitions are loaded by default during the model initialization phase as previously discussed. The root level of the model contains four subsystems:

- **Target Definition**
    - The target definition subsystem is a container for the MotoHawk target definition.
- **Build Configuration**
    - The build configuration subsystem contains blocks related to generic configuration of the model. Including the MotoHawk Toolchain Definition (compiler selection), Version Requirements block, ASB License, etc. These blocks do not need to be located within an execution context.
- **Model Configuration**
    - The model configuration subsystem contains definition-level blocks of features and blocksets used within the model. These blocks do not need to be located within an execution context as they simply define high level features, software stacks, and hardware configurations. These include blocks such as CAN Definitions, Protocol Definitions, Diagnostic & Management Definitions, etc.
- **System Model**
    - The system model subsystem contains the core execution logic of the model. The contents of this subsystem have a standard Simulink dataflow layout (Inputs → Control → Outputs).

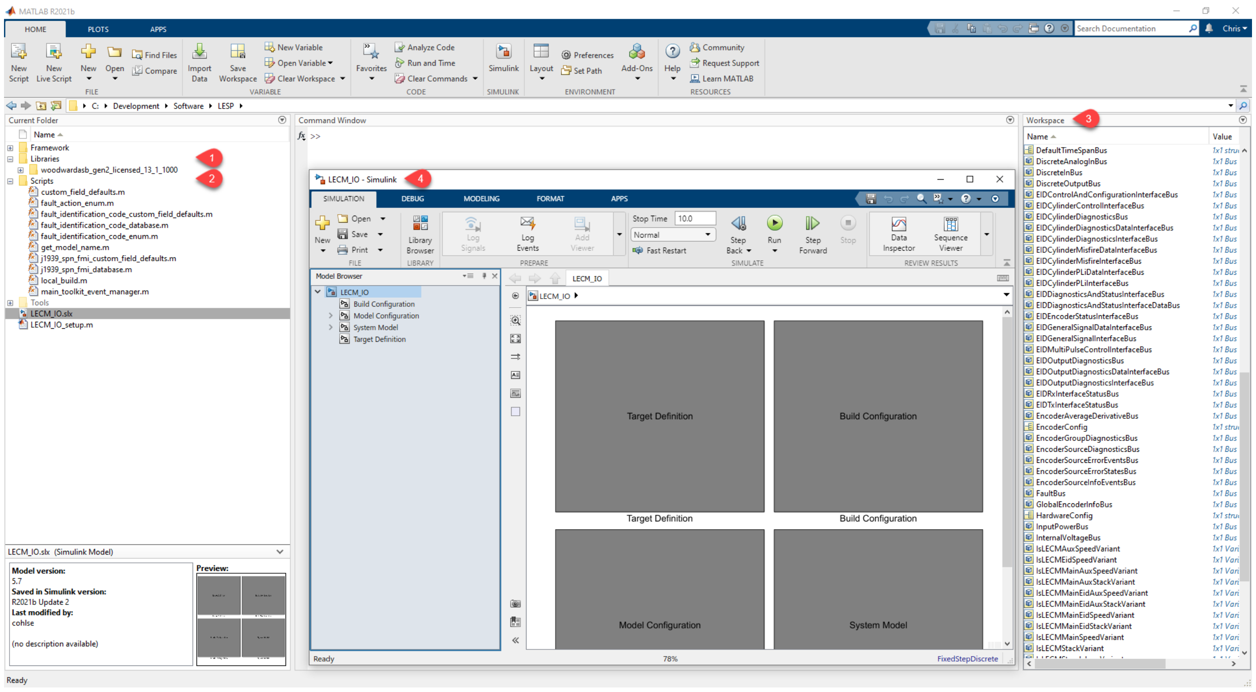
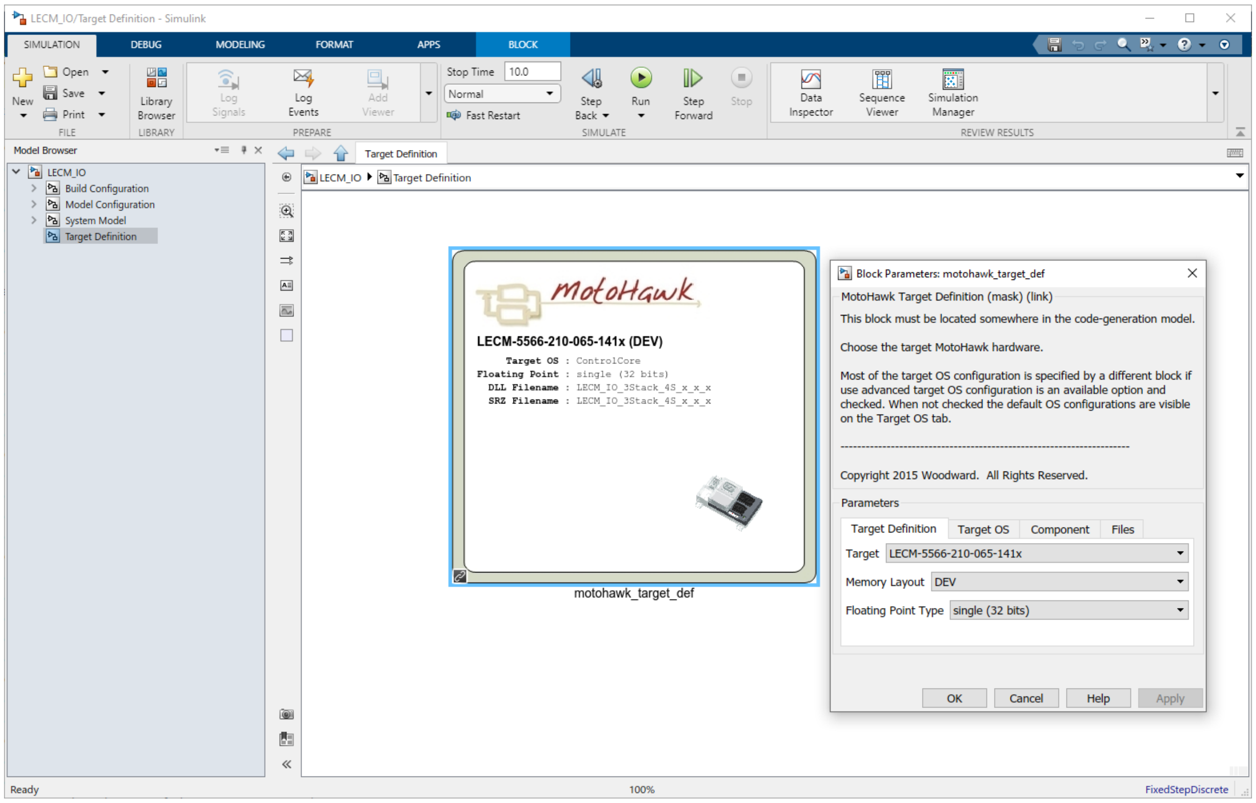


Figure 2-1. Model Root Layout

Target Definition

The target definition subsystem contains the MotoHawk Target Definition block. This is used to control the MotoHawk control module target selection (e.g., LECM Main 3-Stack Variant = LECM-5556-210-065-141x) as well as define the output file names.



LECM-5566-210-065-141x (DEV)

```
      Target OS : ControlCore
 Floating Point : single (32 bits)
   DLL Filename : LECM_IO_3Stack_4S_x_x_x
   SRZ Filename : LECM_IO_3Stack_4S_x_x_x
```

Block Parameters: motohawk_target_def

MotoHawk Target Definition (mask) (link)

This block must be located somewhere in the code-generation model.

Choose the target MotoHawk hardware.

Most of the target OS configuration is specified by a different block if use advanced target OS configuration is an available option and checked. When not checked the default OS configurations are visible on the Target OS tab.

Copyright 2015 Woodward. All Rights Reserved.

Parameters

Target Definition | Target OS | Component | Files

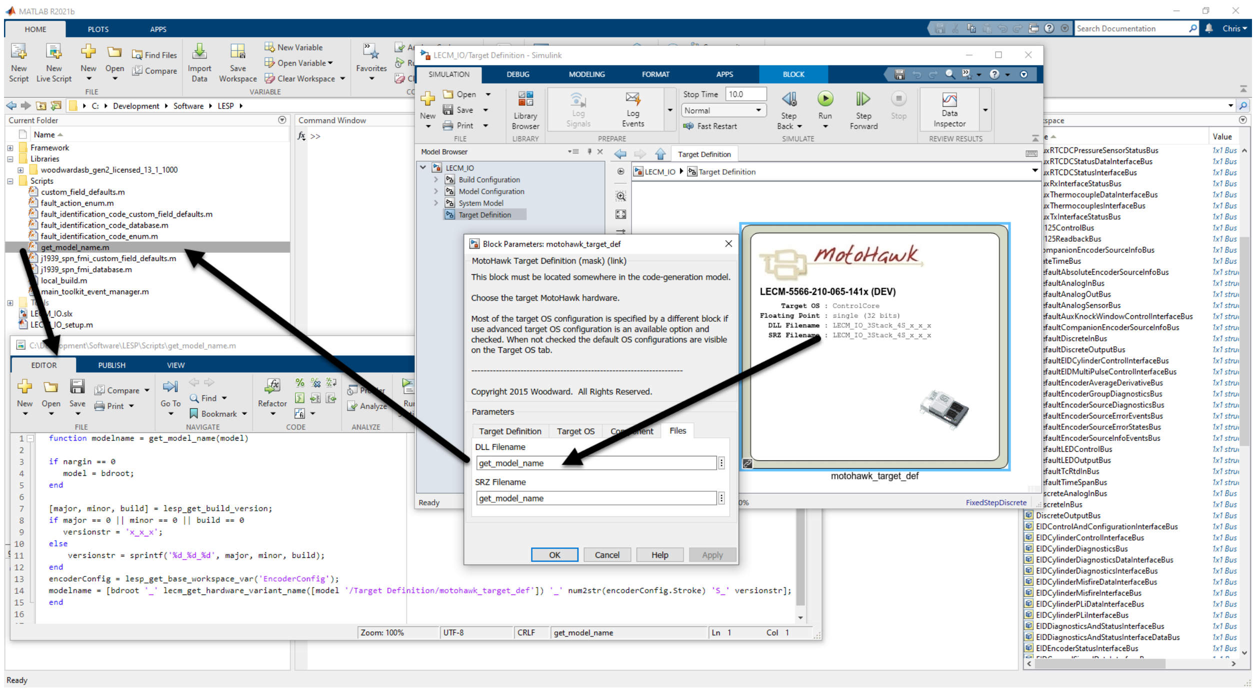
Target: LECM-5566-210-065-141x

Memory Layout: DEV

Floating Point Type: single (32 bits)

OK | Cancel | Help | Apply

Figure 2-2. Target Definition Selection



```
function modelname = get_model_name(model)

if nargin == 0
   model = bdroot;
end

[major, minor, build] = lesp_get_build_version;
if major == 0 || minor == 0 || build == 0
   versionstr = 'x_x_x';
else
    versionstr = sprintf('%d_%d_%d', major, minor, build);
end
encoderConfig = lesp_get_base_workspace_var('EncoderConfig');
modelname = [bdroot '_' lecm_get_hardware_variant_name([model '/Target Definition/motohawk_target_def']) '_' num2str(encoderConfig.Stroke) 'S_' versionstr];
end
```

Block Parameters: motohawk_target_def

MotoHawk Target Definition (mask) (link)

This block must be located somewhere in the code-generation model.

Choose the target MotoHawk hardware.

Most of the target OS configuration is specified by a different block if use advanced target OS configuration is an available option and checked. When not checked the default OS configurations are visible on the Target OS tab.

Copyright 2015 Woodward. All Rights Reserved.

Parameters

Target Definition | Target OS | Component | Files

DLL Filename: get_model_name

SRZ Filename: get_model_name

OK | Cancel | Help | Apply

MotoHawk

LECM-5566-210-065-141x (DEV)

```
      Target OS : ControlCore
 Floating Point : single (32 bits)
   DLL Filename : LECM_IO_3Stack_4S_x_x_x
   SRZ Filename : LECM_IO_3Stack_4S_x_x_x
```

motohawk_target_def

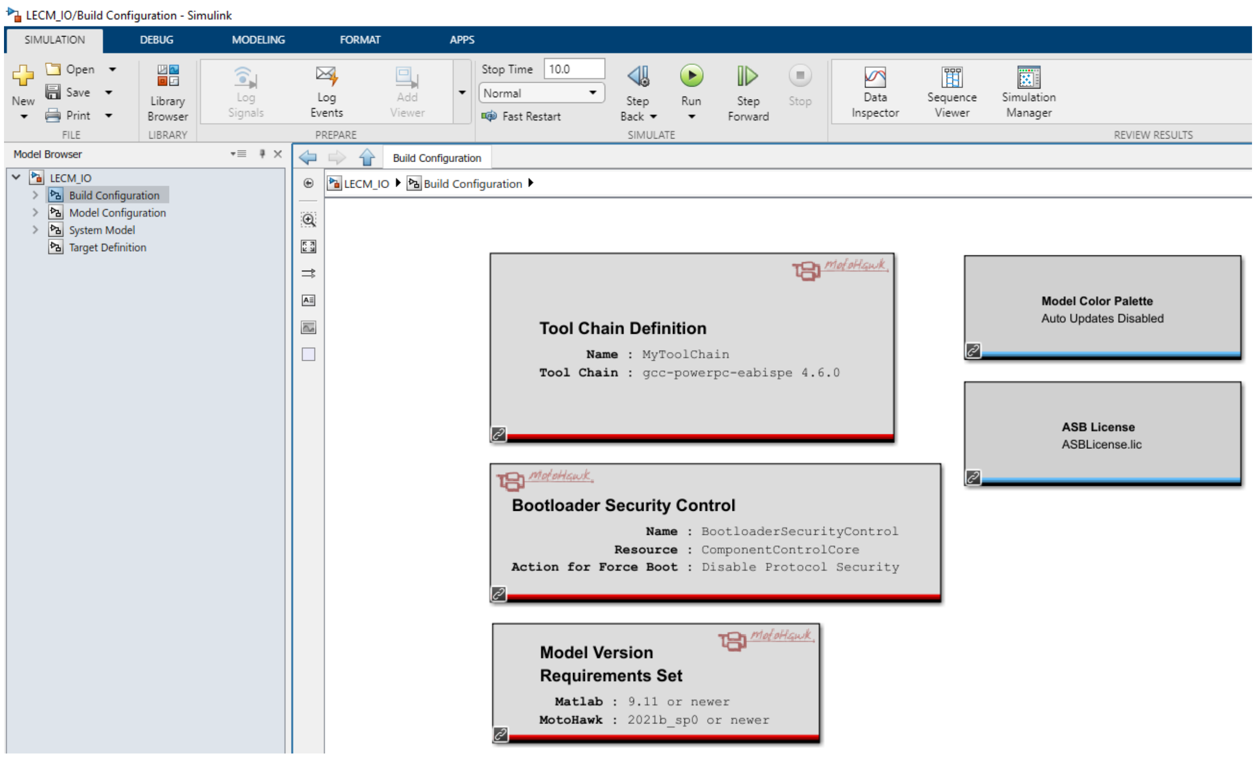
Figure 2-3. Target Definition Name

In the default get_model_name script, the output file naming convention takes on the form <ModelName>_<LECMStackVariant>_<EncoderStroke>_<Version>. The <ModelName> token is set to the filename of the model. The <LECMStackVariant> token uses a friendly LECM variant name of 3Stack, 2Stack, or Standalone based on the current hardware configuration. The <EncoderStroke> token is represented as 2S or 4S respectively. Finally, the version string uses an x_x_x format for Major_Minor_Build based on the values set by the BuildConfig workspace variable. This script is local to the project and can be modified to use any naming or versioning scheme desired by the developer.

Build Configuration

The Build Configuration subsystem contains blocks related to the model/build and other non-execution related blocks including the following:

- **ToolChain Definition**
  - This block defines the compiler used to compile the generated C-code for the target.
- **Bootloader Security Control**
  - This block is used to disable bootloader security if desired, allowing bootstrapping without security privileges.
- **Matlab/MotoHawk Version Requirements**
  - This block is used to verify the specified Matlab/MotoHawk version constraints are met.
- **Model Color Palette**
  - This block can be used to auto-theme blocks in the model (disabled by default).
- **ASB License**
  - This block must be defined in the model to validate the ASB license. Open the mask of this block to update your ASB license.



Tool Chain Definition

```
     Name : MyToolChain
Tool Chain : gcc-powerpc-eabispe 4.6.0
```

Bootloader Security Control

```
                 Name : BootloaderSecurityControl
             Resource : ComponentControlCore
Action for Force Boot : Disable Protocol Security
```

Model Version
Requirements Set

```
  Matlab : 9.11 or newer
MotoHawk : 2021b_sp0 or newer
```

Model Color Palette
Auto Updates Disabled

ASB License
ASBLicense.lic

Figure 2-4. LECM_IO Build Configuration Subsystem

Model Configuration

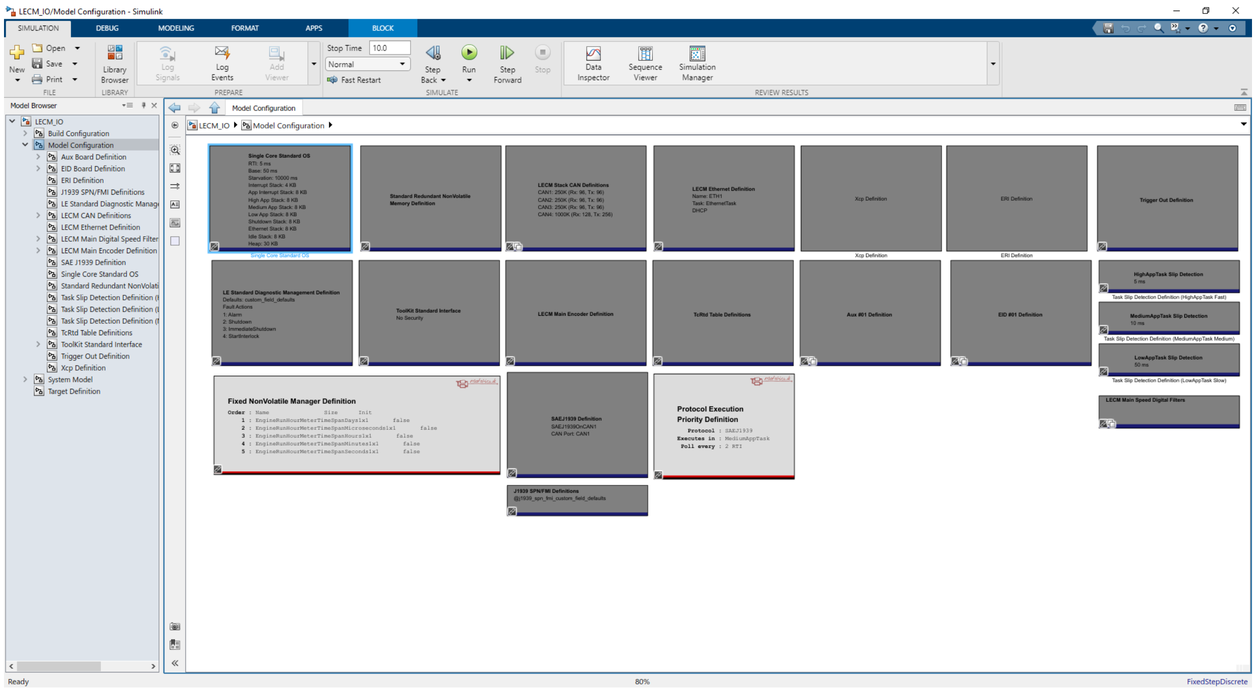
The Model Configuration subsystem defines the core model definition blocks of the OS, hardware, and features. These blocks do not execute within a triggered context. For example, in the LECM_IO package, the following are standard model definitions:

- Standard OS Definition (Tasks, Stacks, Heap, Slip detection definitions)
- Standard NonVolatile Memory + Fixed NonVolatile Memory Definitions
- Standard LECM *variant-based* CAN Definitions
- Standard LECM Ethernet Definition
- Standard XCP/ERI/ToolKit HMI Communication Definitions + Features
- Standard LECM Main Encoder System + Main Speed Digital Filter + Trigger Emulation Definitions
- Standard LESP Diagnostics Management
- Standard LESP Thermocouple/RTD (TcRtd) Conversion Table Definitions
- Standard SAEJ1939 Protocol Definition
- Standard LESP-based Woodward Platform EID and Aux Application Interface Definitions
    - LECM EID Injection, MixedMode, Ignition
    - LECM Aux Thermocouple, Acoustic Knock, RTCDC, Protection IO

See subsequent chapters for more information on each of the above definitions in their respective chapters.

IMPORTANT

Each of the standard LESP-based features/definitions expose a set of default configurations at the mask level that can be adjusted on a per-application basis; however, all blocks are considered “starting points” for the respective feature. If structural changes are required, a local version of the block should be created with the desired custom modifications made using the original block as the “starting point.”



Stop Time 10.0

LECM_IO
- Build Configuration
- Model Configuration
  - Aux Board Definition
  - EID Board Definition
  - ERI Definition
  - J1939 SPN/FMI Definitions
  - LE Standard Diagnostic Manag
  - LECM CAN Definitions
  - LECM Ethernet Definition
  - LECM Main Digital Speed Filter
  - LECM Main Encoder Definition
  - SAE J1939 Definition
  - Single Core Standard OS
  - Standard Redundant NonVolati
  - Task Slip Detection Definition (
  - Task Slip Detection Definition (
  - Task Slip Detection Definition (
  - TcRtd Table Definitions
  - ToolKit Standard Interface
  - Trigger Out Definition
  - Xcp Definition
- System Model
- Target Definition

Single Core Standard OS
RTI: 5 ms
Base: 50 ms
Starvation: 10000 ms
Interrupt Stack: 4 KB
App Interrupt Stack: 8 KB
High App Stack: 8 KB
Medium App Stack: 8 KB
Low App Stack: 8 KB
Shutdown Stack: 8 KB
Ethernet Stack: 8 KB
Idle Stack: 8 KB
Heap: 30 KB
Single Core Standard OS

Standard Redundant NonVolatile Memory Definition

LECM Stack CAN Definitions
CAN1: 250K (Rx: 96, Tx: 96)
CAN2: 250K (Rx: 96, Tx: 96)
CAN3: 250K (Rx: 96, Tx: 96)
CAN4: 1000K (Rx: 128, Tx: 256)

LECM Ethernet Definition
Name: ETH1
Task: EthernetTask
DHCP

Xcp Definition
Xcp Definition

ERI Definition
ERI Definition

Trigger Out Definition

LE Standard Diagnostic Management Definition
Defaults: custom_field_defaults
Fault Actions
1: Alarm
2: Shutdown
3: ImmediateShutdown
4: StartInterlock

ToolKit Standard Interface
No Security

LECM Main Encoder Definition

TcRtd Table Definitions

Aux #01 Definition

EID #01 Definition

HighAppTask Slip Detection
5 ms
Task Slip Detection Definition (HighAppTask Fast)

MediumAppTask Slip Detection
10 ms
Task Slip Detection Definition (MediumAppTask Medium)

LowAppTask Slip Detection
50 ms
Task Slip Detection Definition (LowAppTask Slow)

LECM Main Speed Digital Filters

Fixed NonVolatile Manager Definition
```
Order : Name                               Size      Init
    1 : EngineRunHourMeterTimeSpanDays1x1        false
    2 : EngineRunHourMeterTimeSpanMicroseconds1x1        false
    3 : EngineRunHourMeterTimeSpanHours1x1        false
    4 : EngineRunHourMeterTimeSpanMinutes1x1        false
    5 : EngineRunHourMeterTimeSpanSeconds1x1        false
```

SAEJ1939 Definition
SAEJ1939OnCAN1
CAN Port: CAN1

Protocol Execution Priority Definition
```
   Protocol : SAEJ1939
Executes in : MediumAppTask
 Poll every : 2 RTI
```

J1939 SPN/FMI Definitions
@j1939_spn_fmi_custom_field_defaults

Figure 2-5. LECM_IO Model Configuration Subsystem

System Model

The System Model subsystem contains the core executable logic of the model following a Simulink classic dataflow layout of Inputs → Control → Outputs. In addition, there are the OS Event Management, Diagnostics & Monitoring, and Application containers used to manage other aspects of the model. Inputs can be viewed as "reading" data to be used in downstream Control/Output subsystems. Thus, an optional Feedback subsystem exists for any control-related parameters that also need to be referenced within an "input-related" function. In general, the amount of information to feedback is often non-existent or limited (e.g., fuel demand may be fed back in order to calculate load used in other input calculations). If using Feedback signals, be sure to use 1/z unit delay blocks.

- **OS Event Management**
  - Manager OS event triggers and their order of execution
- **Inputs**
  - Includes containers for hardware reads, IO mappings, input characterizations, functions, calculations, and other application global data.
- **Control**
  - Includes containers for control features (e.g., speed control, pump control, module control, etc.)
- **Outputs**
  - Includes containers output characterization, hardware writes, and datalink
- **Diagnostics & Monitoring**
  - Includes containers for managing diagnostic and monitor-related functions
- **Application**
  - Includes containers for managing global application data and other necessary feature management (e.g., statistics, app performance monitoring, app shutdown management, etc.)

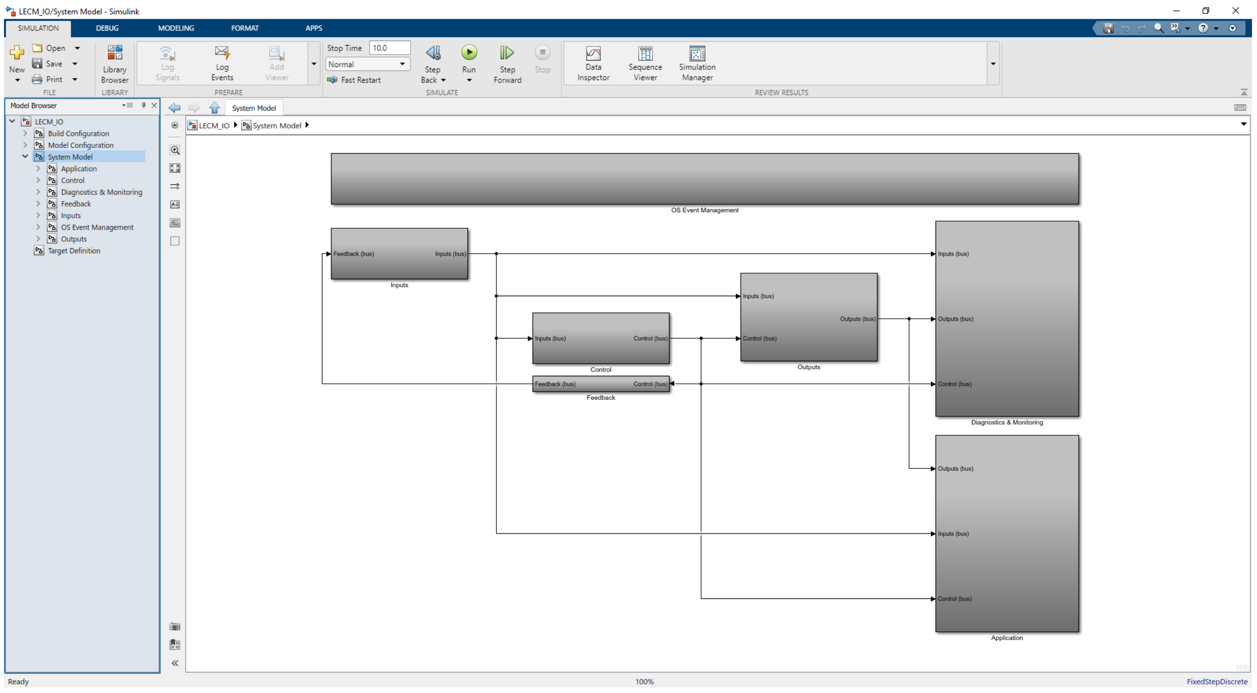


Figure 2-6. LECM_IO System Model Subsystem

OS Event Management

The OS Event Management subsystem contains the OS “events” used to drive application executable logic in a user-prescribed order. This logic can be viewed as two sequential sections. The left section (outlined in red) contains the LESP standard OS Event events for a single core three-tiered tasking structure of “High Priority Fast Rate,” “Medium Priority Medium Rate,” and “Low Priority Slow Rate” in addition to the standard application Startup, Shutdown, and Idle events. These provide the basic application OS events as follows:

- High Priority Fast Rate
  - Executes downstream triggered logic in the High application task at the fastest rate defined by the OSConfig.FastRateSec.
- Medium Priority Medium Rate
  - Executes downstream triggered logic in the Medium application task at the medium rate defined by the OSConfig.MediumRateSec.
- Low Priority Slow Rate
  - Executes downstream triggered logic in the Low application task at the slow rate defined by the OSConfig.SlowRateSec.
- Idle Event
  - Executes when the application idle event is entered
- Startup Event
  - Executes once upon application startup
- Shutdown Event
  - Executes just prior to the application shutting down

The subsequent logical section (outlined in blue) takes the standard OS event triggers and splits them into multiple application-referenced triggers in a user-defined order of execution using Simulink function-call split blocks. This structure provides a single location of OS event triggers and order of execution control. It is possible to extended OS events or modify the structure by using MotoHawk Trigger blocks directly instead of the standard events depending on the application needs; however, the structure is designed to be flexible.

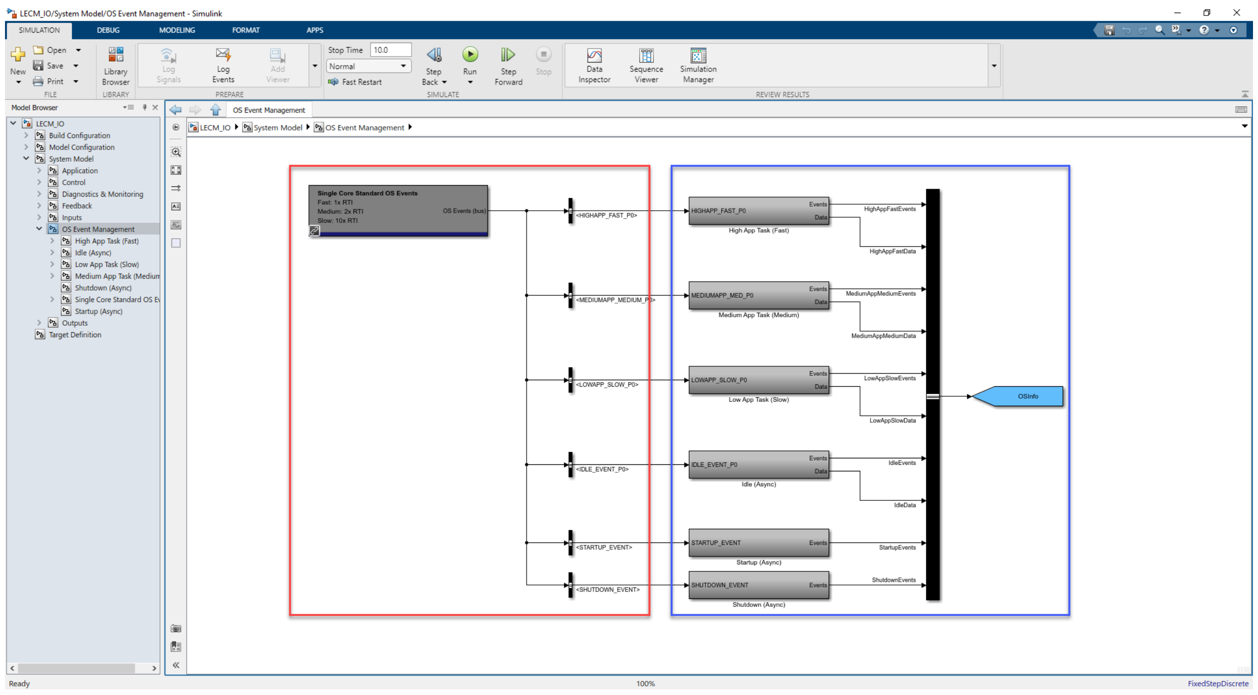


Figure 2-7. OS Event Management Layout

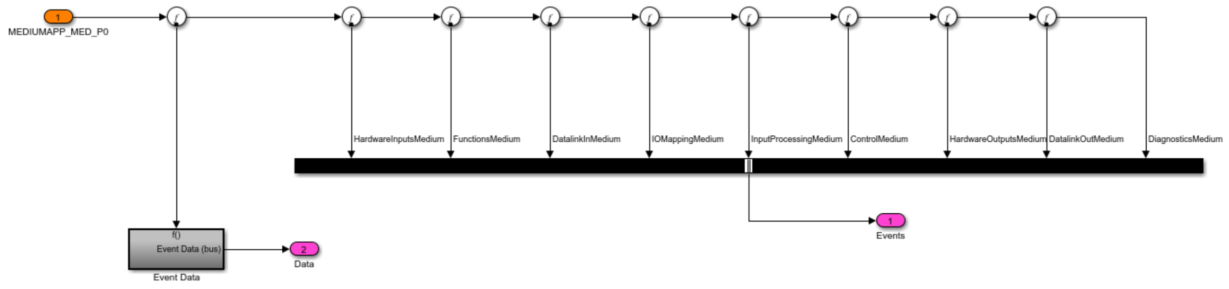


Figure 2-8. Medium Priority Medium Rate Events Example

In the above example, the Medium task/Medium rate OS event is split into multiple events that execute in a left to right order (*HardwareInputsMedium* triggered subsystem executes, then *FunctionsMedium* subsystem executes, etc.).

Inputs

The Inputs subsystem is structured by default with the following layout:

1. **Read Hardware**
    a. This subsystem contains triggered subsystems used to hardwired inputs of the module (e.g., Analog Inputs, Discrete Inputs, etc.)
2. **Application**
    a. This subsystem is used to read global application-level data (e.g., fault event requests, ToolKit HMI requests, Engine global parameters, etc.)
3. **IO Mapping**
    a. This subsystem is used to support *dynamic* mapping of hardware inputs to sensors (e.g., AIN1 maps to Ambient Air Pressure sensor). If static mapping is used, this subsystem can be removed and reduce processing overhead.
4. **Input Processing**

   a. This subsystem is used to convert hardware units to sensor engineering units (e.g., Ambient Air Pressure = AIN1 [V] * Gain [kPa/V] + Offset [kPa] + Filtering + Defaulting, Sensor Diagnostic Range Checks, etc.). This system is also used for Datalink input conversion.

5. **Functions**
   a. This subsystem is used for functions, calculations, and other “virtual” sensing types of inputs (e.g., Engine State, Fault Actions, Fuel Density calculation, etc.)

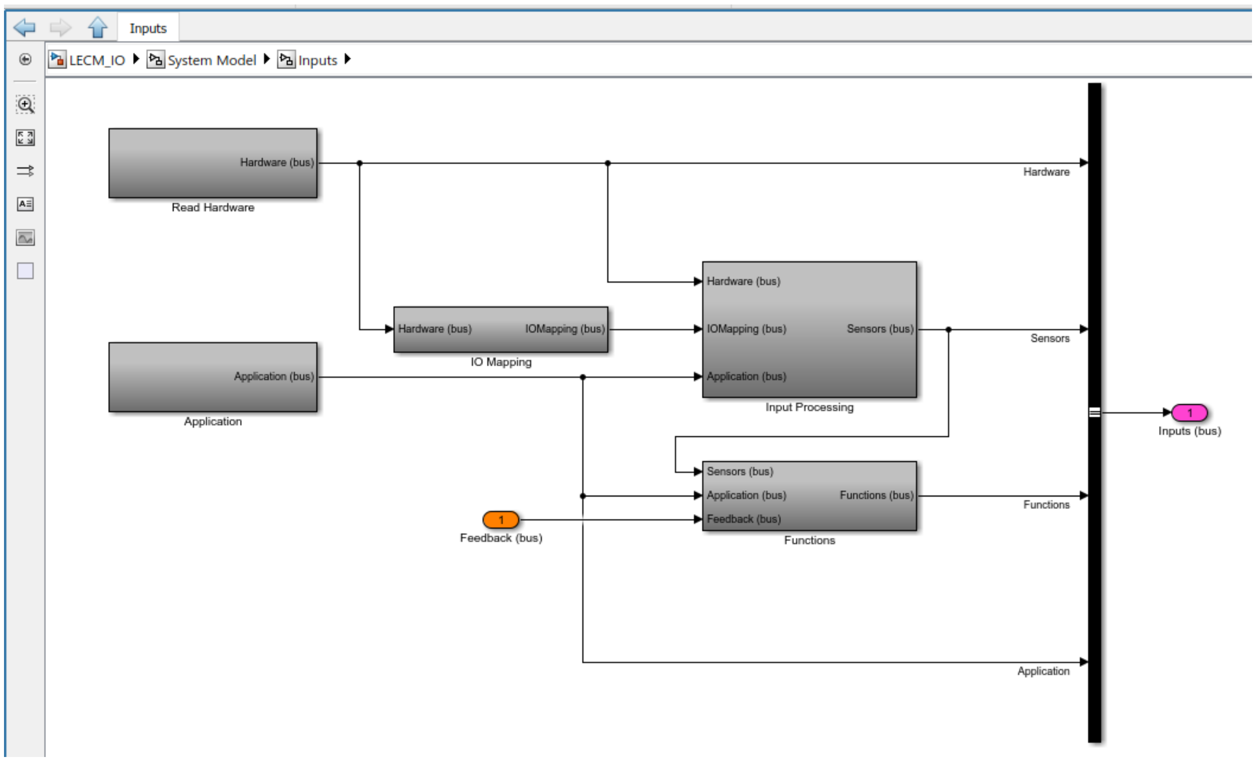


Figure 2-9. Input Subsystem Default Layout

All the inputs are placed onto grouped virtual bus structures and then bussed together for global access in an “Inputs” bus that is referenced in downstream subsystems.

Control

The control subsystem contains the three different contexts for control-related logic: Fast, Medium, and Slow. Each subsystem is triggered from the OS Events as previously described. High priority and fast executing controllers/control logic should be located in the Control (Fast) function-call subsystem, whereas lower priority controllers/control logic should be located in the Medium or Slow triggered subsystems respectively. For example, engine speed and fueling controls may be located in the high priority fast execution context; whereas engine start logic may be located in a lower priority/slower subsystem in order to reduce processing utilization.

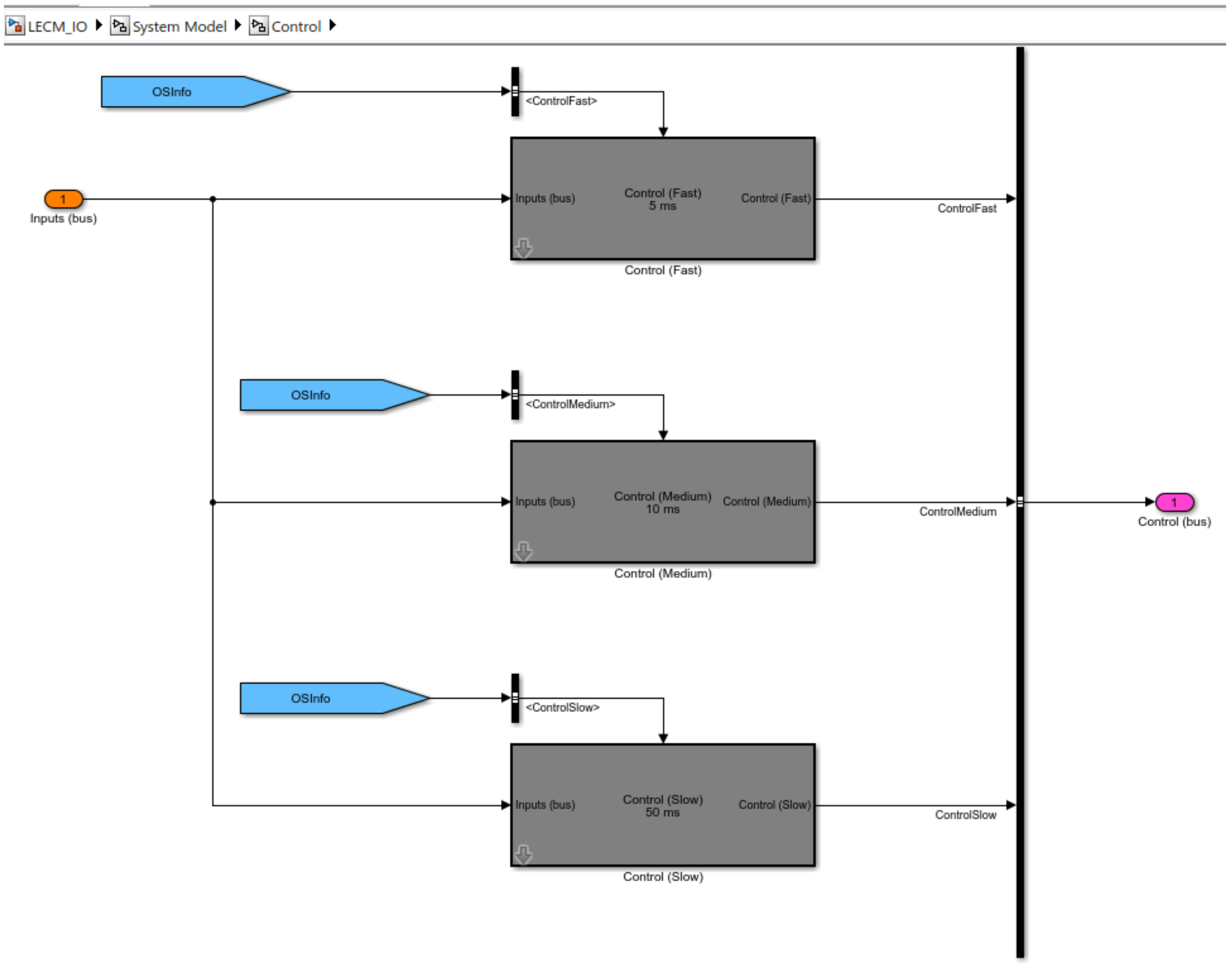


Figure 2-10. Control Subsystem

Outputs Subsystem

The outputs subsystem is used to convert control requests to hardwired commands (e.g., activating an LSO output) and to send/update datalink messages. Each of these core functions is implemented in the standard Fast, Medium, and Slow execution contexts. By default, the applications come structured with a dynamic output mapping structure. If dynamic mapping is not required, this logic can be removed, and static mapping can be used to reduce processing overhead.

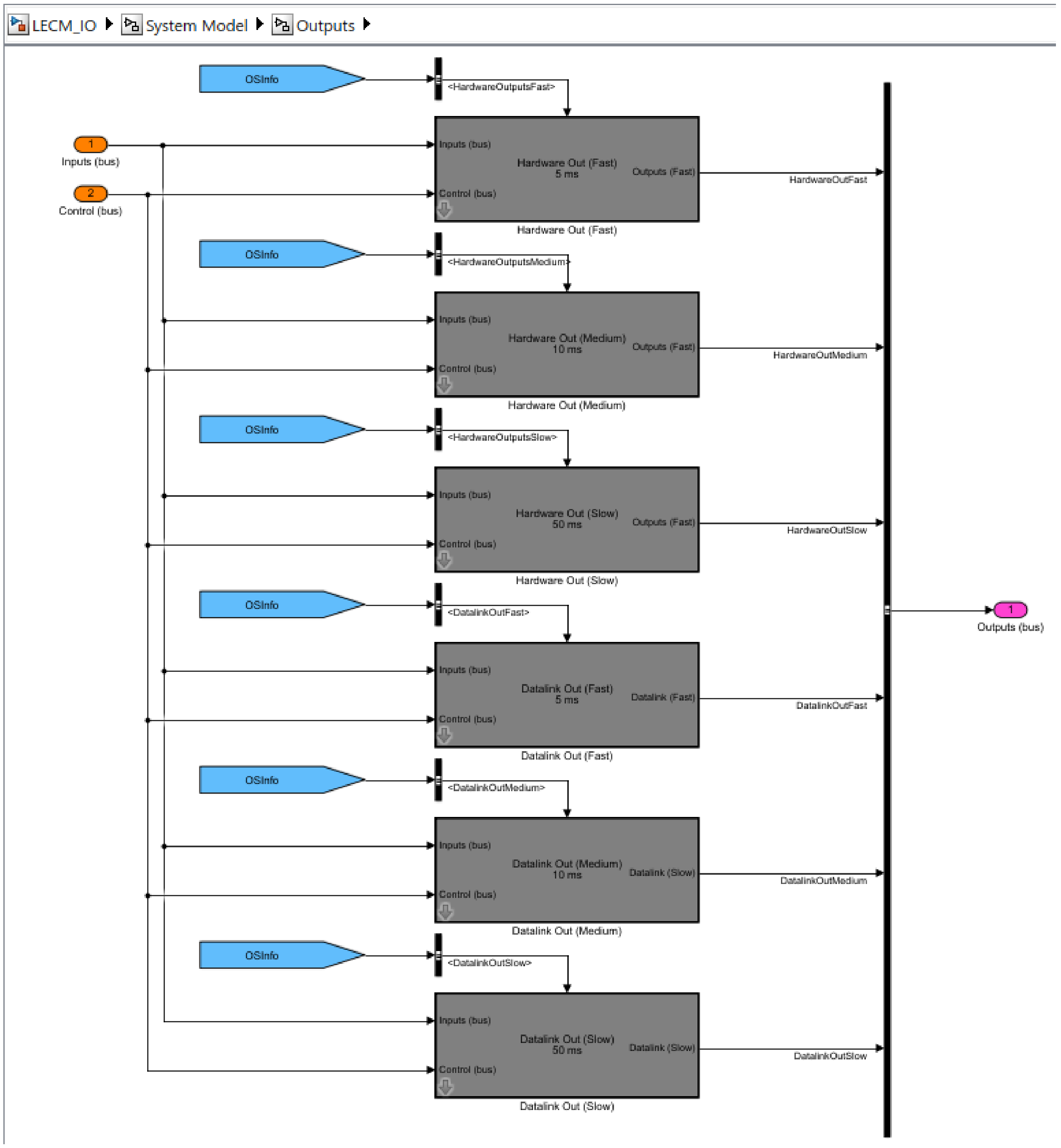


Figure 2-11. Outputs Subsystem

Dynamic Discrete Output Mapping

The dynamic discrete output mappings platform blocks accept an array of “DiscreteOutputBus” types and properly map the commands to the respective hardware output. Example: LECM-based discrete output mapping:

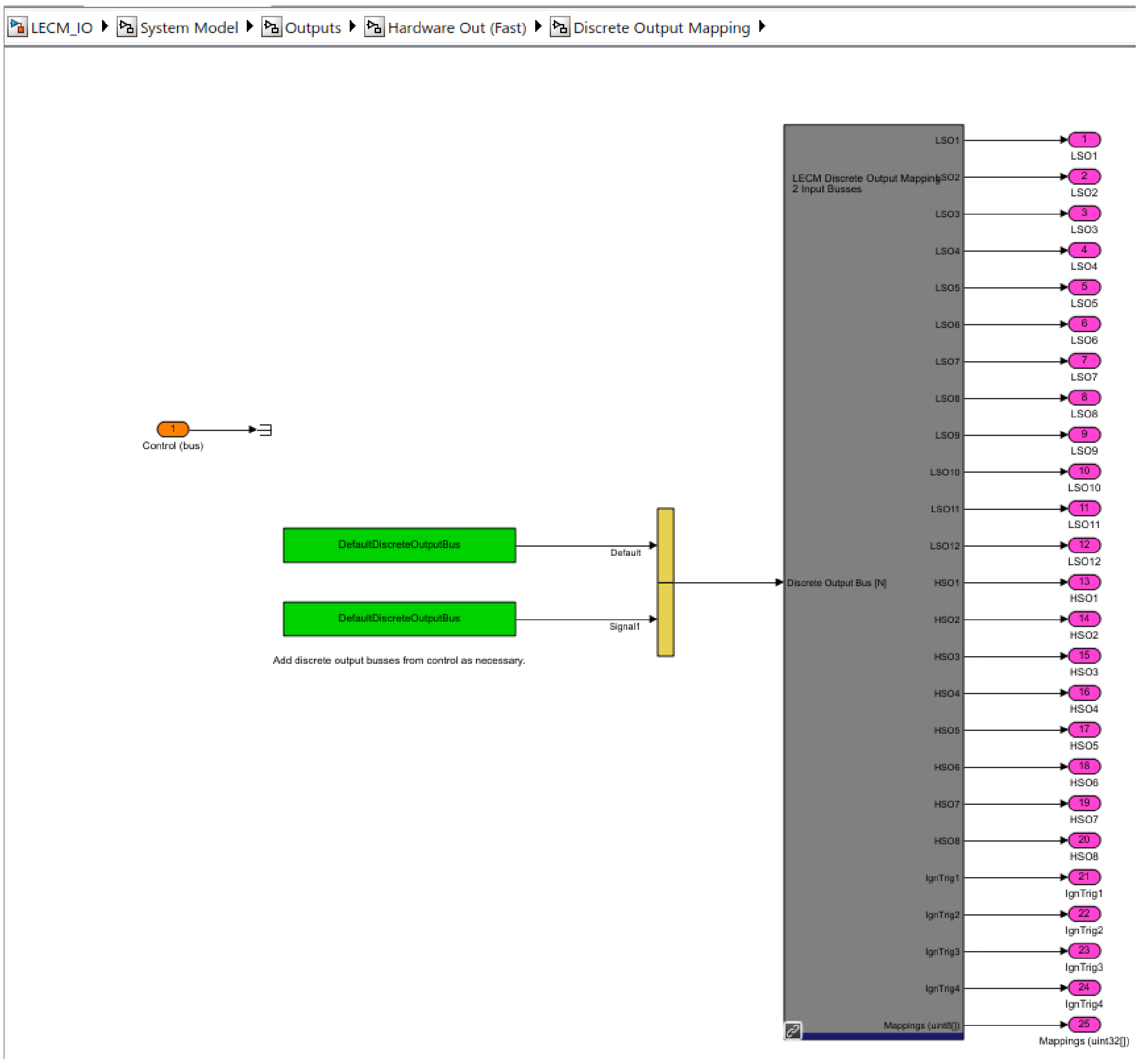


Figure 2-12. LECM Discrete Output Mapping Example

Dynamic Analog Output Mapping

The dynamic analog output mappings platform blocks accept an array of “AnalogOutputBus” types and properly map the commands to the respective hardware output. Example: LECM-based analog output mapping:

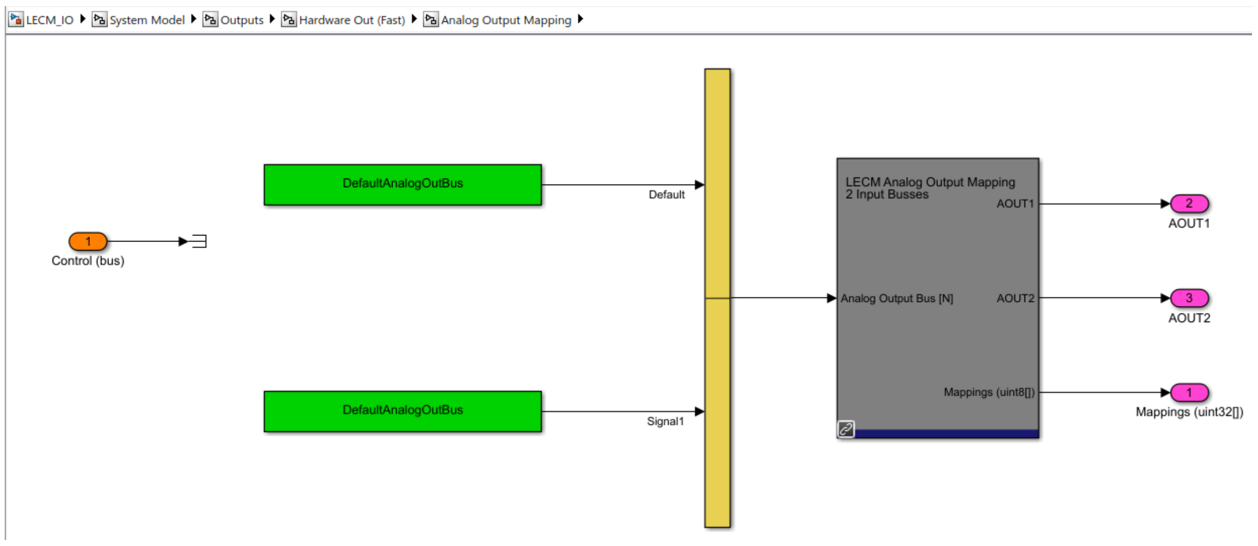


Figure 2-13. LECM Analog Output Mapping Example

Diagnostics & Monitoring Subsystem

The diagnostics and monitoring (D&M) subsystem is used to manage diagnostics and monitors as well as the general D&M activities. LESP standard D&M structure is outlined in a subsequent chapter. Each D&M function should be located is an appropriate execution context.

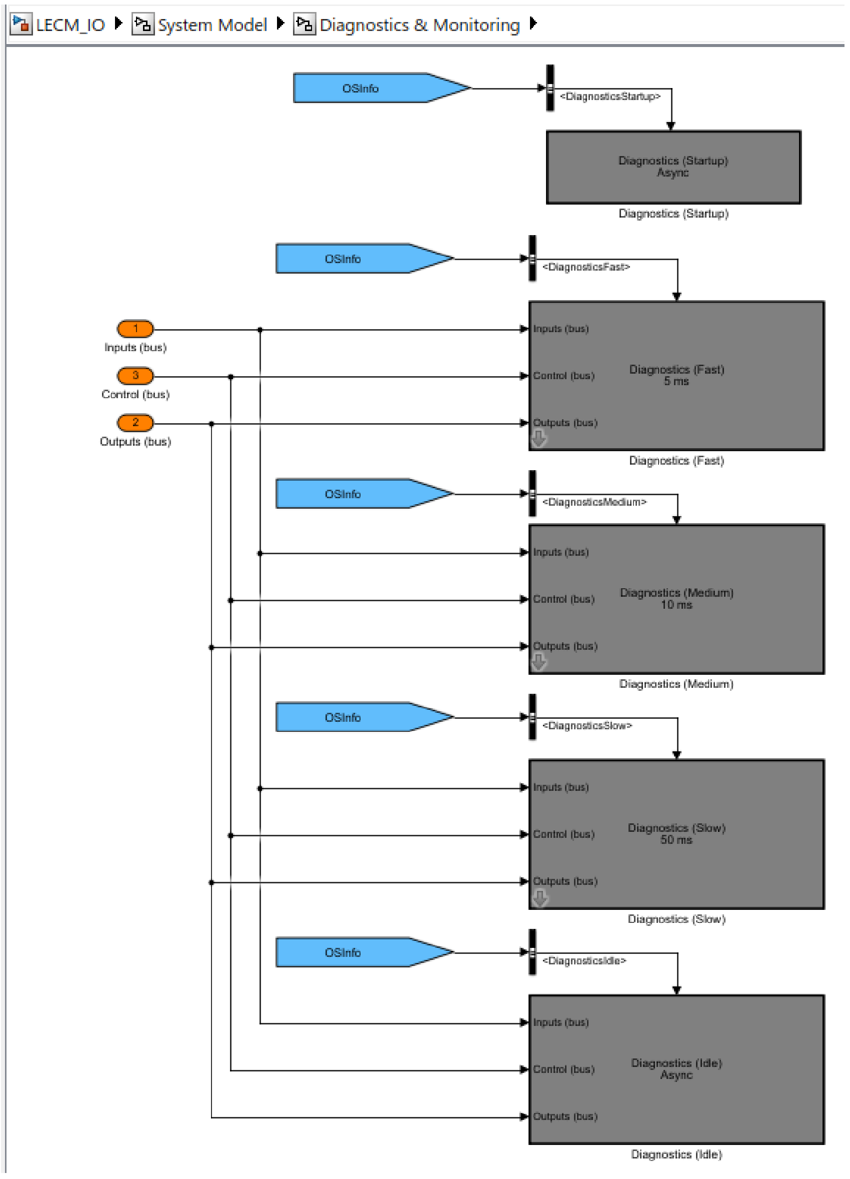


Figure 2-14. Diagnostics & Monitoring Subsystem

Application Subsystem

The application subsystem is used to manage application and background tasks. This may include logic to execute on application startup and shutdown events, in addition to background tasks such as shutdown management, real-time clock management, statistics, etc.

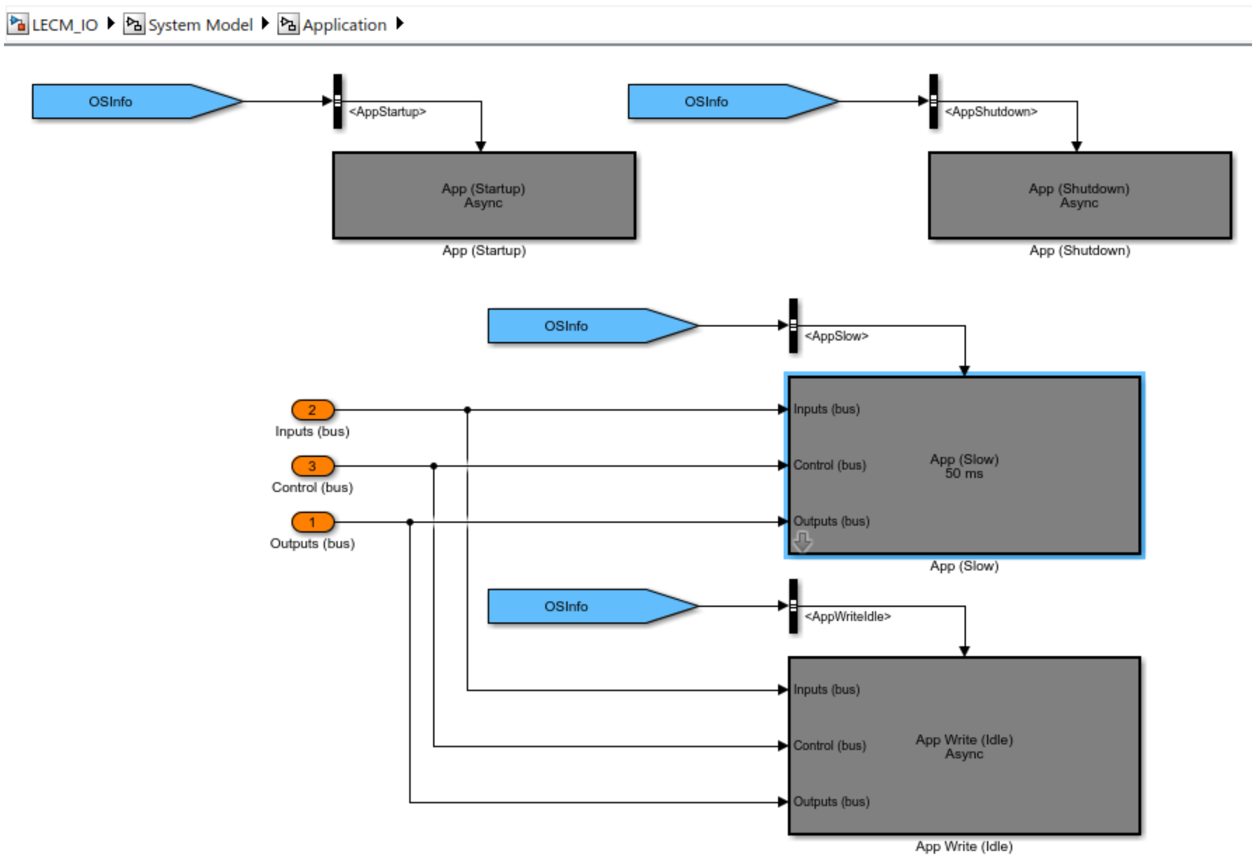


Figure 2-15. Application Subsystem

Chapter 3. EID Interfacing

Introduction

The following chapter describes interfacing with Woodward standard platform EID applications using the prebuilt LESP framework interface blocks. Before getting started, it’s important to understand the Woodward standard EID applications variants.

EID MixedMode

The MixedMode software is designed as an LECM Eid Ignition + Injection driver application with up to 20-cylinder support. The application contains three control options: local, shared memory region control interfaces, and external communication interfaces (J1939). The local control enables the application to act as a standalone board where configuration and controls are local to the board (usually for test purposes). The shared memory region controls options expose multiple control interfaces using ASB shared memory regions that allow for a plug and play (PnP) control blocks in supporting libraries that can be quickly added to LECM main board applications. The external communication interfaces support standard external communications (e.g., J1939, ...) for EID standalone and 3rd party controls. The software architecture is a componentization design with "Plug-N-Play" subsystems to support multiple variations of the application with the control logic being developed in core re-usable components. The scope of this application and the overall high-level architecture is designed to operate the following LE engine contexts:

- Ignition Only (up to 20 cylinders)
- Injection Only (up to 20 cylinders)
- Ignition + Injection (up to 20 cylinders distributed between output types, e.g., 10+10, 8+12, 9+9, ...)
- Injection TypeA + Injection TypeB (up to 20 cylinders of two different injection types distributed between the output types, e.g., 10 SOGAV + 10 Micropilot injectors)
- Injection TypeA + Injection TypeB + Ignition (up to 6 cylinders of two different injection types and ignition in a single EID board)

EID Ignition

The Ignition software is designed as an LECM Eid Ignition only driver application with up to 20-cylinder support. The application contains three control options: local, shared memory region control interfaces, and external communication interfaces (J1939 Standard + IC920 Proprietary). The local control enables the application to act as a standalone board where configuration and controls are local to the board (usually for test purposes). The shared memory region control options expose several control interfaces using ASB shared memory regions that allow for a plug and play (PnP) control blocks in supporting libraries that can be quickly added to LECM main board applications. The external communication interfaces support standard external communications (J1939, IC92x, ...) for EID standalone and 3rd party controls. The software architecture is a componentization design with "Plug-N-Play" subsystems to support multiple variations of the application with the control logic being developed in core re-usable components. The scope of this application and the overall high-level architecture is designed to operate the following LE engine contexts:

- Ignition Only (up to 20 cylinders)
- IC92x Replacement (up to 20 cylinders)

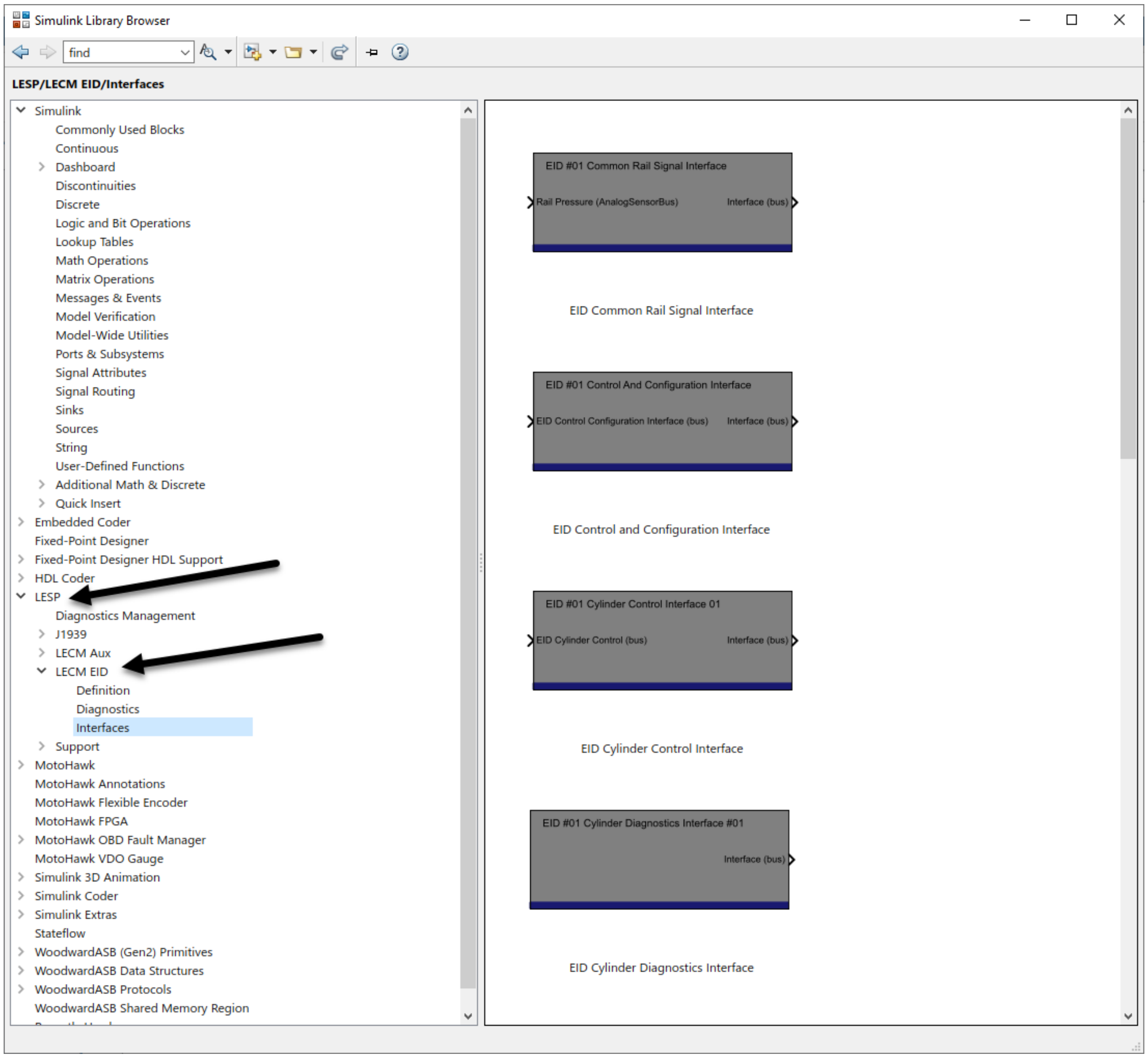
EID Injection

The EidInjection software is designed as an LECM Eid Common Rail + Dual Fuel Injection driver application with up to 20 cylinder / 5 event support. The application contains two control options: local and shared memory region control interfaces. The local control enables the application to act as a standalone board where configuration and controls are local to the board (usually for test purposes). The shared memory region controls options expose multiple control interfaces using ASB shared memory regions that allow for a plug and play (PnP) control blocks in supporting libraries that can be quickly added to LECM main board applications. The software architecture is a componentization design with "Plug-N-Play" subsystems to support multiple variations of the application with the control logic being developed in core re-usable components. The scope of this application and the overall high-level architecture is designed to operate the following LE engine contexts:

- Common Rail Injection, 1-5 events/cylinder/cycle (up to 20 cylinders)
- Dual Fuel Injection, 1-5 events/cylinder/cycle (up to 20 cylinders)

LESP EID Interface Blocks

The EID Interface blocks can be located in the Simulink Library Browser under the LESP folder.



LESP/LECM EID/Interfaces

- Simulink
  - Commonly Used Blocks
  - Continuous
  - Dashboard
  - Discontinuities
  - Discrete
  - Logic and Bit Operations
  - Lookup Tables
  - Math Operations
  - Matrix Operations
  - Messages & Events
  - Model Verification
  - Model-Wide Utilities
  - Ports & Subsystems
  - Signal Attributes
  - Signal Routing
  - Sinks
  - Sources
  - String
  - User-Defined Functions
  - Additional Math & Discrete
  - Quick Insert
- Embedded Coder
- Fixed-Point Designer
- Fixed-Point Designer HDL Support
- HDL Coder
- LESP
  - Diagnostics Management
  - J1939
  - LECM Aux
  - LECM EID
    - Definition
    - Diagnostics
    - Interfaces
  - Support
- MotoHawk
- MotoHawk Annotations
- MotoHawk Flexible Encoder
- MotoHawk FPGA
- MotoHawk OBD Fault Manager
- MotoHawk VDO Gauge
- Simulink 3D Animation
- Simulink Coder
- Simulink Extras
- Stateflow
- WoodwardASB (Gen2) Primitives
- WoodwardASB Data Structures
- WoodwardASB Protocols
- WoodwardASB Shared Memory Region

EID #01 Common Rail Signal Interface
Rail Pressure (AnalogSensorBus) Interface (bus)
EID Common Rail Signal Interface

EID #01 Control And Configuration Interface
EID Control Configuration Interface (bus) Interface (bus)
EID Control and Configuration Interface

EID #01 Cylinder Control Interface 01
EID Cylinder Control (bus) Interface (bus)
EID Cylinder Control Interface

EID #01 Cylinder Diagnostics Interface #01
Interface (bus)
EID Cylinder Diagnostics Interface

Figure 3-1. LECM EID Interface Blocks

IMPORTANT **For more details on EID interfaces and descriptions, please refer to the respective application manuals:**

MixedMode: B35014
Ignition: B35100
Injection: B35175

Definition

As previously described in Chapter 2, under the Model Configuration subsystem, EID interface definitions should be defined. If interfacing with more than one EID, enter a new definition with a unique EID number instance. Note, the definition is a variant subsystem to support compilation on a Main Standalone vs a Main Stacked unit.

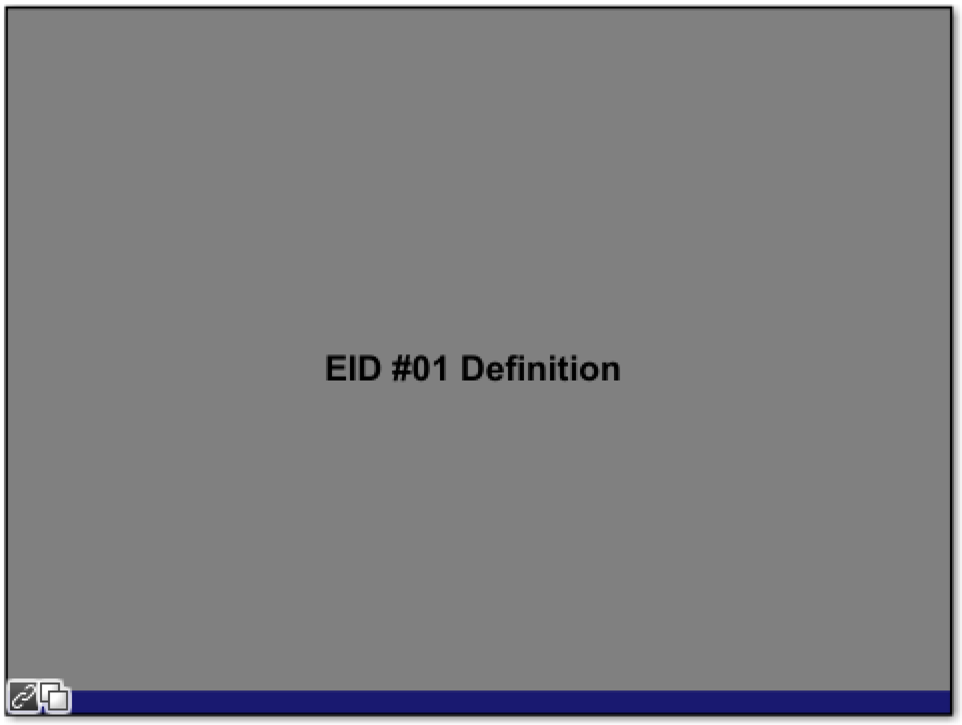


Figure 3-2. EID Interface Definition

The block variant is controlled by the IsLECMStandaloneVariant workspace parameter, which is defined as follows:

IsLECMStandaloneVariant = strcmp(HardwareConfig.Target, 'LECM-5566-210-065-142x')

General EID Interfaces

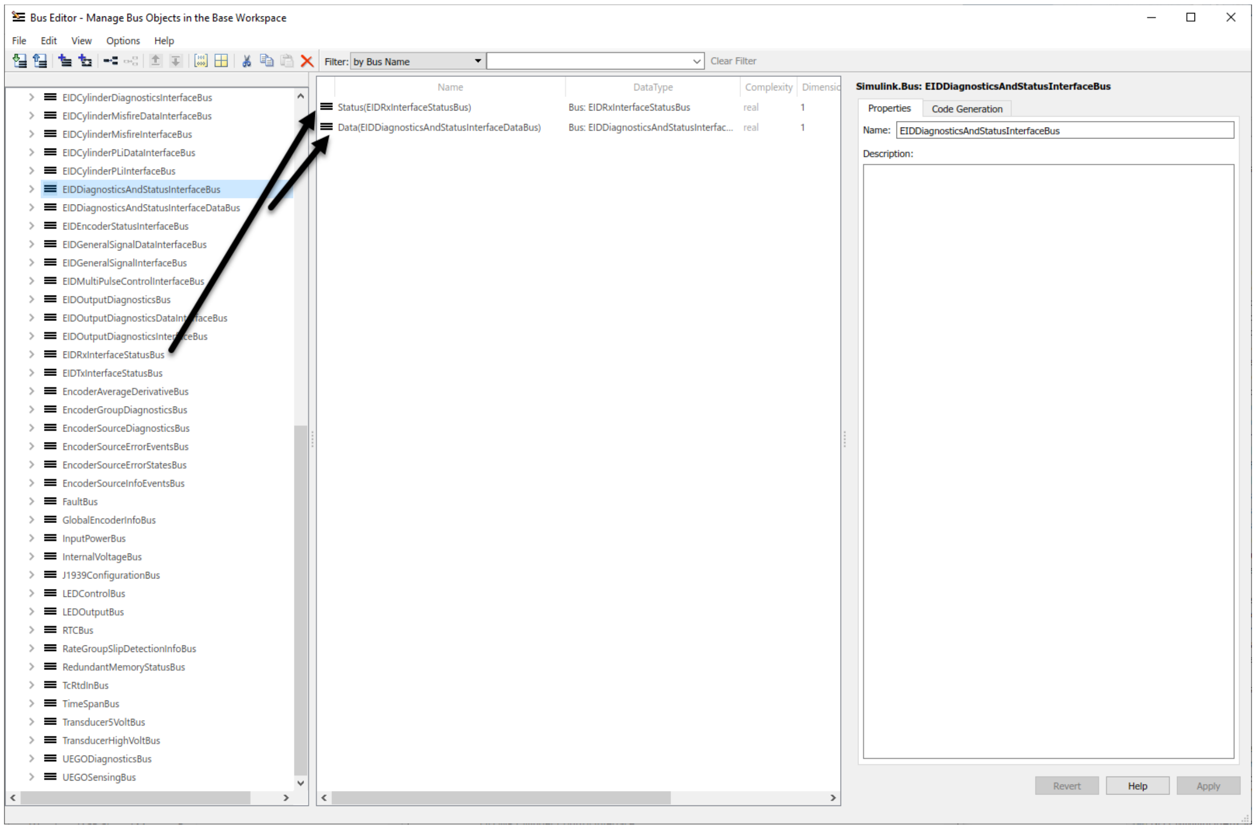
The following section describes common EID interface blocks supported for all EID application variants. All EID → Main feedback/reporting interfaces have a two-layer bus interface as follows:

- Interface (bus)
  - EIDRxInterfaceStatusBus
    - IsEnabled (bool)
    - Timeout_ms (uint32)
    - RegionAge_ms (uint32)
    - ConfigurationFault (bool)

        - TimeoutFault (bool)
        - DatalinkSelect_enum (uin8)
    - *DataBus
        - Data is interface dependent

All Main → EID command interfaces report the following status bus that can be used to drive diagnostics if the interface fails to transmit:

- Interface (bus)
    - EIDTxInterfaceStatusBus
        - IsConnected (bool)
        - TxError (bool)
        - LoopTime_ms (uint32)
        - SourceAddress (uint8)



EIDCylinderDiagnosticsInterfaceBus
EIDCylinderMisfireDataInterfaceBus
EIDCylinderMisfireInterfaceBus
EIDCylinderPLiDataInterfaceBus
EIDCylinderPLiInterfaceBus
EIDDiagnosticsAndStatusInterfaceBus
EIDDiagnosticsAndStatusInterfaceDataBus
EIDEncoderStatusInterfaceBus
EIDGeneralSignalDataInterfaceBus
EIDGeneralSignalInterfaceBus
EIDMultiPulseControlInterfaceBus
EIDOutputDiagnosticsBus
EIDOutputDiagnosticsDataInterfaceBus
EIDOutputDiagnosticsInterfaceBus
EIDRxInterfaceStatusBus
EIDTxInterfaceStatusBus
EncoderAverageDerivativeBus
EncoderGroupDiagnosticsBus
EncoderSourceDiagnosticsBus
EncoderSourceErrorEventsBus
EncoderSourceErrorStatesBus
EncoderSourceInfoEventsBus
FaultBus
GlobalEncoderInfoBus
InputPowerBus
InternalVoltageBus
J1939ConfigurationBus
LEDControlBus
LEDOutputBus
RTCBus
RateGroupSlipDetectionInfoBus
RedundantMemoryStatusBus
TcRtdInBus
TimeSpanBus
Transducer5VoltBus
TransducerHighVoltBus
UEGODiagnosticsBus
UEGOSensingBus

Name	DataType	Complexity	Dimensic
Status(EIDRxInterfaceStatusBus)	Bus: EIDRxInterfaceStatusBus	real	1
Data(EIDDiagnosticsAndStatusInterfaceDataBus)	Bus: EIDDiagnosticsAndStatusInterfac...	real	1

Simulink.Bus: EIDDiagnosticsAndStatusInterfaceBus

Name: EIDDiagnosticsAndStatusInterfaceBus

Description:

Figure 3-3. EID Main Feedback/Reporting Interface Example

Diagnostics & Status Interface

The diagnostics and status interface reports general EID diagnostic and status information to the main board. It's the responsibility of the application developer to integrate this information into the application as deemed appropriate (see diagnostics example in subsequent section below).

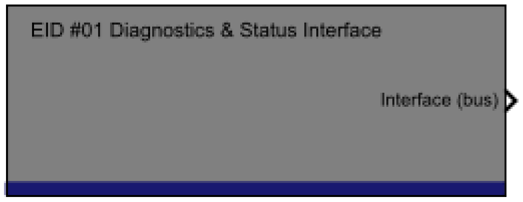


Figure 3-4. EID Diagnostics & Status Interface

The data definition for this interface includes the following:

Table 3-1. EID Diagnostics & Status Interface Definition

Parameter	DataType	Description
ShutdownPendingStatus	Boolean	True if EID will soon shutdown
ShutdownStatus	Boolean	True if EID is in a shutdown state
FPGAFlashRetentionStatus	Boolean	True if EID should be reprogrammed
DriverEnableStatus	Boolean	True if the EID HW driver enable is active
IOLockStatus	Boolean	True if IOLock status is active on the EID
IOLockRequest	Boolean	True if EID is actively requesting IOLock
BackupSettingsInUse	Boolean	True if backup settings are active
UnconfiguredSettingsInUse	Boolean	True if unit has been freshly programmed and settings have not been saved
UnexpectedSystemResetFault_enum	UInt8	See lecm_eid_fault_status_enum
BoostSupplyFault_enum	UInt8	
MemoryFault_enum	UInt8	
OutputFault_enum	UInt8	
OverspeedFault_enum	UInt8	
InternalVoltageFault_enum	UInt8	
HVTransducerFault_enum	UInt8	
TempSensorFault_enum	UInt8	
InvalidConfigFault_enum	UInt8	
SupplyVoltageLowFault_enum	UInt8	
SupplyVoltageHighFault_enum	UInt8	
OperationalTempHighFault_enum	UInt8	
CriticalTempHighFault_enum	UInt8	
InternalError_enum	UInt8	
RemoteInterfaceFault_enum	UInt8	
CANPortFault_enum	UInt8	
ActiveEncoderSource_enum	UInt8	See lecm_eid_active_enc_src_enum
PrimaryCrankStatus	EIDEncoderStatusBus	SourceState_enum (uint8) RelativeLossError (bool) ToothCountMismatch (bool) SignalError (bool) LossError (bool) SyncError (bool) ConfigurationError (bool) See lecm_eid_encoder_src_state_enum
PrimaryCamStatus	EIDEncoderStatusBus	
PrimarySyncStatus	EIDEncoderStatusBus	
AlternateCrankStatus	EIDEncoderStatusBus	
AlternateCamStatus	EIDEncoderStatusBus	
AlternateSyncStatus	EIDEncoderStatusBus	
SystemState_enum	UInt8	See lecm_eid_system_state_enum
AverageSpeed_RPM	UInt16	Average speed as observed by the EID

SoftwareVersionNumber	UInt16	Major software version of the EID firmware

Output Diagnostics Interface

The output diagnostics interface reports summarized EID output driver error conditions for each output #1 - #20.

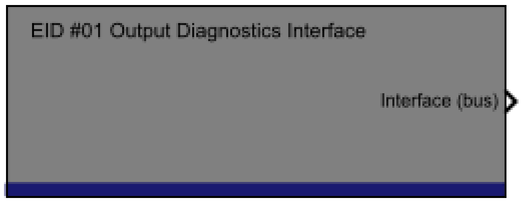


Figure 3-5. EID Output Diagnostics Interface

The data definition for this interface includes the following:

Table 3-2. EID Output Diagnostics Interface Definition

Parameter	DataType	Description
Output01...Output20	EIDOutputDiagnosticsBus	ConfigurationError (bool) BoostVoltageError (bool) OpenError (bool) ShortError (bool) DetrimentalError (bool)

Control & Configuration Interface

The control and configuration interface is used as the global control interface for the EID firmware and supported features. The block accepts an EIDControlAndConfigurationInterfaceBus input type with the following parameters:

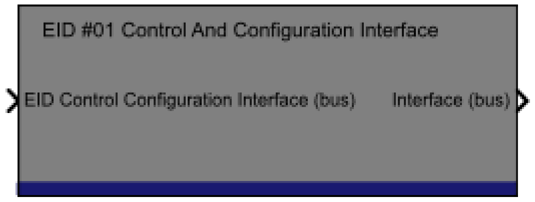

EID #01 Control And Configuration Interface
EID Control Configuration Interface (bus)
Interface (bus)

Figure 3-6. EID Control & Configuration Interface

Table 3-3. EID Control & Configuration Interface Definition

Parameter	DataType	Description
OutputDriverEnable	Boolean	Set to true to enable all output drivers if configured for remote control
FaultReset	Boolean	Toggle from false to true to request Active faults on the EID to clear
EncoderSwitch	Boolean	Toggle from false to true to request the redundant encoder system on the EID to switch (if applicable)
DOutCommand	Boolean	EID DOUT pin state control if configured for remote control
AssertFlashRetention	Boolean	Request EID Flash retention flag to assert
MisfireCylinderSelect_enum	UInt8	Sets the cylinder selection enumeration for misfire test mode. See lecm_eid_misfire_cylinder_select_enum

MisfireLevel_Pct	UInt8	Sets the requested misfire percentage when misfire test mode is active
PrimaryTDCOffset_x16deg	Int16	Sets the primary encoder TDC offset in units of 1/16 degCA if configured for remote configuration. Positive values will advance the offset, negative values will retard the offset.
AlternateTDCOffset_x16deg	Int16	Sets the alternate encoder TDC offset in units of 1/16 degCA if configured for remote configuration. Positive values will advance the offset, negative values will retard the offset.
ContinuityTestModeEnable	Boolean	Set to true to enable continuity test mode
ContinuityTestModeSelect_enum	UInt8	Sets the output # for which to command in continuity test mode. See lecm_eid_continuity_test_mode_output_select_enum
SimulationTestModeEnable	Boolean	Set to true to enable simulation test mode
SimulationTestModeSpeed_RPM	UInt16	Set the simulation test mode speed setpoint
SkipFireEnable	Boolean	Set to true to enable skipfire mode
SkipFireN	UInt16	Sets the number of skips (N) for skipfire mode
SpeedInterpolationSetting_enum	UInt8	Sets the speed interpolation mode on the EID. See lecm_eid_speed_interpolation_enum
InhibitEncoderErrors	Boolean	Request EID to inhibit all encoder errors from reporting

Cylinder PLi Interface

The cylinder Plug Life Indicator (PLi) interface reports PLi values for each output #1 - #20. This interface is only applicable to MixedMode and Ignition variants.

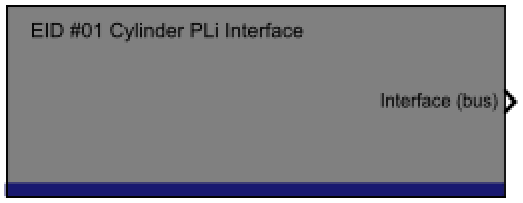


Figure 3-7. EID Cylinder PLi Interface

The data definition for this interface includes the following:

Table 3-4. EID Cylinder PLi Interface Definition

Parameter	DataType	Description
Cylinder01PLi...Cylinder20PLi	Float32	The plug life indicator value for the respective cylinder in units of %

Cylinder Misfire Interface

The cylinder misfire interface reports cylinder misfire speed values for each cylinder #1 - #20.

EID #01 Cylinder Misfire Interface
Interface (bus)

Figure 3-8. EID Cylinder Misfire Interface

The data definition for this interface includes the following:

Table 3-5. EID Cylinder Misfire Interface Definition

Parameter	DataType	Description
Cylinder01RPM...Cylinder20RPM	Float32	The selective misfire speed value for the respective cylinder in units of RPM

Cylinder Diagnostics Interface

The cylinder diagnostics interface reports detailed output diagnostics on a per-cylinder basis speed values for each cylinder #1 - #20. This interface is only applicable to the Ignition variant.

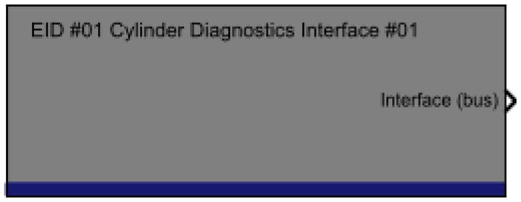


Figure 3-9. EID Cylinder Diagnostics Interface

The data definition for this interface includes the following:

Table 3-6. EID Cylinder Diagnostics Interface Definition

Parameter	DataType	Description
Cylinder01...Cylinder20	EIDCylinderDiagnosticsBus	ControlInterface_enum (uint8) EarlyTerminationError (bool) ShortPSToCoilMinusError (bool) ShortPSToCoilPlusError (bool) ShortGndToCoilPlusError (bool) ShortGndToCoilMinusError (bool) ShortCoilError (bool) OpenCoilError (bool) SevereUnderBoostVoltageError (bool) MaxBoostVoltageError (bool) MinBoostVoltageError (bool) MinOnTimeError (bool) OverrunError (bool)

General Signal Interface

The general signal interface reports general high-speed EID data.

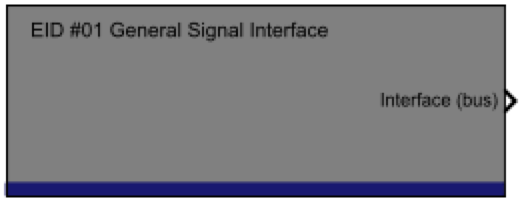


Figure 3-10. EID General Signal Interface

The data definition for this interface includes the following:

Table 3-7. EID General Signal Interface Definition

Parameter	DataType	Description
InstantaneousSpeed_RPM	Float32	Instantaneous speed reports
CycleAverageSpeed_RPM	Float32	Cycle average speed reports

MixedMode & Ignition Control Interfaces

Cylinder Control Interfaces #1 - #3

The cylinder control interfaces #1 - #3 are used to control output drivers that are mapped to the respective interface, including state, duration, and timing. The block accepts an EIDCylinderControlInterfaceBus input type with the following parameters:

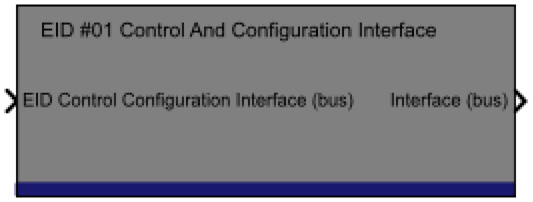

EID #01 Control And Configuration Interface
EID Control Configuration Interface (bus)
Interface (bus)

Figure 3-11. EID Cylinder Control Interfaces #1 - #3

Table 3-8. EID Cylinder Control Interfaces #1 - #3 Definition

Parameter	DataType	Description
CylinderEnables	Boolean [1x20]	True to enable each respective cylinder #1...#20
CylinderDurations	UInt32 [1x20]	The duration command for each respective cylinder #1...#20
TimingOffsets_x16deg	Int16 [1x20]	The individual timing offset in units of for each respective cylinder #1...#20
GlobalTiming_x16deg	Int16	A global timing command applied to all cylinders on the interface
SlotSelect_enum	UInt8	The profile selection for this controlling interface. See lecm_eid_profile_slot_select_enum
TimingUnitSelect_enum	Boolean	The units of the TimingOffsets_x16deg and GlobalTiming_x16deg signals. False = degATDC, True = degBTDC

Injection Control Interfaces

MP Pulse Control Interface #1 - #4

The multi-pulse pulse control interfaces #1 - #4 are used to control the main pulse train of the output drivers that are mapped to the respective interface, including pulse state, base duration, and relative timings. This interface provides the “base” pulse-train commands and is augmented on an individual cylinder basis by the MP Cylinder Control Interfaces. The block accepts an EIDMultiPulseControlInterfaceBus input type with the following parameters:

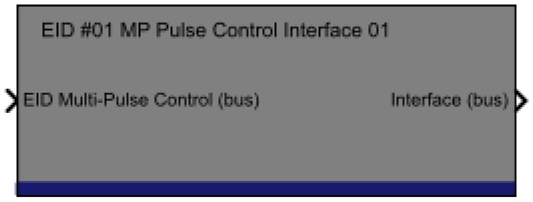

EID #01 MP Pulse Control Interface 01
EID Multi-Pulse Control (bus)
Interface (bus)

Figure 3-12. EID MP Pulse Control Interfaces #1 - #4

Table 3-9. EID MP Pulse Control Interfaces #1 - #4 Definition

Parameter	DataType	Description
Mode_enum	UInt8	Sets the controlling mode of the pulse train. See lesp_crs_fuel_control_mode_enum
TimingRangeResolutionSelect_enum	UInt8	Sets the desired timing range/resolution of the interface. See lesp_crs_fuel_timing_range_resolution_enum
SlotSelect_enum	UInt8	Sets the desired profile used by the interface. See lecm_eid_profile_slot_select_enum
PulseEnables	Boolean [1x5]	True to enable the state of each Pulse #1...Pulse #5
TimingOffsets_degATDC	Int16 [1x5]	Relative timing offsets of each Pulse #1...Pulse #5 relative to each cylinder's TDC
Magnitudes	Float32 [1x5]	Magnitude command of the output drivers based on the selected Mode_enum

MP Cylinder Control Interface #1 - #4

The multi-pulse cylinder control interfaces Cylinder #1-#10 (#11-#20) #1 - #4 are used to adjust the primary pulse train as controlled by the MP Pulse Control interface and make individual adjustments on a per-cylinder basis, including individual cylinder state control, timing, and magnitude trims. The block accepts an EIDMultiPulseCylinderInterfaceBus input type with the following parameters:

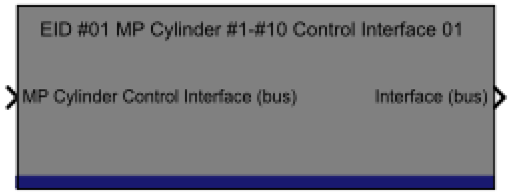

EID #01 MP Cylinder #1-#10 Control Interface 01
MP Cylinder Control Interface (bus)
Interface (bus)

Figure 3-13. EID MP Cylinder Control Interfaces #1 - #4

Table 3-10. EID MP Cylinder Control Interfaces #1 - #4 Definition

Input	DataType	Description
Mode_enum	UInt8	Must match the Mode_enum of the corresponding Pulse Control interface. See lesp_crs_fuel_control_mode_enum
CylinderEnables	Boolean [1x10]	Individual on/off enable control for the respective cylinder
PulseSOITrimApplyFlags	Boolean [1x5]	Set to true to indicate if the Cylinder SOI Trim should apply to the respective Pulse #1...Pulse #5
CylinderSOITrims_x8deg	Int8 [1x5]	Relative timing offsets applied to each Pulse Trim apply indicated pulse relative to each cylinder's TDC. Note: These values have units of 1/8 degree resolution (i.e., 1 = 1/8 degCA).
CylinderMagnitudeTrims	Float32 [1x5]	Relative magnitude trim added to the Main Pulse (Pulse #3) of the corresponding cylinder

Common Rail Signal Interface

The common rail signal interface is used to set the rail pressure command on the EID and accepts the RailPressure as an AnalogSensor input type.

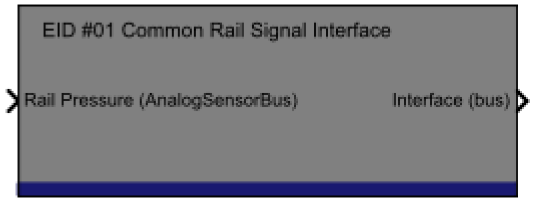

EID #01 Common Rail Signal Interface
Rail Pressure (AnalogSensorBus)
Interface (bus)

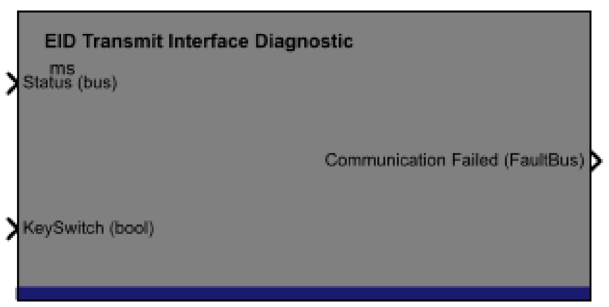
Figure 3-14. EID Common Rail Signal Interface

Table 3-11. EID Common Rail Signal Interface Definition

Input	DataType	Description
Rail Pressure	AnalogSensor	Rail Pressure signal as an AnalogSensor bus input type with units in bar

EID Interface Diagnostics

The EID interface library comes with standard receive and transmit interface diagnostics that can be associated with a fault to indicate an error condition has occurred with the interface. The blocks output a FaultBus type and accept the EIDTxInterfaceStatusBus and EIDRxInterfaceStatusBus types in the Status ports respectively.


EID Transmit Interface Diagnostic
ms
Status (bus)
Communication Failed (FaultBus)
KeySwitch (bool)

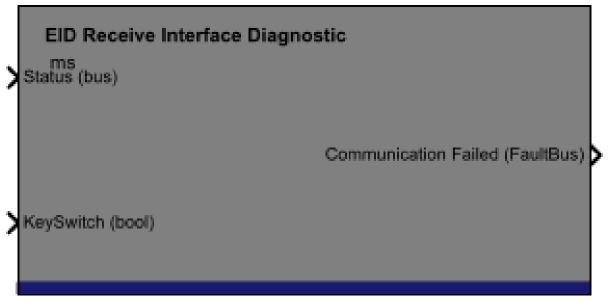

EID Receive Interface Diagnostic
ms
Status (bus)
Communication Failed (FaultBus)
KeySwitch (bool)

Figure 3-15. EID Interface Transmit/Receive Diagnostics

In addition, the EID Faults diagnostic block can be used to create 7 summarized faults for the respective EID instance as follows:

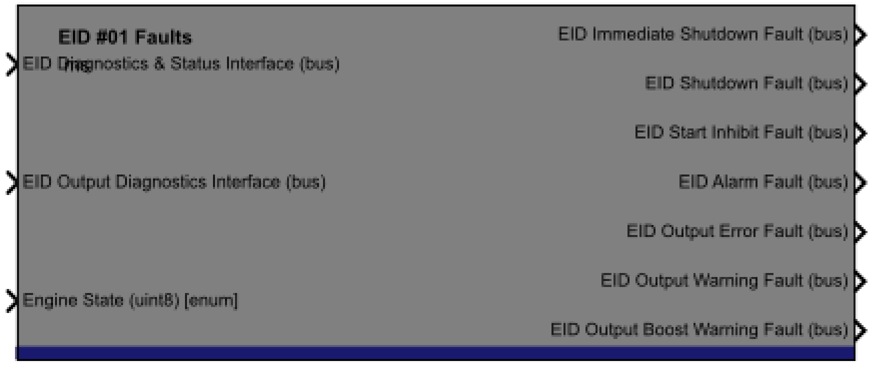

EID #01 Faults
ms
EID Diagnostics & Status Interface (bus)
EID Output Diagnostics Interface (bus)
Engine State (uint8) [enum]
EID Immediate Shutdown Fault (bus)
EID Shutdown Fault (bus)
EID Start Inhibit Fault (bus)
EID Alarm Fault (bus)
EID Output Error Fault (bus)
EID Output Warning Fault (bus)
EID Output Boost Warning Fault (bus)

Figure 3-16. EID Faults Diagnostic

Chapter 4.
Aux Interfacing

Introduction

The following chapter describes interfacing with Woodward standard platform Aux applications using the prebuilt LESP framework interface blocks. Before getting started, it's important to understand the Woodward standard Aux applications variants.

Aux 24 Thermocouple

The Aux 24 thermocouple software is designed to provide 24 extra thermocouples from the aux board. Datalink communications include ASB shared memory region interfaces as well as configurable J1939 for external communications. Both J- and K-type thermocouples are supported with open-wire detection.

Aux Acoustic Knock + Thermocouple

The Aux acoustic knock + thermocouple software is designed for up to 20 cylinders of acoustic knock and up to 24 thermocouples. Any desired mix of acoustic knock and thermocouples is possible based on hardware configuration. Thermocouple features match those as defined in the Aux 24 thermocouple variant. Acoustic knock samples knock sensors within a user-defined angular window and calculates a knock ratio or knock intensity metric for each cylinder [0 – 100%] based on observed frequency content within the sampling window. Knock ratios are reported with ASB shared memory region interfaces as well as J1939 standard messages for external communications.

Aux Real Time Combustion Diagnostics & Control (RTCDC) + Thermocouple

The Aux RTCDC + thermocouple software is designed for up to 12 cylinders of in-cylinder pressure sensing and combustion monitoring. Pressure and thermocouple mix depends on hardware variant. Standard combustion analytics (e.g., SOC, heat release, peak pressure, mean effective pressures, etc...) are reported using ASB shared memory region interfaces as well as J1939 messages for external communications. Thermocouple features match those as defined in the Aux 24 thermocouple variant.

Aux Acoustic Knock + RTCDC

The Aux acoustic knock + RTCDC software is design as primarily an acoustic knock application with a small sample set of RTCDC-enabled pressure inputs (e.g., 1, 2, or 4 pressure inputs depending on hardware configuration). The same features for acoustic knock and RTCDC exist; however, these application variants provide a means for acoustic knock protection with limited in-cylinder combustion metric analysis to support acoustic knock setup and/or limited RTCDC-based cylinder monitoring.

Aux Protection + Remote IO

The Aux protection + remote IO application is designed as a general-purpose Aux IO expansion application along with optional configurable protection-related functions. This application is designed with only J1939 Proprietary messaging available on both internal and external CAN datalinks.

LESP Aux Interface Blocks

The Aux Interface blocks can be located in the Simulink Library Browser under the LESP folder.

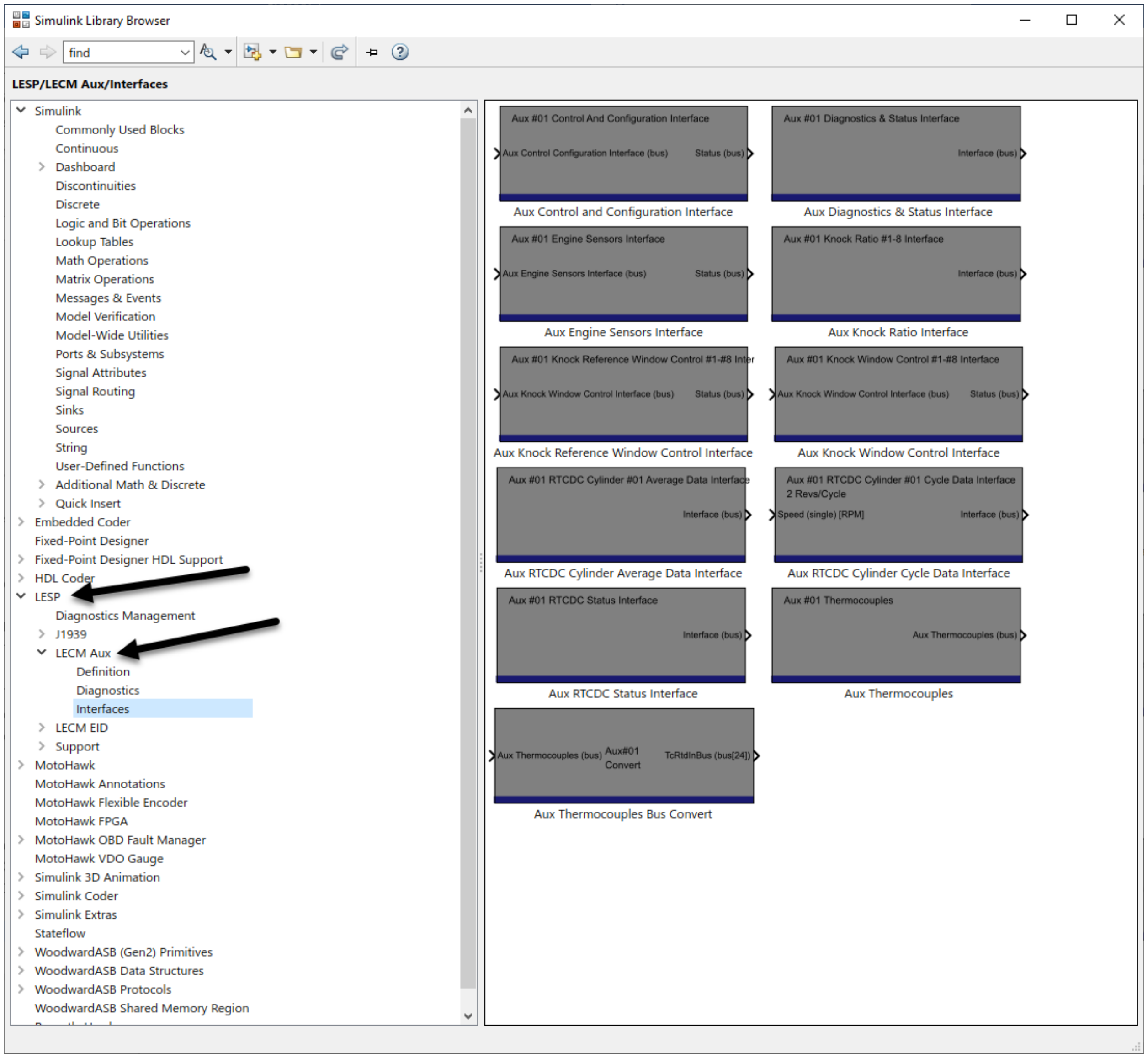

Simulink Library Browser
find
LESP/LECM Aux/Interfaces
Simulink
Commonly Used Blocks
Continuous
Dashboard
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Matrix Operations
Messages & Events
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
String
User-Defined Functions
Additional Math & Discrete
Quick Insert
Embedded Coder
Fixed-Point Designer
Fixed-Point Designer HDL Support
HDL Coder
LESP
Diagnostics Management
J1939
LECM Aux
Definition
Diagnostics
Interfaces
LECM EID
Support
MotoHawk
MotoHawk Annotations
MotoHawk Flexible Encoder
MotoHawk FPGA
MotoHawk OBD Fault Manager
MotoHawk VDO Gauge
Simulink 3D Animation
Simulink Coder
Simulink Extras
Stateflow
WoodwardASB (Gen2) Primitives
WoodwardASB Data Structures
WoodwardASB Protocols
WoodwardASB Shared Memory Region
Aux #01 Control And Configuration Interface
Aux Control Configuration Interface (bus)
Status (bus)
Aux Control and Configuration Interface
Aux #01 Diagnostics & Status Interface
Interface (bus)
Aux Diagnostics & Status Interface
Aux #01 Engine Sensors Interface
Aux Engine Sensors Interface (bus)
Status (bus)
Aux Engine Sensors Interface
Aux #01 Knock Ratio #1-8 Interface
Interface (bus)
Aux Knock Ratio Interface
Aux #01 Knock Reference Window Control #1-#8 Inter
Aux Knock Window Control Interface (bus)
Status (bus)
Aux Knock Reference Window Control Interface
Aux #01 Knock Window Control #1-#8 Interface
Aux Knock Window Control Interface (bus)
Status (bus)
Aux Knock Window Control Interface
Aux #01 RTCDC Cylinder #01 Average Data Interface
Interface (bus)
Aux RTCDC Cylinder Average Data Interface
Aux #01 RTCDC Cylinder #01 Cycle Data Interface
2 Revs/Cycle
Speed (single) [RPM]
Interface (bus)
Aux RTCDC Cylinder Cycle Data Interface
Aux #01 RTCDC Status Interface
Interface (bus)
Aux RTCDC Status Interface
Aux #01 Thermocouples
Aux Thermocouples (bus)
Aux Thermocouples
Aux Thermocouples (bus)
Aux#01 Convert
TcRtdInBus (bus[24])
Aux Thermocouples Bus Convert

Figure 4-1. LECM Aux Interface Blocks

IMPORTANT

For more details on Aux interfaces and descriptions, please refer to the respective application manuals:

Thermocouple: B35112
Acoustic Knock: B35138
RTCDC: B35079
Protection IO: B35188

Definition

As previously described in Chapter 2, under the Model Configuration subsystem, Aux interface definitions should be defined. If interfacing with more than one Aux, enter a new definition with a unique Aux number instance. Note: The definition is a variant subsystem to support compilation on a Main Standalone vs a Main Stacked unit.

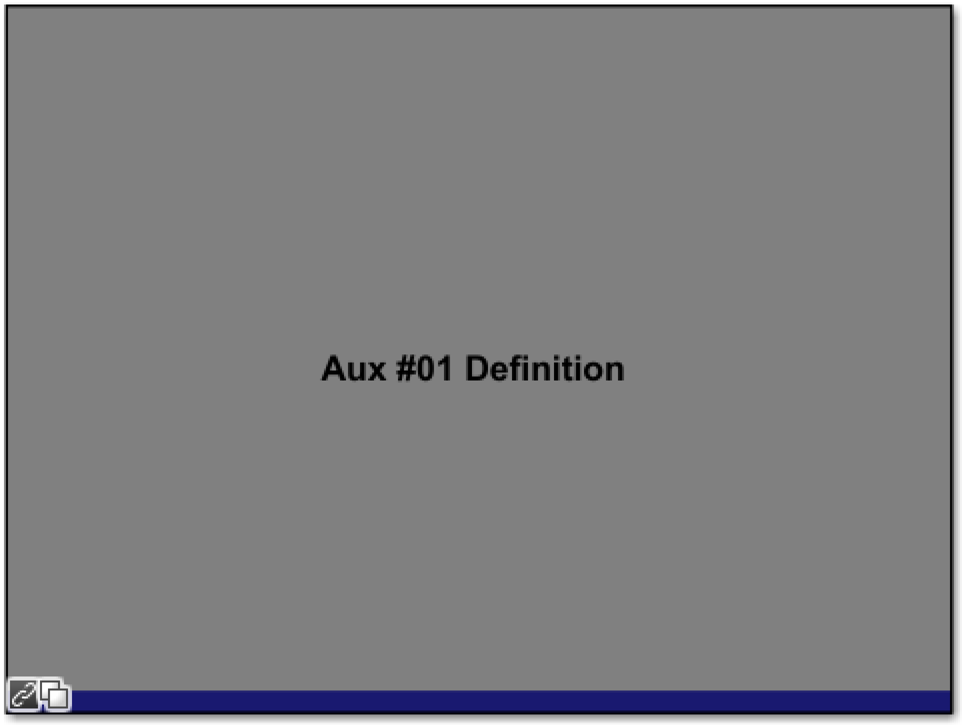


Figure 4-2. Aux Interface Definition

The block variant is controlled by the IsLECMStandaloneVariant workspace parameter, which is defined as follows:

IsLECMStandaloneVariant = strcmp(HardwareConfig.Target, 'LECM-5566-210-065-142x')

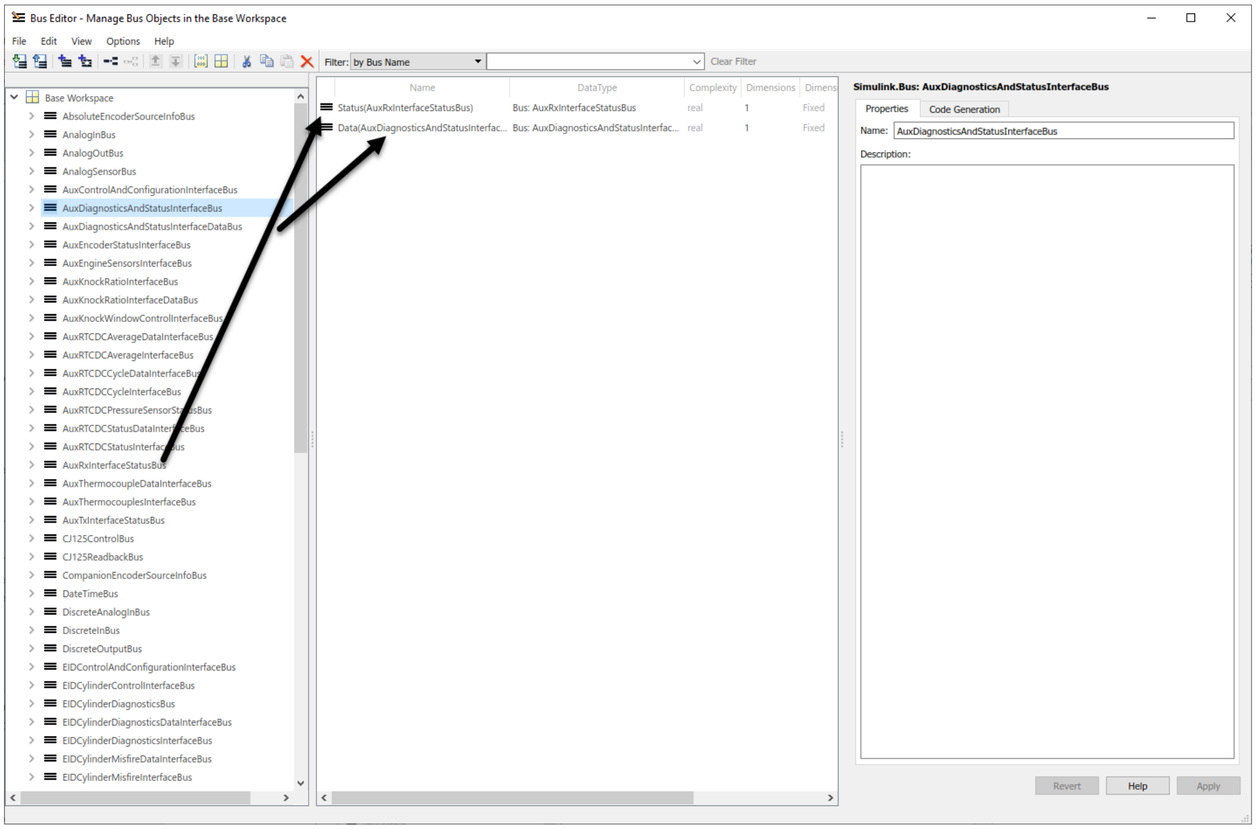
General Aux Interfaces

The following section describes common Aux interface blocks supported for all (except protection IO) Aux application variants. All Aux → Main feedback/reporting interfaces have a two-layer bus interface as follows:

- Interface (bus)
    - AuxRxInterfaceStatusBus
        - IsEnabled (bool)
        - Timeout_ms (uint32)
        - RegionAge_ms (uint32)
        - ConfigurationFault (bool)
        - TimeoutFault (bool)
        - DatalinkSelect_enum (uin8)
    - *DataBus
        - Data is interface dependent

All Main → Aux command interfaces report the following status bus that can be used to drive diagnostics if the interface fails to transmit:

- Interface (bus)
    - AuxTxInterfaceStatusBus
        - IsConnected (bool)
        - TxError (bool)



Base Workspace
- AbsoluteEncoderSourceInfoBus
- AnalogInBus
- AnalogOutBus
- AnalogSensorBus
- AuxControlAndConfigurationInterfaceBus
- AuxDiagnosticsAndStatusInterfaceBus
- AuxDiagnosticsAndStatusInterfaceDataBus
- AuxEncoderStatusInterfaceBus
- AuxEngineSensorsInterfaceBus
- AuxKnockRatioInterfaceBus
- AuxKnockRatioInterfaceDataBus
- AuxKnockWindowControlInterfaceBus
- AuxRTCDCAverageDataInterfaceBus
- AuxRTCDCAverageInterfaceBus
- AuxRTCDCCycleDataInterfaceBus
- AuxRTCDCCycleInterfaceBus
- AuxRTCDCPressureSensorStatusBus
- AuxRTCDCStatusDataInterfaceBus
- AuxRTCDCStatusInterfaceBus
- AuxRxInterfaceStatusBus
- AuxThermocoupleDataInterfaceBus
- AuxThermocouplesInterfaceBus
- AuxTxInterfaceStatusBus
- CJ125ControlBus
- CJ125ReadbackBus
- CompanionEncoderSourceInfoBus
- DateTimeBus
- DiscreteAnalogInBus
- DiscreteInBus
- DiscreteOutputBus
- EIDControlAndConfigurationInterfaceBus
- EIDCylinderControlInterfaceBus
- EIDCylinderDiagnosticsBus
- EIDCylinderDiagnosticsDataInterfaceBus
- EIDCylinderDiagnosticsInterfaceBus
- EIDCylinderMisfireDataInterfaceBus
- EIDCylinderMisfireInterfaceBus

Name	DataType	Complexity	Dimensions	Dimens
Status(AuxRxInterfaceStatusBus)	Bus: AuxRxInterfaceStatusBus	real	1	Fixed
Data(AuxDiagnosticsAndStatusInterfac...	Bus: AuxDiagnosticsAndStatusInterfac...	real	1	Fixed

Simulink.Bus: AuxDiagnosticsAndStatusInterfaceBus

Name: AuxDiagnosticsAndStatusInterfaceBus

Description:

Figure 4-3. Aux → Main Feedback/Reporting Interface Example

Diagnostics & Status Interface

The diagnostics and status interface reports general Aux diagnostic and status information to the main board. It's the responsibility of the application developer to integrate this information into the application as deemed appropriate (see diagnostics example in subsequent section below).

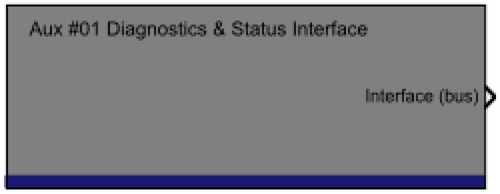


Figure 4-4. Aux Diagnostics & Status Interface

The data definition for this interface includes the following:

Table 4-1. Aux Diagnostics & Status Interface Definition

Parameter	DataType	Description
ApplicationShutdownInhibited	Boolean	True if the Aux application is currently inhibited from shutting down.
IOLockStatus	Boolean	True if the Aux application IOLock is active.
IOLockRequest	Boolean	True if Aux application is currently requesting IOLock.
BackupSettingsInUse	Boolean	True if backup settings are active.

UnconfiguredSettingsInUse	Boolean	True if unit has been freshly programmed and settings have not been saved.
UnexpectedSystemResetFault	UInt8	See lecm_aux_fault_status_enum
MemoryFault	UInt8	
InternalVoltageFault	UInt8	
TemperatureSensorFault	UInt8	
TemperatureHighFault	UInt8	
SupplyVoltageLowFault	UInt8	
SupplyVoltageHighFault	UInt8	
RemoteInterfaceFault	UInt8	
CANPortFault	UInt8	
InternalError	UInt8	
DOut1Fault	UInt8	
DOut2Fault	UInt8	
AOut1Fault	UInt8	
AOut2Fault	UInt8	
AOut1Current_mA	Float32	Current readback of AOut1 in mA
AOut2Current_mA	Float32	Current readback of AOut2 in mA
ActiveEncoderSourceEnum	UInt8	See lecm_aux_active_enc_src_enum
AverageSpeed_RPM	UInt16	Average speed as observed by Aux encoder system
PrimaryCrankStatus	AuxEncoderStatusInterfaceBus	State_enum (uint8) RelativeLoss (bool) ToothCountMismatch (bool) SignalError (bool) LossError (bool) SyncError (bool) ConfigurationError (bool)
PrimaryCamStatus	AuxEncoderStatusInterfaceBus	
PrimarySyncStatus	AuxEncoderStatusInterfaceBus	
AlternateCrankStatus	AuxEncoderStatusInterfaceBus	
AlternateCamStatus	AuxEncoderStatusInterfaceBus	
AlternateSyncStatus	AuxEncoderStatusInterfaceBus	
SoftwareVersionNumber	UInt16	Major software version of the Aux firmware.

Control & Configuration Interface

The control and configuration interface is used as the global control interface for the Aux firmware and supported features. The block accepts an AuxControlAndConfigurationInterfaceBus input type with the following parameters:

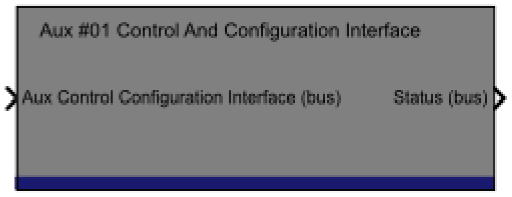

Aux #01 Control And Configuration Interface
Aux Control Configuration Interface (bus)
Status (bus)

Figure 4-5. Aux Control & Configuration Interface

Table 4-2. Aux Control & Configuration Interface Definition

Parameter	DataType	Description
FaultReset	Boolean	Toggle from false to true to request Active faults on the Aux to clear.
InhibitApplicationShutdown	Boolean	Set to true to inhibit the Aux application from executing its shutdown sequence when requested.

EncoderSwitch	Boolean	Toggle from false to true to request the redundant encoder system on the Aux to switch (if applicable).
SimulationTestModeEnable	Boolean	Set to true to enable simulation test mode.
SimulationTestModeSpeed_RPM	UInt16	Set the simulation test mode speed setpoint (standalone units only).
SpeedInterpolationSetting_enum	UInt8	Sets the speed interpolation mode on the Aux. See lecm_aux_speed_interpolation_enum
KnockDetectionEnable	Boolean	Set to true to enable acoustic knock detection.
DOut1Command	Boolean	Aux DOUT1 pin state control if configured for remote control
DOut2Command	Boolean	Aux DOUT2 pin state control if configured for remote control
PrimaryTDCOffset_x16deg	Int16	Sets the primary encoder TDC offset in units of 1/16 degCA if configured for remote configuration. Positive values will advance the offset, negative values will retard the offset.
AlternateTDCOffset_x16deg	Int16	Sets the alternate encoder TDC offset in units of 1/16 degCA if configured for remote configuration. Positive values will advance the offset, negative values will retard the offset.
AOut1Command_uA	UInt16	AOUT1 current command in uA if configured for remote control.
AOut2Command_uA	UInt16	AOUT2 current command in uA if configured for remote control.

Thermocouple Interfaces #1 - #3

The thermocouple interfaces report Aux thermocouple values and statuses to the main board. LESP implements all three interfaces to in a single block to report thermocouple statuses for AIN1...AIN24. Each interface maps to the AIN inputs as follows:

- Thermocouple Interface #1 = AIN1...AIN8
- Thermocouple Interface #2 = AIN9...AIN16
- Thermocouple Interface #3 = AIN17...AIN24

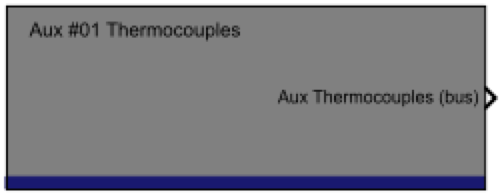


Figure 4-6. Aux Thermocouple Interfaces

The data definition for this interface includes the following:

Table 4-3. Aux Thermocouple Interfaces Definition

Parameter	DataType	Description
Statuses_enum	UInt8 [1x8]	Individual thermocouple statuses. See Iecm_aux_thermocouple_status_enum
Temperatures_degC	Float32 [1x8]	Individual thermocouple temperatures in units of °C

Engine Sensors Interface

The engine sensors interface is used to set the engine state, Manifold Air Pressure, and Engine Coolant Temp signals on the Aux and accepts an AuxEngineSensorsInterfaceBus input type.

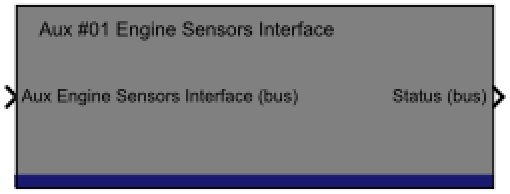

Aux #01 Engine Sensors Interface
Aux Engine Sensors Interface (bus)
Status (bus)

Figure 4-7. Aux Engine Sensors Interface

Table 4-4. Aux Engine Sensors Interface Definition

Parameter	DataType	Description
EngineState_enum	UInt8	A simplified Aux engine state. See Iecm_aux_engine_state_enum
ManifoldAirPress	AnalogSensorBus	The current manifold air pressure in units of kPa
EngineCoolantTemp	AnalogSensorBus	The current engine coolant temperature in units of degC

Acoustic Knock Interfaces

Cylinder Knock Ratio Interfaces #1 - #3

The cylinder knock ratio interfaces #1 - #3 are used to report the cylinder knock ratios (knock intensities). The interfaces mapped to the cylinders as follows:

- Interface #1 = Cylinders #1...#8
- Interface #2 = Cylinders #9...#16
- Interface #3 = Cylinders #17...#24

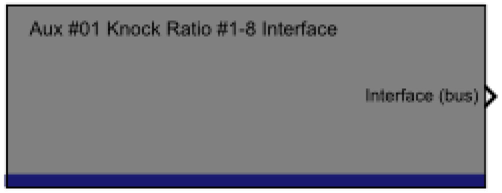


Figure 4-8. Aux Cylinder Knock Ratio Interfaces #1 - #3

Table 4-5. Aux Cylinder Knock Ratio Interfaces #1 - #3 Definition

Parameter	DataType	Description
Statuses_enum	UInt8 [1x8]	Reports the respective cylinder knock window status. See lecm_aux_knock_window_status_enum
NewSamples	Boolean [1x8]	Reports true if the reported knock ratio is a new sample
KnockRatios_pct	Float32 [1x8]	Reports the current cylinder knock ratio in units of %
KnockSaturations_pct	Float32 [1x8]	Reports the current cylinder knock window saturation in units of %
ReferenceSaturations_pct	Float32 [1x8]	Reports the current cylinder reference window saturation in units of %

Cylinder Knock Window Control Interfaces #1 - #3

The cylinder knock window control interfaces #1 - #3 are used to control the knock window start angles and durations from the Main board. The interfaces map to the cylinder knock windows as follows:

- Interface #1 = Cylinder Knock Windows #1 - #8
- Interface #2 = Cylinder Knock Windows #9 - #16
- Interface #3 = Cylinder Knock Windows #17 - #24

The block accepts an AuxKnockWindowControlInterfaceBus input type with the following parameters:

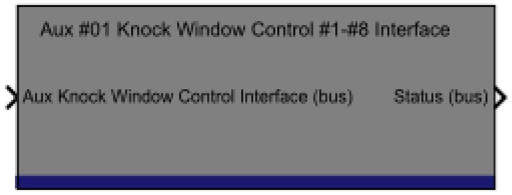

Aux #01 Knock Window Control #1-#8 Interface
Aux Knock Window Control Interface (bus)
Status (bus)

Figure 4-9. Aux Cylinder Knock Window Control Interfaces #1 - #3

Table 4-6. Aux Cylinder Knock Window Control Interfaces #1 - #3 Definition

Parameter	DataType	Description
StartAngles_x16degATDC	Int16 [1x8]	The cylinder knock window start angles in units of degATDC for the respective cylinders
Widths_x16deg	UInt16 [1x8]	The cylinder knock window widths in units of degCA for the respective cylinders

Cylinder Reference Window Control Interfaces #1 - #3

The cylinder reference window control interfaces #1 - #3 are used to control the reference window start angles and durations from the Main board. The interfaces map to the cylinder reference windows as follows:

- Interface #1 = Cylinder Reference Windows #1 - #8
- Interface #2 = Cylinder Reference Windows #9 - #16
- Interface #3 = Cylinder Reference Windows #17 - #24

The block accepts an AuxKnockWindowControlInterfaceBus input type with the following parameters:

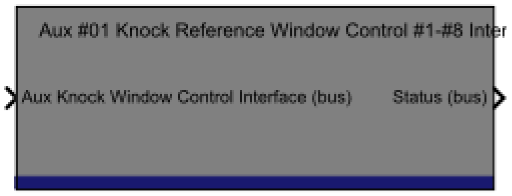

Aux #01 Knock Reference Window Control #1-#8 Inter
Aux Knock Window Control Interface (bus)
Status (bus)

Figure 4-10. Aux Cylinder Reference Window Control Interfaces #1 - #3

Table 4-7. Aux Cylinder Reference Window Control Interfaces #1 - #3 Definition

Parameter	DataType	Description
StartAngles_x16degATDC	Int16 [1x8]	The cylinder reference window start angles in units of degATDC for the respective cylinders
Widths_x16deg	UInt16 [1x8]	The cylinder reference window widths in units of degCA for the respective cylinders

RTCDC Interfaces

RTCDC Status Interface

The RTCDC status interface is used to report the RTCDC sampling status information. The interface reports the following information:

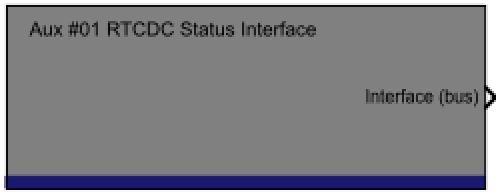


Figure 4-11. Aux RTCDC Status Interface

Table 4-8. Aux RTCDC Status Interface Definition

Parameter	DataType	Description
TotalMissedSamples	UInt32	Reports the total number of RTCDC samples that have been missed.
ADCVectorSamplingStatus_enum	UInt8	Reports the status of the RTCDC sampling engine. See lecm_aux_rtcdc_sampling_status_enum.
MeanCycles	UInt8	Reports the total number of average cycles used for average metric reporting.
PeggingSource_enum	UInt8	Reports the active pegging source. See lecm_aux_rtcdc_pegging_source_enum.
PeggingSourceStatus_enum	UInt8	Reports the status of the pegging source signal. See lecm_aux_rtcdc_pegging_source_status_enum.

RTCDC Cycle Data Interfaces #1 - #20

The RTCDC cycle data interfaces report the combustion metrics for the respective cylinder on a per engine-cycle basis. Because these interfaces are cycle-based, the timeout mechanism is based on a function of the engine speed; thus, the block requires the current speed as an input in order to properly determine if the interface is timed out. The interface reports the following information:

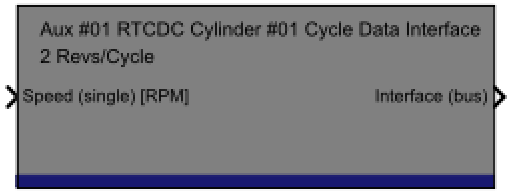


Figure 4-12. Aux RTCDC Cycle Data Interfaces

Table 4-9. Aux RTCDC Cycle Data Interface Definition

Parameter	DataType	Description
PressureSensorStatus	AuxRTCDCPressureSensorStatusBus	IsFaulted (bool) SensorOpenCircuitStatus (bool) SensorShortCircuitStatus (bool) SignalNoiseStatus (bool) SignalDriftedLowStatus (bool) SignalDriftedHighStatus (bool) PeggingErrorStatus (bool) DataValiditySuspectStatus (bool) SensorNonResponsiveStatus (bool)
SOC_degATDC	Float32	Reports the start of combustion in units of degATDC
CA10_degATDC	Float32	Reports the 10% mass fraction burn in units of degATDC
CAx1_degATDC	Float32	Reports the x1% mass fraction burn in units of degATDC
CA50_degATDC	Float32	Reports the 50% mass fraction burn in units of degATDC
CAx2_degATDC	Float32	Reports the x2% mass fraction burn in units of degATDC
CA90_degATDC	Float32	Reports the 90% mass fraction burn in units of degATDC
PMEP_bar	Float32	Reports the pumping mean effective pressure in units of bar.
IMEPGross_bar	Float32	Reports the gross indicated mean effective pressure in units of bar.
IMEPNet_bar	Float32	Reports the net indicated mean effective pressure in units of bar.
COV_IMEP_pct	Float32	Reports the coefficient of variance of IMEP in units of %
PeakPressure_bar	Float32	Reports the peak pressure in units of bar.
PeakLocation_degATDC	Float32	Reports the peak location in units of degATDC.
ROPRMax_barperdeg	Float32	Reports the maximum rate of pressure rise in units of bar/degCA.
MisfireStatus	Boolean	Reports true if the cycle is considered a misfire condition.
MisfireCount	UInt32	Increments by 1 each time a misfire event is observed.
MisfirePercentOfAverage_pct	Float32	A relative average misfire percent.
LatefireStatus	Boolean	Reports true if the cycle is considered a latefire condition.

LatefireCount	UInt32	Increments by 1 each time a latefire event is observed.
LatefirePercentOfAverage_pct	Float32	A relative average of latefire percent.
KnockIntensity_pct	Float32	Reports the pressure-based knock intensity.
KnockRippleSum_bar	Float32	Reports an accumulation of the amount of knock ripple in units of bar.
MaxKnockPressure_bar	Float32	Reports the maximum knock pressure in units of bar.
CombustionIntensity	Float32	Reports the combustion intensity metric.

RTCDC Average Data Interfaces #1 - #20

The RTCDC average data interfaces report the averaged combustion metrics for the respective cylinder. The interface reports the following information:

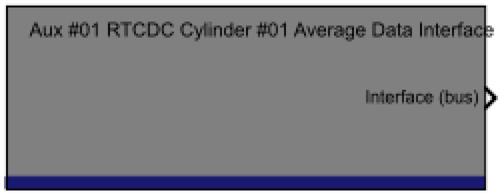


Figure 4-13. Aux RTCDC Average Data Interfaces

Table 4-10. Aux RTCDC Average Data Interface Definition

Parameter	DataType	Description
MeanCycles	UInt8	Reports the current number of cycles used to average
CompressionGammaAverage	Float32	Reports the average compression gamma
ExpansionGammaAverage	Float32	Reports the average expansion gamma
SOCAverage_degATDC	Float32	Reports the average start of combustion
CA10Average_degATDC	Float32	Reports the average 10% mass fraction burn
CA50Average_degATDC	Float32	Reports the average 50% mass fraction burn
CA90Average_degATDC	Float32	Reports the average 90% mass fraction burn
PMEPAverage_bar	Float32	Reports the average pumping mean effective pressure
IMEPGrossAverage_bar	Float32	Reports the average gross indicated mean effective pressure
IMEPNetAverage_bar	Float32	Reports the average net indicated mean effective pressure
PeakPressure_bar	Float32	Reports the average peak pressure
PeakLocation_degATDC	Float32	Reports the average peak location
CumulativeHRAverage_Joule	Float32	Reports the cumulative heat release average
HRRMaxAverage_Joule	Float32	Reports the heat release rate maximum average
HRRMaxLocationAverage_degATDC	Float32	Reports the heat release rate maximum location average
ROPRMaxAverage_barperdeg	Float32	Reports the maximum rate of pressure rise average
KnockIntensityAverage_pct	Float32	Reports the knock intensity average

Aux Interface Diagnostics

The Aux interface library comes with standard receive and transmit interface diagnostics that can be associated with a fault to indicate an error condition has occurred with the interface. The blocks output a FaultBus type and accept the AuxTxInterfaceStatusBus and AuxRxInterfaceStatusBus types in the Status ports respectively.

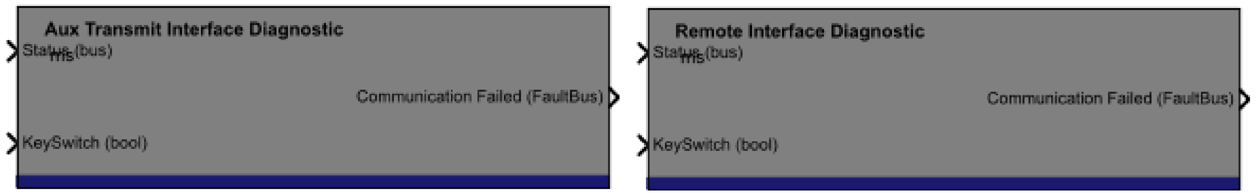

Aux Transmit Interface Diagnostic
ms
Status (bus)
Communication Failed (FaultBus)
KeySwitch (bool)
Remote Interface Diagnostic
ms
Status (bus)
Communication Failed (FaultBus)
KeySwitch (bool)

Figure 4-14. Aux Interface Transmit/Receive Diagnostics

In addition, the Aux Faults diagnostic block can be used to create two summarized faults for the respective Aux instance as follows:

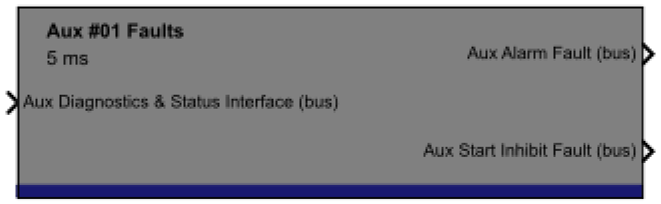

Aux #01 Faults
5 ms
Aux Alarm Fault (bus)
Aux Diagnostics & Status Interface (bus)
Aux Start Inhibit Fault (bus)

Figure 4-15. Aux Faults Diagnostic

Chapter 5. Diagnostics Management

Introduction

The following chapter describes the LE system platform approach to diagnostics management using the MotoHawk OBD fault management blockset. The MotoHawk OBD fault management blockset is used in order to support built-in SAEJ1939 DM messaging and ISO15765 services that are integrated with the OBD fault management blockset, even though OBD-style diagnostics are not used; therefore, only a limited set of OBD fault blocks are used with customizations to support a general non-OBD based fault management structure.

Standard Platform Diagnostic Structure

Fault States

The large engine system platform diagnostic structure implements faults with that support the following states:

<u>Primary Fault State</u>

This is the general state of the fault used to indicate if the fault is inactive, active, and/or previously active.

- Inactive State
  - The fault is current inactive.
- Active State
  - The fault is active. This state maps to the OBD fault state of "Confirmed" and is a latching condition.
- Stored
  - The fault was previously active. This state maps to the OBD fault state of "Previously Active" and is a latching condition.

<u>ACK/NACK Fault State</u>

This is a parallel state to the primary fault state used to indicate if a fault has been acknowledged once it initially enters an active state. This can be used to drive a horn or other form of alarm indication for systems that support fault acknowledgement features without a change to the primary fault state.

- Not Acknowledged
  - A parallel state to the primary fault Inactive/Active/Stored state that indicates the fault has not been acknowledged after the <u>first</u> occurrence.
- Acknowledged
  - Indicates the fault is not in an Active state OR is in an Active state and has been subsequently acknowledged.

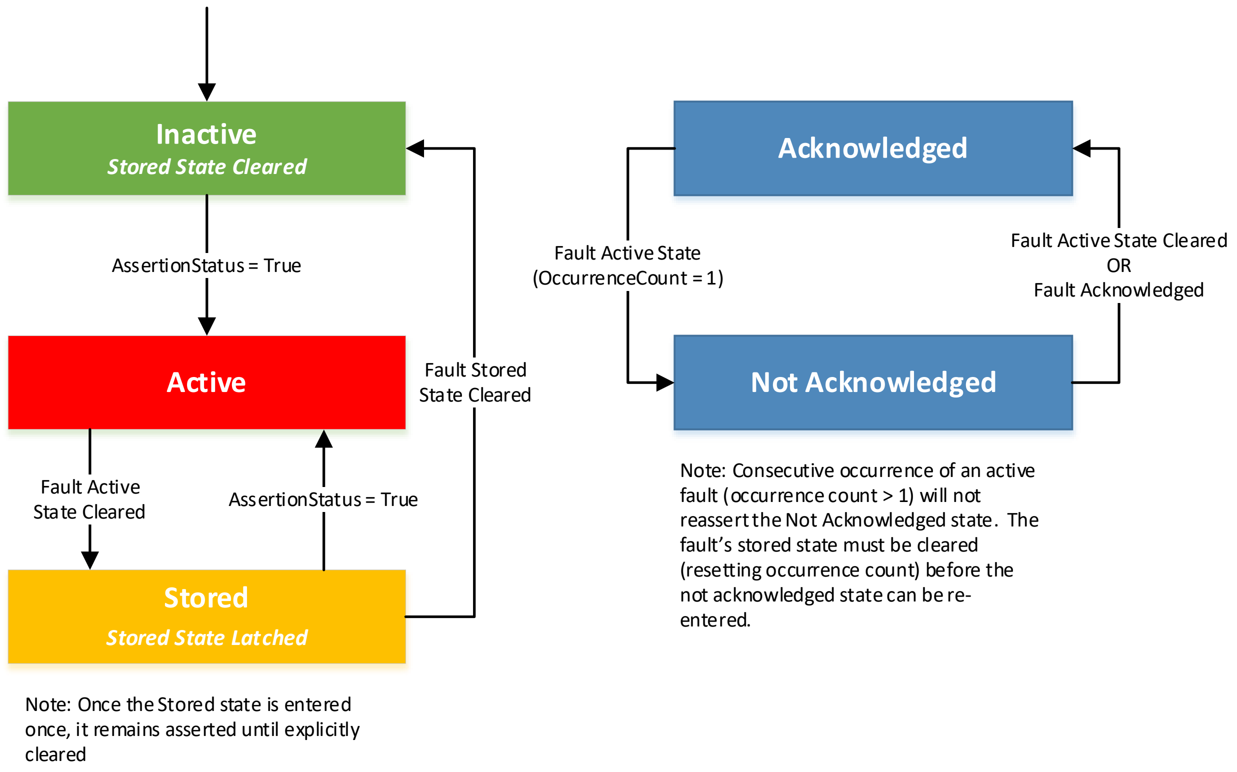

Inactive
Stored State Cleared
AssertionStatus = True
Active
Fault Stored State Cleared
Fault Active State Cleared
AssertionStatus = True
Stored
Stored State Latched
Note: Once the Stored state is entered once, it remains asserted until explicitly cleared
Acknowledged
Fault Active State (OccurrenceCount = 1)
Fault Active State Cleared OR Fault Acknowledged
Not Acknowledged
Note: Consecutive occurrence of an active fault (occurrence count > 1) will not reassert the Not Acknowledged state. The fault's stored state must be cleared (resetting occurrence count) before the not acknowledged state can be re-entered.

Figure 5-1. Fault States

Fault Properties

In addition to the fault states, the platform supports the following properties that are automatically created for each fault instance:

- [ReadOnly] IdentificationCode (uint8) [enum]
  - Allows user to create a unique identification code for the fault in the form of "FLTxxxx" where xxxx are digits from 0000-9999.
- [ReadOnly] ConditionalStatus (boolean)
  - Indicates true if the condition to assert the fault is true, independent of the actual fault assertion status. This logic is used to avoid clearing a fault if the condition for the fault is still valid.
- [ReadOnly] InhibitStatus (boolean)
  - Indicates true if the fault's inhibit status is active, indicating the fault will not assert, regardless of the conditional status.
- [ReadOnly] OccurrenceCount (uint8)
  - Increments by 1 each time the fault enters the Active state. The value is limited from 0 – 127 to support SAEJ1939 DM FMI reporting.
- [ReadOnly] FirstTimeOfOccurrence (datetime)
  - Records the real time clock timestamp of the very first occurrence of the fault (i.e., OccurrenceCount == 1).
- [ReadOnly] RecentTimeOfOccurrence (datetime)
  - Records the real time clock timestamp of the most recent occurrence of the fault (i.e., OccurrenceCount > 0).
- [ReadWrite] FaultCondition1 (uint8) [enum]
  - Enumerated fault condition that asserts the corresponding FaultAction1 fault action when the fault's state matches the condition. Available values include Active, Stored, or Not Acknowledged.

- [ReadWrite] FaultAction1 (uint8) [enum]
    - Used to assert the mapped fault action when the corresponding fault's FaultCondition1 state matches. Fault actions are defined by the application developer.
- [ReadWrite] FaultCondition2 (uint8) [enum]
    - Second instance equivalent to FaultCondition1 that allows for a fault to be mapped to a second fault action.
- [ReadWrite] FaultAction2 (uint8) [enum]
    - Second instance equivalent to FaultAction1.
- [ReadWrite] AutoRecoveryTime (uint8) [s]
    - When non-zero, the fault will automatically clear in this many seconds if the condition for the fault no longer exists. A value of 0 disables this feature and the fault will require an explicit clear request.
- [ReadOnly] AutoRecoveryTimeRemaining (uint8) [s]
    - The total amount of time remaining before the fault will automatically clear.
- [ReadWrite] ActiveClearEngineStoppedRequired (boolean)
    - Check to require the engine to be in a stopped state before the fault will be allowed to clear.
- [ReadOnly] EngineHours (timespan)
    - A snapshot of the engine runtime hours of the most recent occurrence of the fault entering an active state.

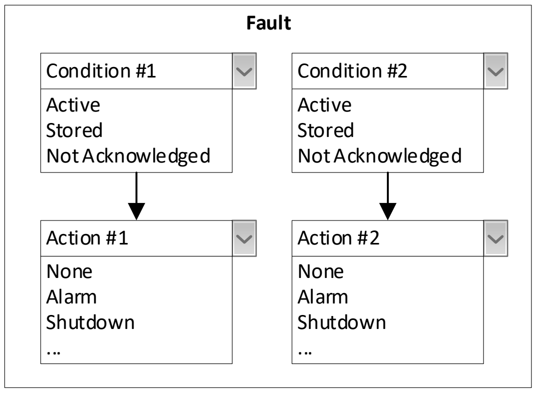
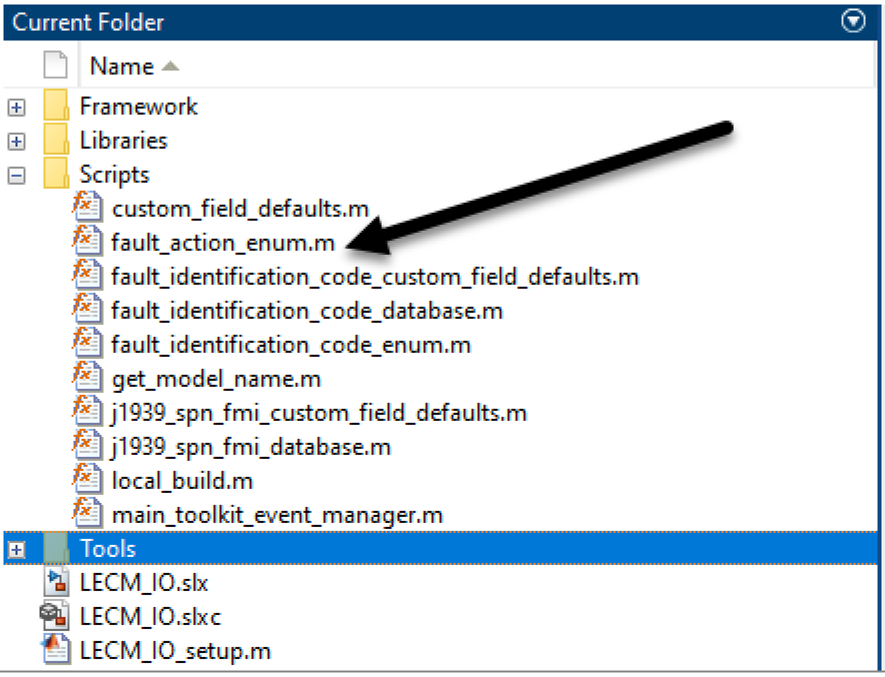


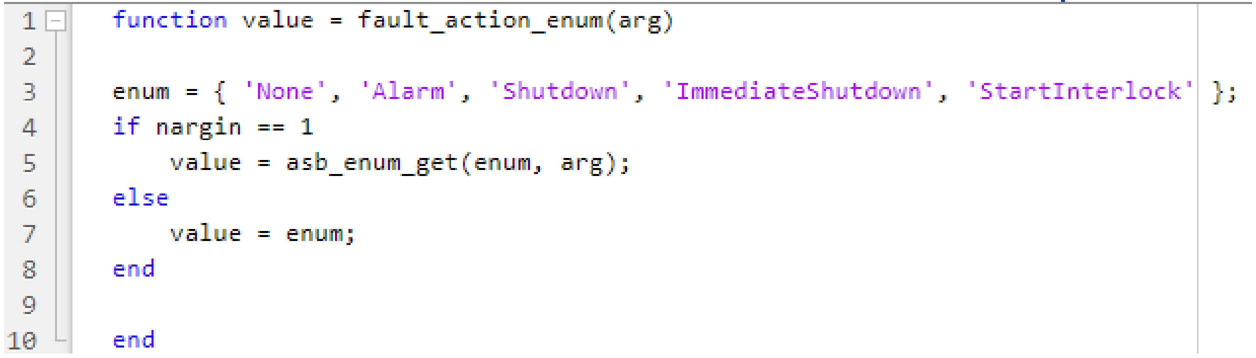
Figure 5-2. Example Fault Condition/Action Mapping

These fields are referred to as "Custom Fields" and effectively allow the developer to extend the "definition" of a fault with additional properties. Each defined custom field is applied to each fault defined in the application.

Fault Actions

Fault actions are logical signals that are defined by the application developer, in which the application is designed to support up to 30 unique actions. Fault actions are defined in the local project ./Scripts directory with the fault_action_enum.m script.


Current Folder
Name
Framework
Libraries
Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m
Tools
LECM_IO.slx
LECM_IO.slxc
LECM_IO_setup.m



```
function value = fault_action_enum(arg)

enum = { 'None', 'Alarm', 'Shutdown', 'ImmediateShutdown', 'StartInterlock' };
if nargin == 1
    value = asb_enum_get(enum, arg);
else
    value = enum;
end

end
```

Figure 5-3. Fault Action Enumeration Definition

The above figure shows an example of five fault actions:

- None
  - This fault action should always be defined to indicate a “no mapping” condition.
- Alarm
  - Used to indicate an alarm indicator status.
- Shutdown
  - Used to indicate a fault is requesting the engine to stop under normal conditions.
- ImmediateShutdown
  - Used to indicate a fault is requesting an emergency stop.
- StartInterlock
  - Used to indicate a fault is requesting the engine starter to be inhibited from engagement

The above fault actions are a sample starting point. Any naming convention of fault actions can be used (up to 30). These will generate the equivalent logical fault action signals that can then be retrieved with the Fault Actions Get block and used in downstream logic to drive the appropriate requested behaviors.

Fault Property Defaulting

ReadWrite fault properties can be configured to a default value using the custom_field_defaults.m script located under the local ./Scripts folder.

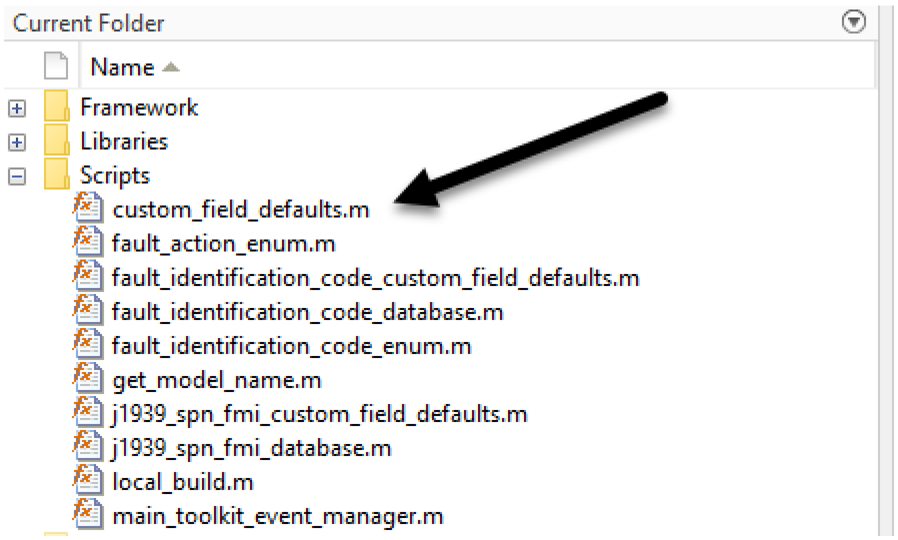
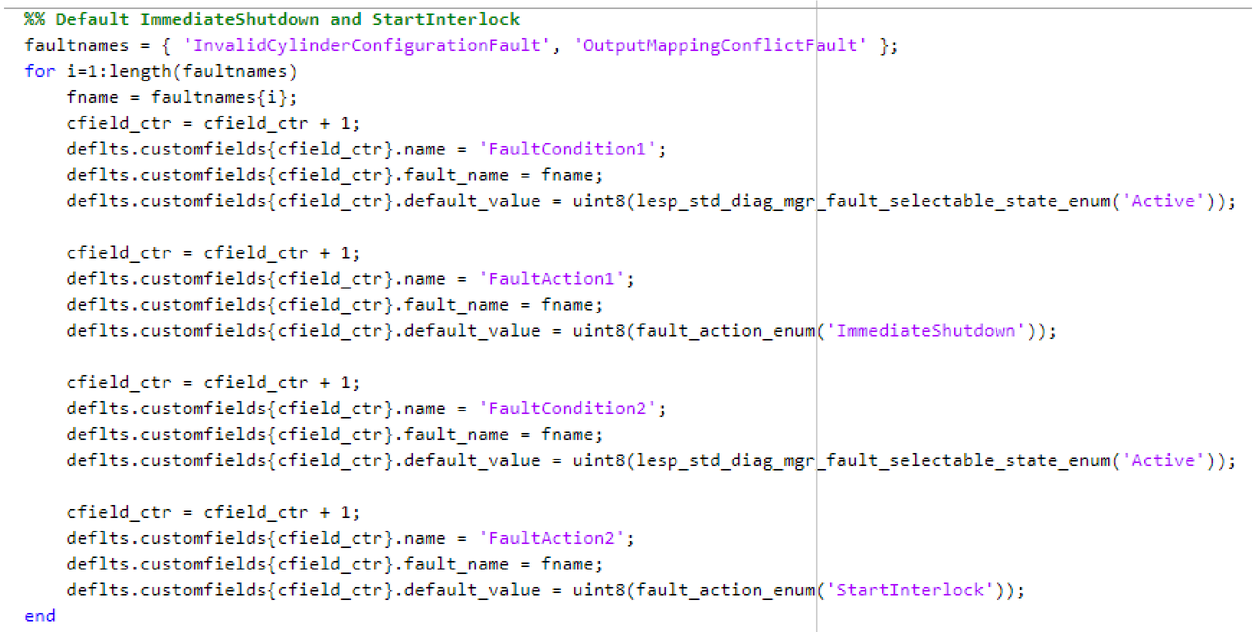

Current Folder
Name
Framework
Libraries
Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m

Figure 5-4. Fault Property Defaults

The figure below illustrates an example of using this file to set fault action mapping defaults for faults named “InvalidCylinderConfigurationFault” and “OutputMappingConflictFault” to Active → ImmediateShutdown and Active → StartInterlock.



```
%% Default ImmediateShutdown and StartInterlock
faultnames = { 'InvalidCylinderConfigurationFault', 'OutputMappingConflictFault' };
for i=1:length(faultnames)
    fname = faultnames{i};
    cfield_ctr = cfield_ctr + 1;
    deflts.customfields{cfield_ctr}.name = 'FaultCondition1';
    deflts.customfields{cfield_ctr}.fault_name = fname;
    deflts.customfields{cfield_ctr}.default_value = uint8(lesp_std_diag_mgr_fault_selectable_state_enum('Active'));

    cfield_ctr = cfield_ctr + 1;
    deflts.customfields{cfield_ctr}.name = 'FaultAction1';
    deflts.customfields{cfield_ctr}.fault_name = fname;
    deflts.customfields{cfield_ctr}.default_value = uint8(fault_action_enum('ImmediateShutdown'));

    cfield_ctr = cfield_ctr + 1;
    deflts.customfields{cfield_ctr}.name = 'FaultCondition2';
    deflts.customfields{cfield_ctr}.fault_name = fname;
    deflts.customfields{cfield_ctr}.default_value = uint8(lesp_std_diag_mgr_fault_selectable_state_enum('Active'));

    cfield_ctr = cfield_ctr + 1;
    deflts.customfields{cfield_ctr}.name = 'FaultAction2';
    deflts.customfields{cfield_ctr}.fault_name = fname;
    deflts.customfields{cfield_ctr}.default_value = uint8(fault_action_enum('StartInterlock'));
end
```

Figure 5-5. Fault Property Defaults Example

SAEJ1939 SPN/FMI Fault Property Extensions

In addition to the standard fault properties, if J1939 Diagnostic Message (DM) is desired, the faults can be extended with SPN and FMI fault properties with the J1939 SPN/FMI Definitions block. This block will add SPN/FMI fault properties to all defined faults using custom field extensions, which are required for proper J1939 DM Diagnostic Trouble Code (DTC) fault reporting.

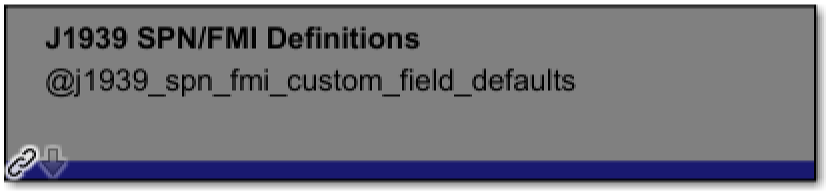

J1939 SPN/FMI Definitions
@j1939_spn_fmi_custom_field_defaults

Figure 5-6. J1939 SPN/FMI Definitions Block

The default J1939 SPN/FMI values are controlled with a database file. The database file must then be converted to a j1939_spn_fmi_custom_field_defaults.m script, which is then referenced by the J1939 SPN/FMI Definition block.

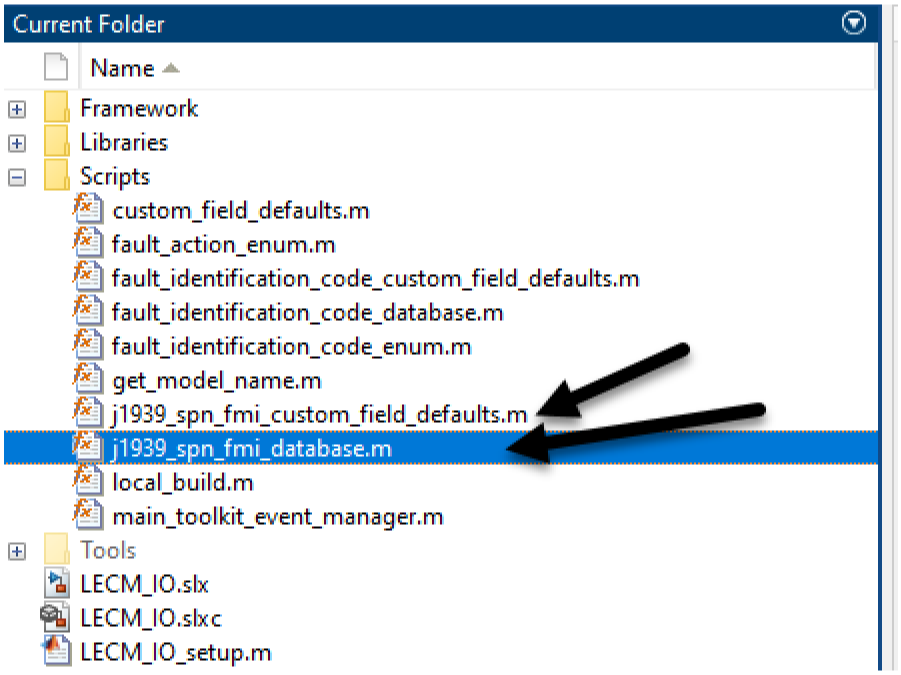
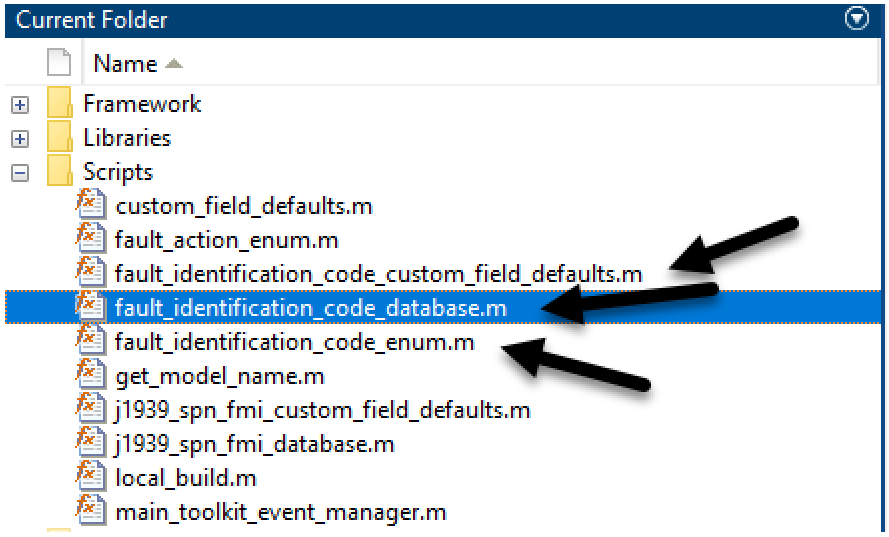

Current Folder
Name
Framework
Libraries
Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m
Tools
LECM_IO.slx
LECM_IO.slxc
LECM_IO_setup.m

Figure 5-7. J1939 SPN/FMI Database

The conversion from the database file to the custom field defaults file can be managed with the LESP Fault Property manager tool. This tool is described in more details in subsequent sections.

Identification Code Fault Property

As previously mentioned, the standard diagnostic management structure supports a fault identification code for each fault, that can be used as a unique numeric identifier for each fault. This property is a uint16 numeric value but presented as an enumeration of the form “FLTxxxx.” The fault identification codes are defined in the fault_identification_code_database.m script under the local ./Scripts directory. The database file must be converted to an equivalent enumeration definition (fault_identification_code_enum.m) and custom field defaults form (fault_identification_code_custom_field_defaults.m). This conversion is handled automatically using the Fault Property Manager (see below).

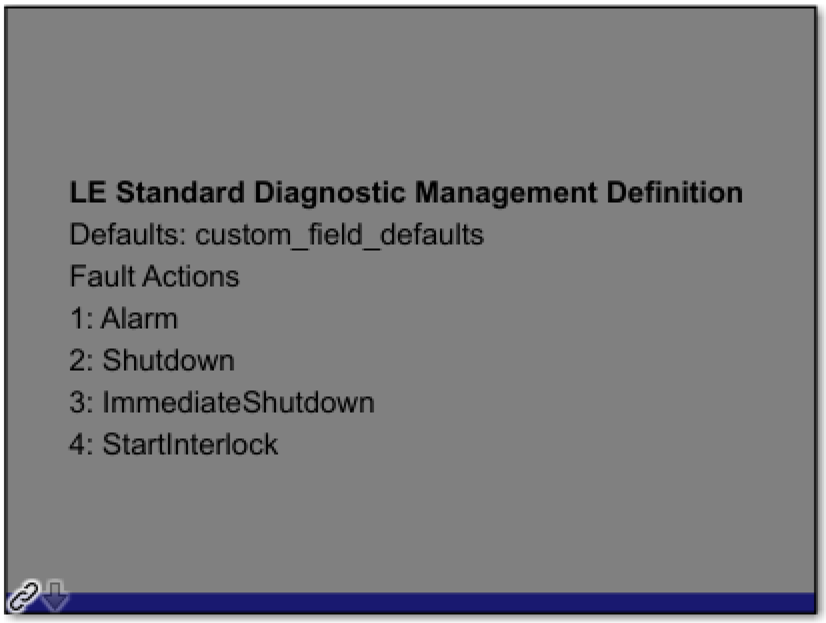


Framework
Libraries
Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m

Figure 5-8. Fault Identification Code Database

Diagnostic Management Definition

The LE system platform standard fault definition block should be located in the Model Configuration subsystem.



LE Standard Diagnostic Management Definition
Defaults: custom_field_defaults
Fault Actions
1: Alarm
2: Shutdown
3: ImmediateShutdown
4: StartInterlock

Figure 5-9. LE Standard Diagnostic Management Definition Block

The mask of the block supports three tabs with configuration parameters:

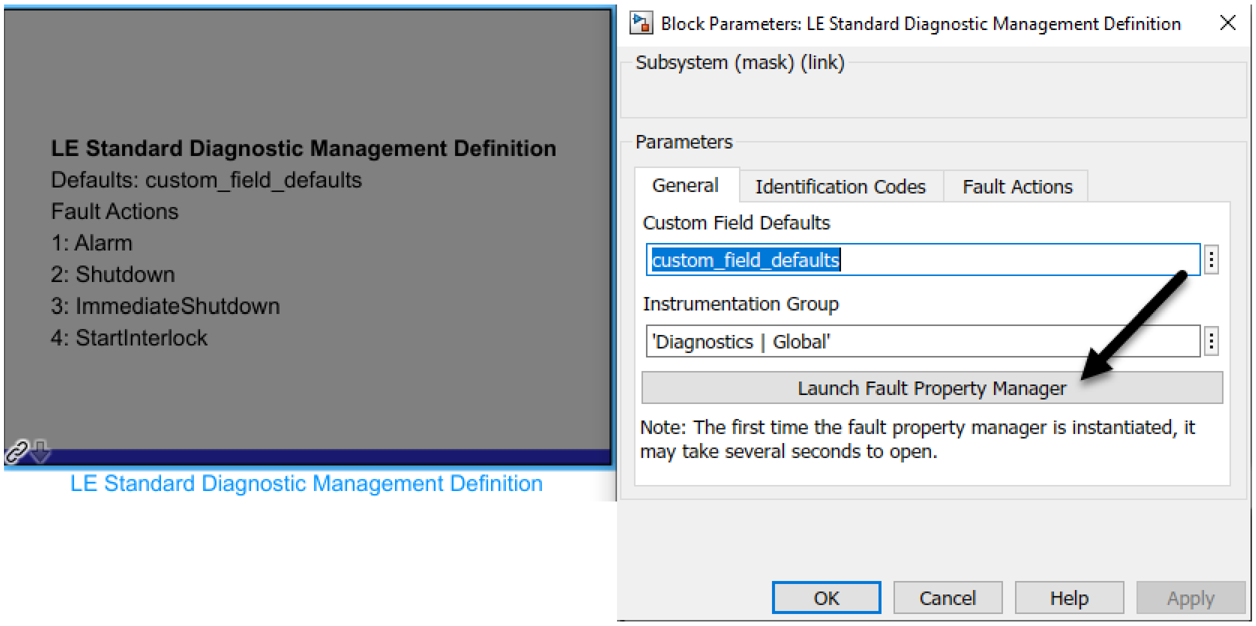
- **General**
  - Custom Field Defaults
    - Enter the local custom_field_defaults script.
  - Instrumentation Group
    - Enter a default instrumentation group path for global D&M properties.
- **Identification Codes**
  - Identification Code Defaults

        - Enter the script for the fault_identification_code_custom_field_defaults
    - Identification Code Enum
        - Enter the script for the fault_identification_code_enum
- **Fault Actions**
    - Fault Actions Enumeration
        - Enter the script for the local fault_action_enum

IMPORTANT **When changing the fault action enumeration definition, this block will require an explicit refresh to update the underlying structure. See Chapter 8 for an example/process of adding a fault action.**

Fault Property Manager

The fault property manager tool is used to manage fault identification codes as well as J1939 SPN/FMI fault associations. The fault property manager can be launched from the fault definition block:


LE Standard Diagnostic Management Definition
Defaults: custom_field_defaults
Fault Actions
1: Alarm
2: Shutdown
3: ImmediateShutdown
4: StartInterlock

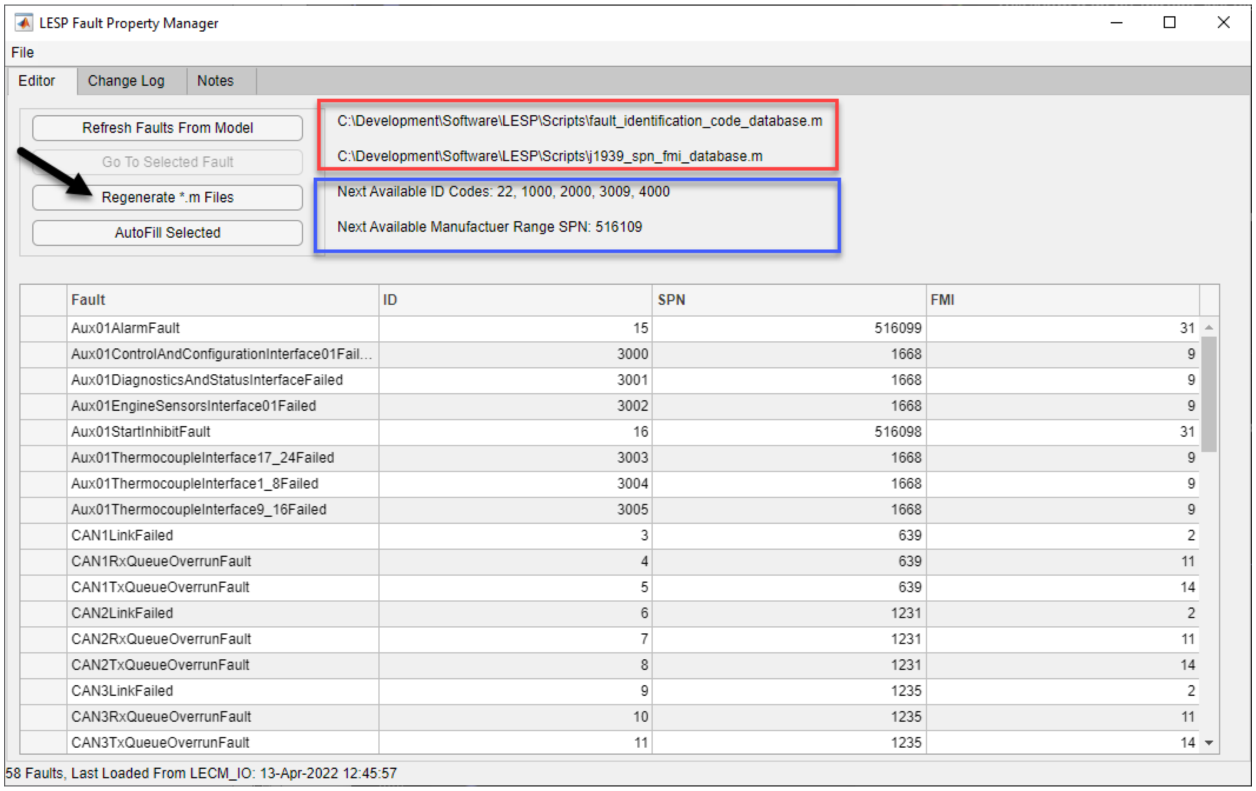
Block Parameters: LE Standard Diagnostic Management Definition
Subsystem (mask) (link)
Parameters
Custom Field Defaults: custom_field_defaults
Instrumentation Group: 'Diagnostics | Global'
Note: The first time the fault property manager is instantiated, it may take several seconds to open.
LE Standard Diagnostic Management Definition
General
Identification Codes
Fault Actions
Launch Fault Property Manager
OK
Cancel
Help
Apply

Figure 5-10. Fault Property Manager Launcher

When the fault manager is first launched, it will load all faults defined in the model. Using the default databases as outlined in the red box (verify the correct files are referenced), it will retrieve the current ID codes, J1939 SPN, and J1939 FMI values for each corresponding fault. If the fault is not defined in the database, it will default to 0. Use this tool, the headers to sort, and the blue box to show the next available values. Replace all 0 valued ID/SPN/FMI defaults with proper values and then click the “Regenerate *.m Files” to automatically regenerate the following files:

- fault_identification_code_database.m
- fault_identification_code_custom_field_defaults.m
- fault_identification_code_enum.m
- j1939_spn_fmi_database
- j1939_spn_fmi_custom_field_defaults.m

This tool provides for a simple method to manage these dependencies files without any direct file manipulation required. In addition, the tool supports the ability to import/export the *.m database files to Microsoft® Excel equivalents.



C:\Development\Software\LESP\Scripts\fault_identification_code_database.m

C:\Development\Software\LESP\Scripts\j1939_spn_fmi_database.m

Next Available ID Codes: 22, 1000, 2000, 3009, 4000

Next Available Manufactuer Range SPN: 516109

Fault	ID	SPN	FMI
Aux01AlarmFault	15	516099	31
Aux01ControlAndConfigurationInterface01Fail...	3000	1668	9
Aux01DiagnosticsAndStatusInterfaceFailed	3001	1668	9
Aux01EngineSensorsInterface01Failed	3002	1668	9
Aux01StartInhibitFault	16	516098	31
Aux01ThermocoupleInterface17_24Failed	3003	1668	9
Aux01ThermocoupleInterface1_8Failed	3004	1668	9
Aux01ThermocoupleInterface9_16Failed	3005	1668	9
CAN1LinkFailed	3	639	2
CAN1RxQueueOverrunFault	4	639	11
CAN1TxQueueOverrunFault	5	639	14
CAN2LinkFailed	6	1231	2
CAN2RxQueueOverrunFault	7	1231	11
CAN2TxQueueOverrunFault	8	1231	14
CAN3LinkFailed	9	1235	2
CAN3RxQueueOverrunFault	10	1235	11
CAN3TxQueueOverrunFault	11	1235	14

58 Faults, Last Loaded From LECM_IO: 13-Apr-2022 12:45:57

Figure 5-11. Fault Property Manager

The figure below illustrates all auto-generated files managed by the property manager, thus not requiring any direct modification by the user (highlighted). All other files are explicitly managed/modified by the user.

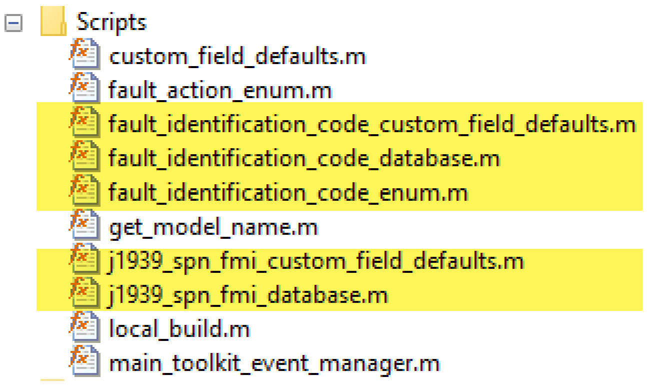

Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m

Figure 5-12. Fault Property Manager Auto-Generated Files

Fault Actions Get

The Fault Actions Get block is used to retrieve the current logical states of all fault actions defined in the Fault Definition (fault_action_enum.m). This block is typically located in the Inputs/Functions and returns the Fault Action states to be consumed in downstream logic to drive appropriate fault action behaviors. It also supports a Fault Protection Override input and built-in tunables to force the fault actions that are mapped to be included with the fault protection override feature to de-assert if the fault protection override input is active.

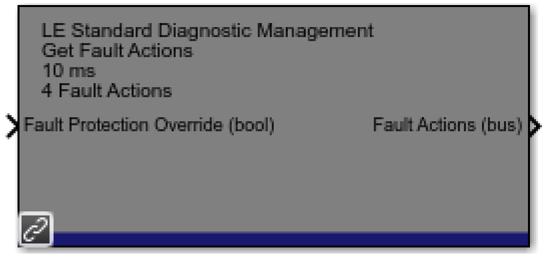

LE Standard Diagnostic Management
Get Fault Actions
10 ms
4 Fault Actions
Fault Protection Override (bool)
Fault Actions (bus)

Figure 5-13. Fault Action Get Block

IMPORTANT **When changing the fault action enumeration definition, this block may require an explicit refresh to update the underlying structure. See Chapter 8 for an example/process of adding a fault action.**

Fault Requests Get

The Fault Request Get block is used to report event counters for fault manager requests. This block is generally located in the Inputs/Application Read subsystem. A change to the event counter can be used in downstream logic to take actions when one of the following requests occurs:

- Clear All Active Fault Request
- Clear All Stored Fault Request
- Acknowledge All Fault Request
- Fault-Related Memory Save Request

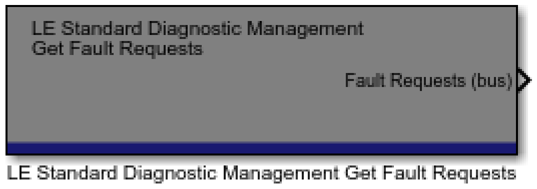

LE Standard Diagnostic Management
Get Fault Requests
Fault Requests (bus)
LE Standard Diagnostic Management Get Fault Requests

Figure 5-14. Fault Requests Get Block

Diagnostic Management Fault Manager

The diagnostic management fault manager block is the core engine for fault management and must be located in an execution context within the model. This block is generally located in Diagnostics Idle or Slow execution context. This block manages fault assertion/de-assertion events, auto-recovery behaviors, and fault timestamping.

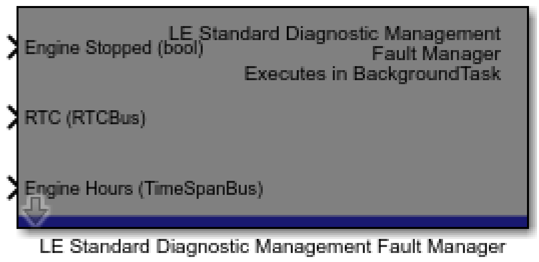

LE Standard Diagnostic Management
Fault Manager
Executes in BackgroundTask
Engine Stopped (bool)
RTC (RTCBus)
Engine Hours (TimeSpanBus)
LE Standard Diagnostic Management Fault Manager

Figure 5-15. Diagnostic Management Fault Manager

ToolKit Event Manager

If using Woodward ToolKit as an HMI, service, or developer tool, display of faults and their corresponding property values is supported with a ToolKit Event Manager. The LE standard diagnostic management implements a standard ToolKit event manager using the LE Standard ToolKit Event Manager block in addition to a toolkit event manager definition that describes the conditions for showing faults, the filter conditions, and the fault properties to show.

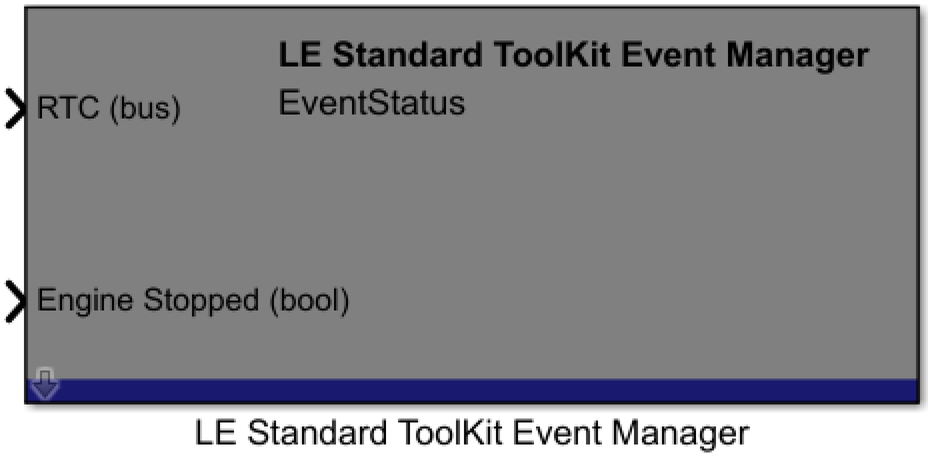
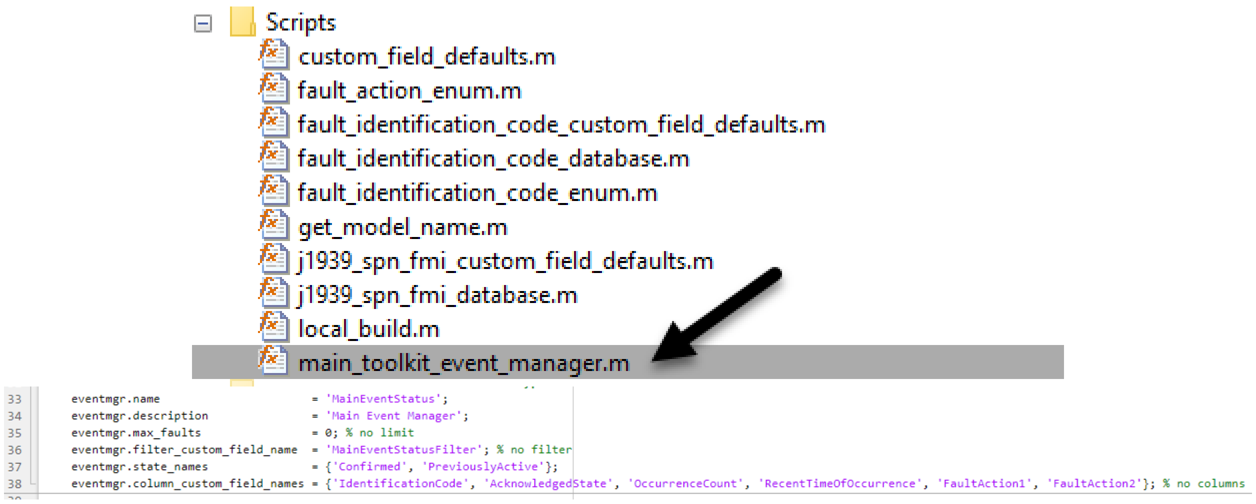

LE Standard ToolKit Event Manager
RTC (bus)
EventStatus
Engine Stopped (bool)
LE Standard ToolKit Event Manager

Figure 5-16. ToolKit Event Manager


Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m

```
eventmgr.name                     = 'MainEventStatus';
eventmgr.description              = 'Main Event Manager';
eventmgr.max_faults               = 0; % no limit
eventmgr.filter_custom_field_name  = 'MainEventStatusFilter'; % no filter
eventmgr.state_names              = {'Confirmed', 'PreviouslyActive'};
eventmgr.column_custom_field_names = {'IdentificationCode', 'AcknowledgedState', 'OccurrenceCount', 'RecentTimeOfOccurrence', 'FaultAction1', 'FaultAction2'}; % no columns
```

Figure 5-17. ToolKit Event Manager Definition File

IMPORTANT

Note: The state names used to show faults must match the OBD fault state names. In the case of LESP D&M implementation, Confirmed OBD fault state maps to the Active fault state and PreviouslyActive OBD fault state maps to the Stored fault state.

Fault Definition

The fault definition blocks are used to define a fault in the model. These blocks are designed to accept a FaultBus type and properly map the Active/Stored states to the OBD fault state equivalents. The library supports two fault definition instances. The basic definition and a secondary definition that also allows for a custom field defaults file specific to the fault to be defined. The fault definition blocks do not support filtering. It’s the responsibility of the application developer to implement fault filtering external to this block if desired.

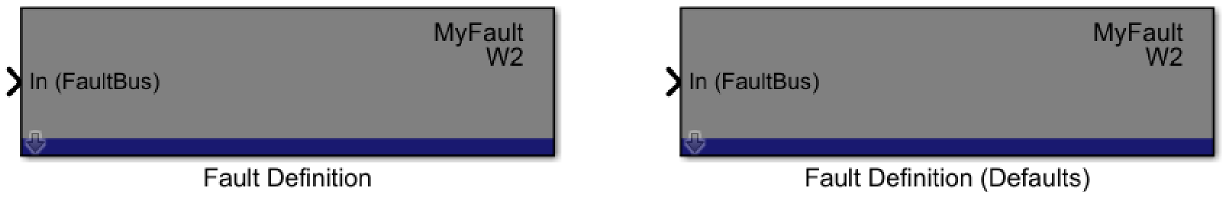


Figure 5-18. Fault Definition Blocks

<u>Example Fault Definition:</u>

In this example, an ASB Range Comparison block is used to detect if a sensor value is out of range. This value is fed into a Boolean Persistence Filter, which is used to filter that the condition persists for at least 1 second before driving the fault assertion status. The conditional status is tied directly to the fault condition (pre-filter). The inhibit logic is based on if the sensor is used (“Disabled”). These signals are assigned to a FaultBus create block, which is an input to the fault definition.

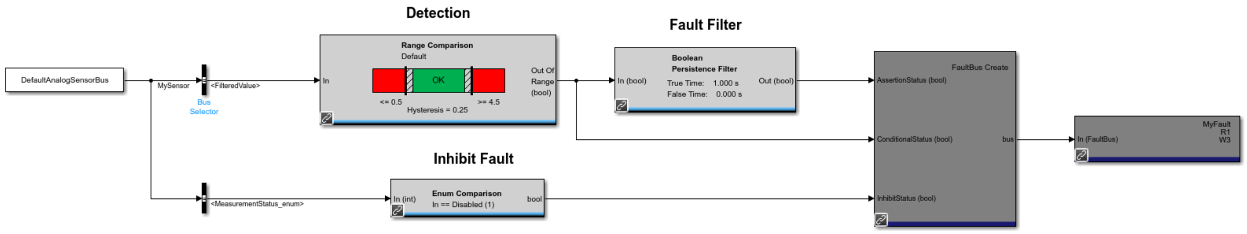


Figure 5-19. Fault Definition Example

Chapter 6.
LECM Main Encoder

Introduction

The following chapter describes LECM Main encoder system definition. This encoder blockset supports standard and configurable NX, N-M, N+1, and SingleTooth encoder patterns with up to a single 3-sensor encoder system with failover to Cam as an optional backup. The encoder logic blocks exist in the lecm_flex_encoder_simple_lib.slx Simulink library and are not made visible in the Simulink browser as these blocks should already be integrated into the package application.

IMPORTANT

This manual assumes the developer has completed MotoHawk flexible encoder training and has familiarity with the flexible encoder blockset.

In addition, the LECM_IO package will be used as an example when describing the encoder implementation; however, the relative locations of blocks and implementation should match other LECM-based packages.

The encoder system defines encoder sources and mappings to speed inputs as follows:

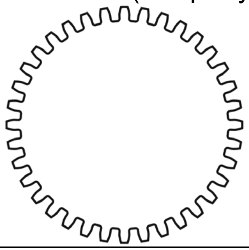
Table 6-1. LECM Main Encoder Sources Definition

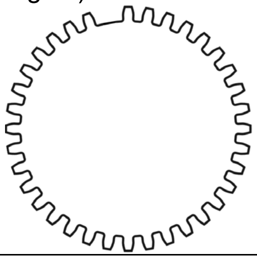
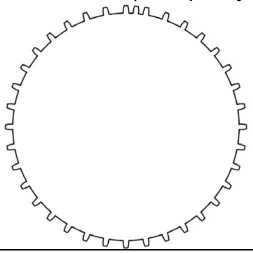
Source	Type/Speed Input	Description
Crank	Absolute Source MainSpeed1	The primary speed Crank encoder source used to track encoder position and drive crank synchronous behaviors.
Cam	Absolute Source MainSpeed2	A secondary absolute encoder source intended for Cam usage that can independently track encoder position and support a failover option (run only on Cam) OR simply be used to provide halfcycle information to the Crank source.
Sync	Companion Source	A supporting encoder source not capable of position tracking but is intended to provide sync information to the Crank source. This source is only used in 3-sensor encoder systems with an NX Crank pattern to provide the “TDC” or “Reset” event.

Supported Encoder Patterns

Any source can be mapped to one of the following encoder patterns:

Table 6-2. LECM Main Encoder Pattern Support

Pattern	Configurations	Description
EncNXConfig1 EncNXConfig2	Runtime configurable from 2 – 750 teeth	Two independent instances of NX encoder pattern definitions (N equally spaced teeth). 

EncNMinusMConfig1 EncNMinusMConfig2	Runtime configurable with N from 2 – 750, M >= 1	Two independent instances of N-M encoder pattern definitions (N equally spaced teeth, with M missing tooth region). 
EncNPlus1Config1 EncNPlus1Config2	Runtime configurable with from 1 to 36	Two independent instances of N+1 encoder pattern definitions (N equally spaced teeth, with 1 extra tooth). 
EncSingleTooth	Fixed	A fixed single one tooth (1X) pattern definition.
Emulated60X	Fixed	A pre-defined unique 60X pattern used for encoder emulation with standard EID/Aux platform applications.

Supported Encoder System Examples

In this first 3-sensor example, the Crank is configured as the primary encoder with an 190X pattern. Since the NX pattern has no unique marker to indicate a "zero" position, the Sync companion source is a single tooth encoder pattern that revolves at the same rate as the crank and provides the "zero" position marker. This is also commonly referred to as the "TDC, Reset, or Sync" event. The Cam also uses the single tooth pattern but revolves once per engine cycle and provides the halfcycle information (0 – 360° vs 360 – 720° phase of the engine cycle) to the Crank source.

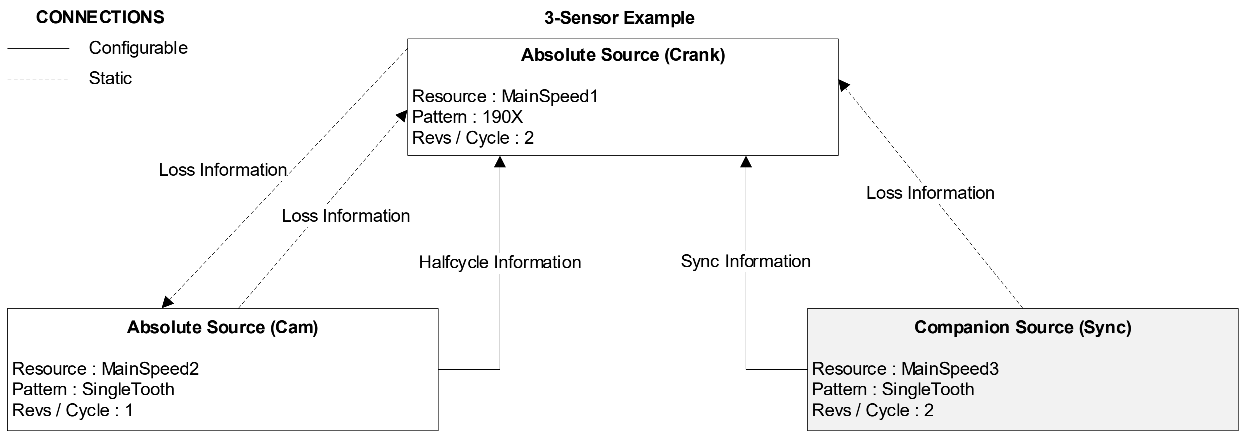


Figure 6-1. 3-Sensor Encoder Example

In this second example, only the Crank and Cam are used in a 2-sensor encoder system. The Crank utilizes a 60-1 pattern. Since this pattern type supports a unique "zero" position marker, a Sync source is not required. The Cam utilizes a single-tooth encoder pattern and provides the halfcycle information to the Crank in the same manner as described in the example above.

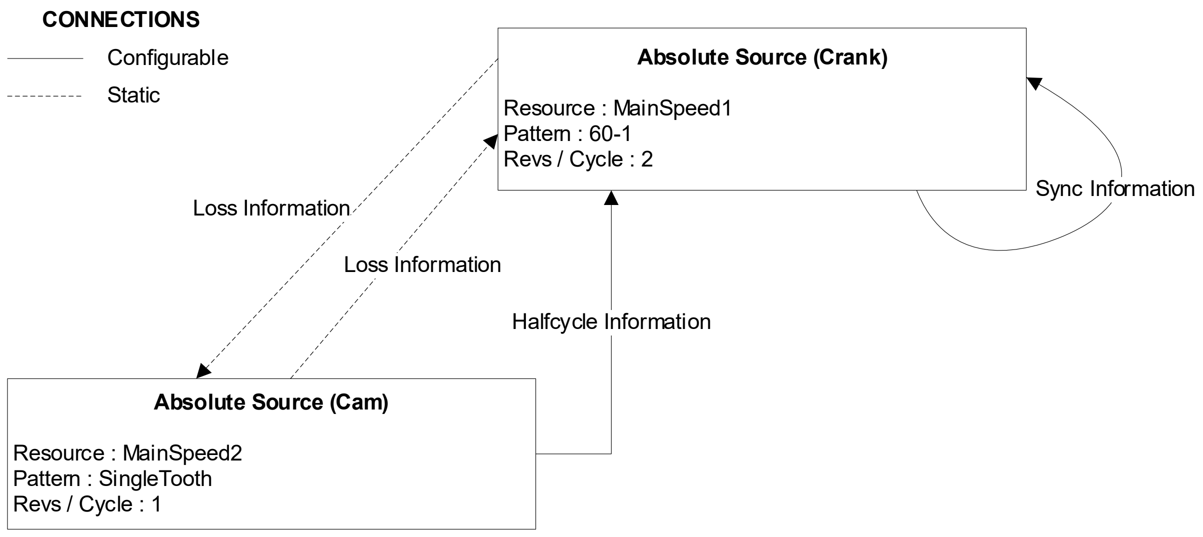


Figure 6-2. 2-Sensor Encoder Example (N-M Crank)

This next example is similar to the example above; however, the Cam also utilizes an N-M pattern like the crank. The Cam still provides half-cycle information to the Crank source since it revolves twice per engine cycle but can also be used as a failover (run on Cam only) if the Crank signal were to be lost. This provides a partially redundant encoder system.

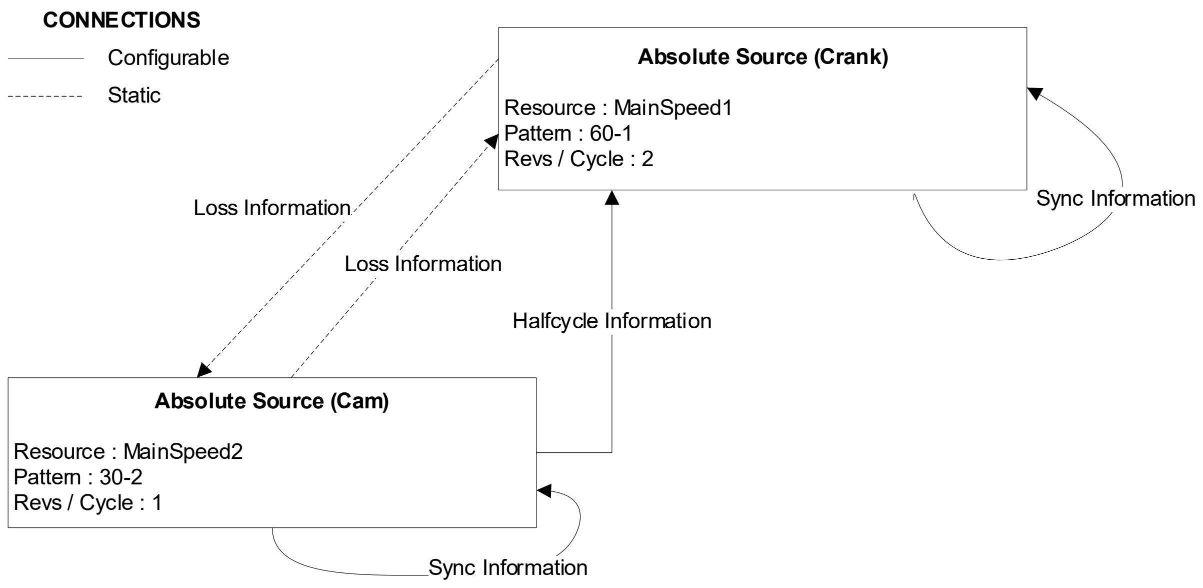


Figure 6-3. 2-Sensor Encoder Example with Cam Failover Option

This next example is similar to the example above; however, the “Crank” source is treated as a Cam in that it revolves only once per engine cycle as well. This type of encoder system in-effect consists of two independent Cam sources (two encoder sources that revolve once per engine cycle) and provides a truly redundant encoder system.

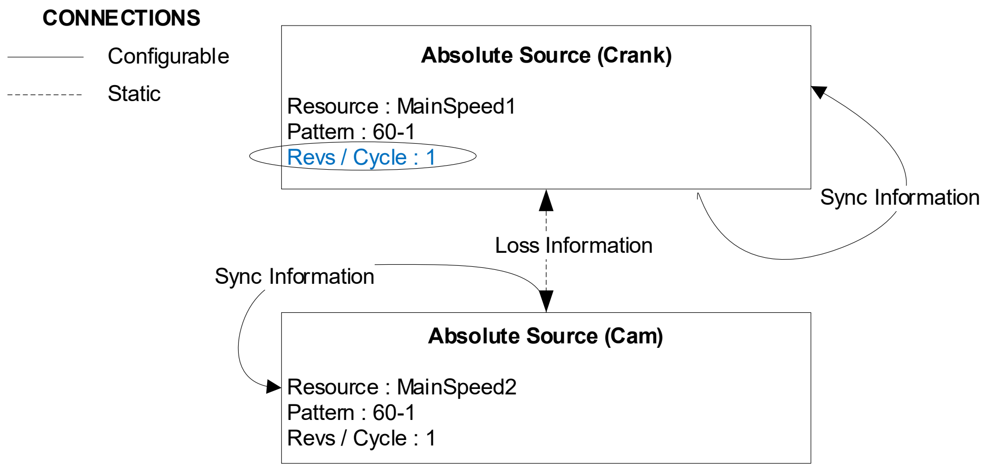


Figure 6-4. 2-Sensor Redundant Encoder System

There are multiple other encoder patterns/configuration options available with this encoder system design; however, if more complex redundant systems are required, the blockset will need to be extended with other speed sources (e.g., Eid/Aux speeds on a stacked LECM) to support redundant 2/3-sensor encoder systems.

Blockset

Encoder Configuration Workspace Variable

As previously mentioned, the blockset also relies on a workspace struct variable denoted as EncoderConfig with the following fields:

- EncoderConfig
    - Stroke (uint8) = 2 or 4
    - System (LecmFlexEncoderSystemsDefn) – This value is only used for variant encoder systems and not applicable to the LECM Main Simple Encoder system.

This variable is set in the <Model>_setup.m script.

LECM Main Encoder Definition

The LECM Main Encoder Definition must first be defined in the model under Model Configuration. This block is used to define the Crank, Cam, Sync encoder sources and encoder patterns.

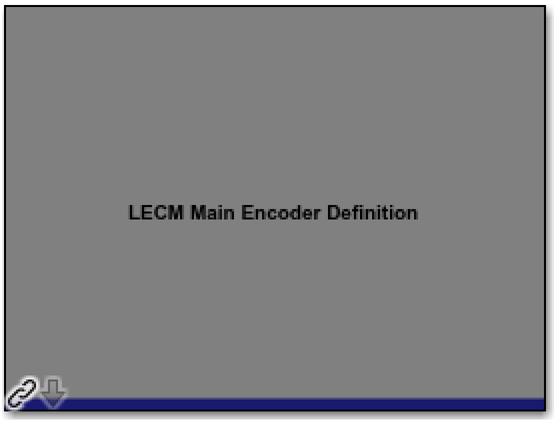


Figure 6-5. LECM Main Encoder Definition

LECM Main Encoder Inputs

The encoder speeds, error events, source information, and other global encoder information is reported via the LECM Main Encoder Inputs block, which should be integrated in the package under the Inputs as illustrated in the example below. This information is made available on an Encoder bus structure to be used in downstream logic.

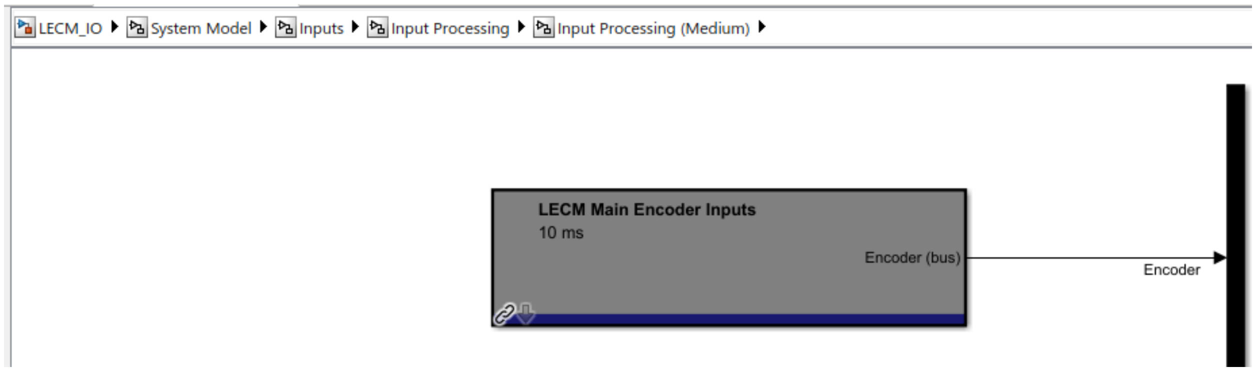


Figure 6-6. LECM Main Encoder Inputs

Global Encoder Information

The first information bus structure on the Encoder inputs bus is of the GlobalEncoderInfoBus, which includes the following data:

Table 6-3. Global Encoder Info Bus Definition

Parameter	**DataType**	**Description**
EncoderState_enum	UInt8	This value reports the LESP-based encoder state. See lesp_encoder_state_enum and a detailed description of the LESP encoder state logic described in a subsequent section.
IsSpeedDetected	Boolean	This flag asserts when speed from any encoder source or pseudo encoder speed is non-zero and active.
IsCrankshaftRotating	Boolean	This flag asserts when speed is observed an any external encoder source (does not include pseudo encoder).
InstSpeed_RPM	Float32	This value reports the instantaneous speed of the current *active* encoder source.
AverageSpeed_RPM	Float32	This value reports the average speed of the current *active* encoder source.
CycleAverageSpeed_RPM	Float32	This value reports the cycle average speed of the current *active* encoder source.
AverageDerivative	EncoderAverageDerivativeBus	This value reports the averaged speed-based derivative of the *active* encoder source.
EncoderAngle_1_16degCA	Float32	Reports the current encoder crank angle in units of degCA (0-360 or 0 – 720).

IsAlternateActive	Boolean	Reports true if the alternate Cam source is the current active encoder source.
ActiveAbsoluteSource_enum	UInt8	Reports the current active encoder source.
DegreesPerEngineCycle_deg	Float32	Reports the current number of degrees per engine cycle (360 for 2-stroke or 720 for 4-stroke).

LESP-based Encoder State

It's important to note the encoder state reporting based on LESP-logic adds an additional state to the standard MotoHawk encoder state:

- lesp_encoder_state_enum
  - [0] Not Created – Reports if encoder system has all sources disabled.
  - [1] ZeroSpeed – Reports a zerospeed condition.
  - [2] Rotating – Reports a non-zerospeed condition; however, the active source has not achieved synchronization.
  - [3] Partial Synchronization – Reports a non-zerospeed condition; however, the active source has not achieved full synchronization.
  - [4] Full Synchronization Pending – This is a LESP-specific state added to indicate MotoHawk encoder system is reporting full synchronization; however, the LESP-based encoder state requires the observation of N user-defined error-free revolutions before accepting the Full Synchronization state.
  - [5] Full Synchronization – System has achieved full synchronization and at least N user-defined error-free revolutions.

This logic provides a mechanism to fully trust the encoder state prior to enabling key features (e.g., injection).

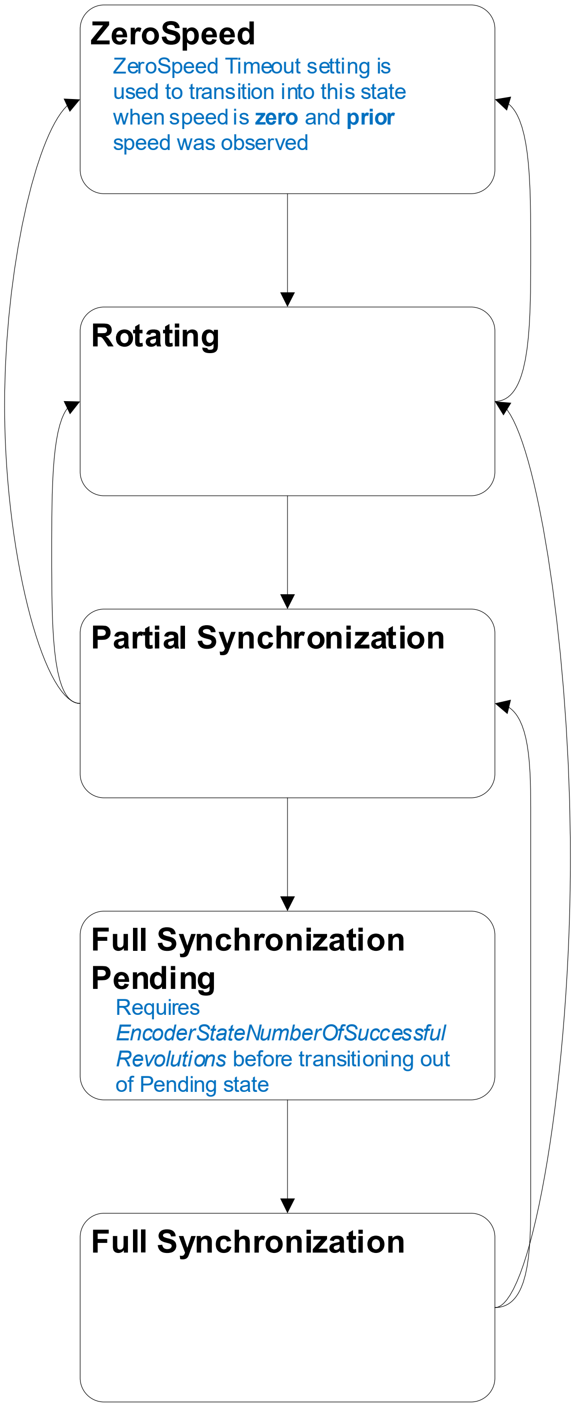

ZeroSpeed
ZeroSpeed Timeout setting is used to transition into this state when speed is **zero** and **prior** speed was observed
Rotating
Partial Synchronization
Full Synchronization Pending
Requires *EncoderStateNumberOfSuccessful Revolutions* before transitioning out of Pending state
Full Synchronization

Figure 6-7. LESP-based Encoder State

Encoder Source Information

The encoder source information bus reports the Crank, Cam, and Sync source-specific content using the AbsoluteEncoderSourceInfoBus for Crank and Cam sources and the CompanionEncoderSourceInfoBus for the Sync source. This information can be used in downstream logic to drive application behaviors or encoder failover logic.

Table 6-4. Absolute Encoder Source Info Bus Definition

Parameter	**DataType**	**Description**
State_enum	UInt8	Reports a source-dependent state. See motohawk_flexible_encoder_state_enum.
InstantaneousSpeed_RPM	Float32	Reports the instantaneous speed of this source.
AverageSpeed_RPM	Float32	Reports the average speed of this source.
CycleAverageSpeed_RPM	Float32	Reports the cycle average speed of this source.
AverageDerivative	EncoderAverageDerivativeBus	Reports the average derivative of this source.
IsCreated	Boolean	Reports true if this source is created.
NumberOfTeethInPattern	UInt16	Reports the number of teeth defined in the associated pattern of this source.
NumberOfRevsPerCycle	UInt8	Reports the number of revolutions per engine cycle defined for this source.
SourceID	UInt8	Reports a unique identifier for this source.
LossCompanionID	UInt8	Reports the unique identifier of another source that is the loss companion of this source.
HalfCycleCompanionID	UInt8	Reports the unique identifier of another source that is assigned as the halfcycle companion of this source.
SyncCompanionID	UInt8	Reports the unique identifier of another source that is assigned as the sync companion of this source.
SyncCompanion	CompanionEncoderSourceInfoBus	Reports the source-specific information of the sync companion for this source.

Table 6-5. Companion Encoder Source Info Bus Definition

Parameter	**DataType**	**Description**
State_enum	UInt8	Reports a source-dependent state. See motohawk_flexible_encoder_compstate_enum.
InstantaneousSpeed_RPM	Float32	Reports the instantaneous speed of this source.
IsCreated	Boolean	Reports true if this source is created.
NumberOfTeethInPattern	UInt16	Reports the number of teeth defined in the associated pattern of this source.
NumberOfRevsPerCycle	UInt8	Reports the number of revolutions per engine cycle defined for this source.
SourceID	UInt8	Reports a unique identifier for this source.
LossCompanionID	UInt8	Reports the unique identifier of another source that is the loss companion of this source.

Encoder Diagnostics Information

The last set of bus structures include the encoder source diagnostic event and state reports. MotoHawk flexible encoder reports encoder errors as events to the application layer. These events are converted to local counters and error states that can be used to drive application behaviors. Each encoder source (Crank, Cam, Sync) reports the following encoder error-related information in an EncoderSourceDiagnosticsBus type. Error fault counts are auto cleared by the encoder state logic. To drive failover behaviors, it's recommended to use the encoder error states.

Table 6-6. Encoder Source Diagnostics Bus Definition

Parameter	**DataType**	**Description**
Info	EncoderSourceInfoEventsBus	Reports event-related information
ErrorEvents	EncoderSourceErrorEventsBus	Reports error event counters
ErrorStates	EncoderSourceErrorStatesBus	Reports error states

Table 6-7. EncoderSourceInfoEventsBus Definition

Parameter	**DataType**	**Description**
NumberOfResyncEvents	UInt32	Increments by 1 each time the encoder observes a synchronization event (i.e., "tooth #0" is observed on the source).
NumberOfTeethBetweenSyncEvents	UInt16	Reports the most recent number of teeth observed between consecutive NumberOfResyncEvents.

Table 6-8. EncoderSourceErrorEventsBus Definition

Parameter	**DataType**	**Description**
AbsentKeyFaultCount	UInt32	Increments by 1 when the respective fault event is observed.
BadHalfCycleWindowFaultCount		
InvertedKeyFaultCount		
LossFaultCount		
PhaseFaultCount		
ReverseRotationFaultCount		
SyncFaultCount		
NoiseSuspectedFaultCount		
SlipFaultCount		
TotalFaultCount	UInt32	An accumulated total of all the above faults.

Table 6-9. EncoderSourceErrorStatesBus Definition

Parameter	**DataType**	**Description**
AbsentKeyErrorState	UInt8	Reports an error state of OK, Intermittent, or Continuous for the respective fault condition.
BadHalfCycleWindowErrorState		
InvertedKeyErrorState		
LossErrorState		
PhaseErrorState		
ReverseRotationErrorState		
SyncErrorState		
NoiseSuspectedErrorState		
SlipErrorState		
TotalErrorState		

LECM Main Encoder Control

Next, is the LECM Main Encoder Control block which should be integrated in the Control subsystem. This block accepts four inputs from user-defined external logic to drive Cam failover conditions and when to enable/activate the pseudo encoder system. Internally, the block supports encoder source TDC offset, and synchronizer slip controls.

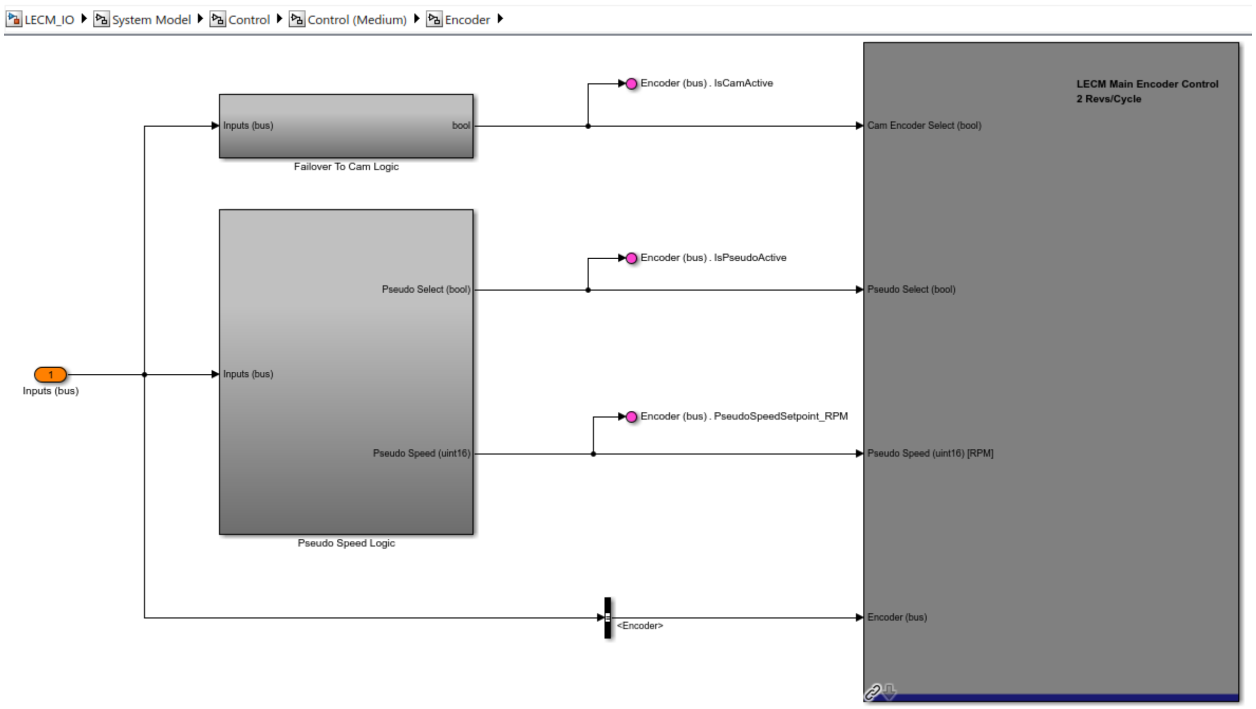

LECM_IO ▸ System Model ▸ Control ▸ Control (Medium) ▸ Encoder ▸
Encoder (bus) . IsCamActive
LECM Main Encoder Control
2 Revs/Cycle
Inputs (bus)
bool
Failover To Cam Logic
Cam Encoder Select (bool)
Encoder (bus) . IsPseudoActive
Pseudo Select (bool)
Pseudo Select (bool)
1
Inputs (bus)
Inputs (bus)
Encoder (bus) . PseudoSpeedSetpoint_RPM
Pseudo Speed (uint16)
Pseudo Speed (uint16) [RPM]
Pseudo Speed Logic
<Encoder>
Encoder (bus)

Figure 6-8. LECM Main Encoder Control

Table 6-10. LECM Main Encoder Control Inputs

Parameter	DataType	Description
Cam Encoder Select	Boolean	Set to true to request the Cam absolute source be the active encoder source.
Pseudo Select	Boolean	Set to true to enable the pseudo encoder system. This will override the active encoder source.
Pseudo Speed [RPM]	UInt16	Sets the desired pseudo encoder speed setpoint.
Encoder	Inputs encoder bus	The bus structure from the LECM Main Encoder Inputs block.

LECM Main Encoder Diagnostic Management

Lastly, the LECM Main Encoder Diagnostic Management block should be integrated in the Diagnostics & Monitoring subsystem of the model. This block is responsible for managing fault counters and error states for each encoder source. In addition, it outputs encoder pattern status FaultBus types that can be integrated into an application fault definition as defined by the developer. An input also exists to inhibit reporting of encoder errors. This input is intended for special use-cases (e.g., engine is stopping) in which reporting of encoder error should be disabled. The block requires the LECM Main Encoder Inputs bus.

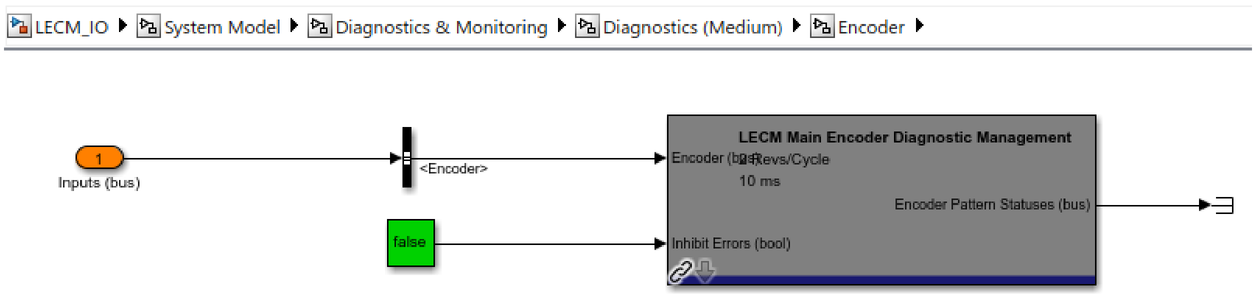


Figure 6-9. LECM Main Encoder Diagnostic Management

Error Events

The LECM Main Encoder Diagnostic Management block implements two sets of encoder error event counters. The primary set increments each error counter by 1 on the report of the respective error event but is automatically cleared by the encoder state logic. These counters are used to drive the encoder state. The second set of error event counters are used for HMI purposes and are denoted as “history” event counters. These counters also increment by 1 on the report of the respective error event but are only cleared when the encoder state transitions *out of* a ZeroSpeed condition. This enables a history of event counts between engine run cycles. It’s recommended to use error state to drive application encoder-related behaviors (e.g., failover logic).

Error Events → State Conversion

The LECM Main Encoder Diagnostic Management block contains the logic for converting observations of encoder error events to a state that is function of the engine cycle. The states consist of the following:

- [0] No Error – No error events
- [1] Intermittent – Error events are being observed intermittently
- [2] Continuous – Error events are being observed continuously

The state conversion is performed using an up-down counter and an emulated TDC1 event. On the observation of an encoder error report, an internal state count increments by the up count. If the count is at or above the intermittent level, the state is equal to *Intermittent*. If the count is at or above the continuous level, the state is equal to *Continuous*. On the next emulated TDC1 event, if the state count has not changed, it will decrement the state count by 1. Once the state count reaches zero, the state will report *No Error*. This provides a mechanism to observe asynchronous reports of encoder error events in the context of an engine cycle and to determine if it is happening intermittently or continuously. The up counts, intermittent, and continuous levels are configurable values. Note the TDC1 events are emulated in the time domain based on the engine speed in order to have an independent TDC1 report from the encoder system.

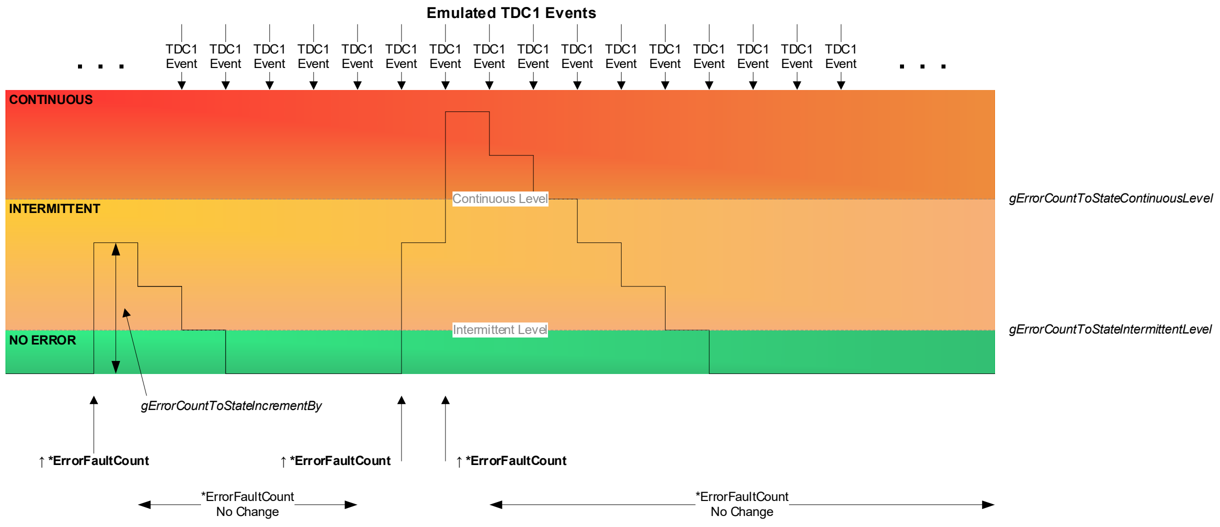

Emulated TDC1 Events
TDC1 Event
CONTINUOUS
INTERMITTENT
NO ERROR
Continuous Level
Intermittent Level
gErrorCountToStateContinuousLevel
gErrorCountToStateIntermittentLevel
gErrorCountToStateIncrementBy
↑ *ErrorFaultCount
*ErrorFaultCount No Change

Figure 6-10. Encoder Error State Logic

LECM Encoder Emulation

Lastly, the LECM Encoder Emulation block configures the ability to emulate an encoder pattern (nominally the Emulated60X pattern) on MainSpeed4, IgnTrig1, and IgnTrig2. For multiple LECM systems or a stacked unit, this enables the encoders for the external LECMs or stacked EID/Aux units to use the emulated encoder signal. This signal remains in sync with the LECM Main encoder system, enabling a single encoder source definition for the entire system. This block by default is located under the Control subsystem. If encoder emulation is not required, MainSpeed4 is needed for other uses, or the IgnTrig1/IgnTrig2 are used for other purposes, this block can be removed or used as a reference for a custom implementation.

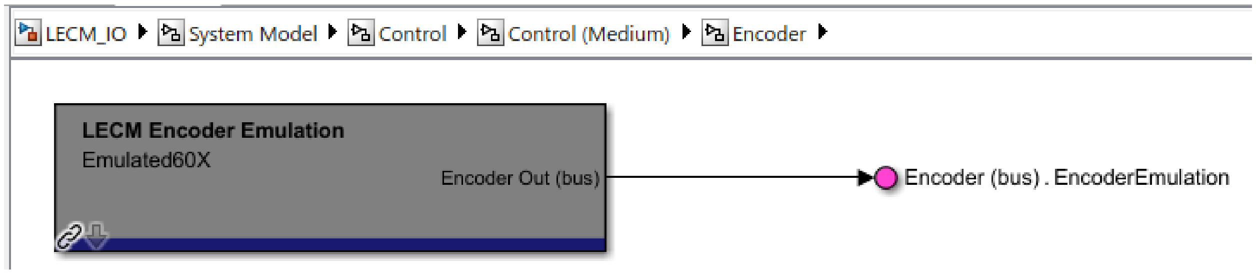


Figure 6-11. LECM Encoder Emulation

Chapter 7. ToolKit

Introduction

The following chapter describes the default ToolKit HMI Service Tool included with the packages. The Service Tool provides a default starting point for interfacing with the default packaged software. It's up to the developer to extend the Service Tool and add new tools and pages for new features.

IMPORTANT

Before getting started, ensure a ToolKit Developer License has been authorized on the developer's PC. The developer license can be verified under the Tools ribbon of ToolKit by clicking on the "License Authorization" button.

This document is not intended to describe the details of using the Woodward ToolKit Service Tool. Please refer to ToolKit training and built-in ToolKit documentation for a full set of ToolKit features and development.

This document will use the LECM IO package as an example; however, different packages may contain different initial page content.

ToolKit Integration/Features

Interfacing with ToolKit requires MotoHawk applications to define XCP communications in addition to a ToolKit integration definition that defines the ToolKit features in the application. The parameter namespace is defined by a Service Interface Definition file (*.sid). This is the ToolKit equivalent of an ASAP2 database (*.a2l) used for other 3rd party XCP-based tools. The SID file is automatically generated by the build process when the ToolKit interface definition is defined. These features include the following:

- ToolKit Event Manager Definitions
- ToolKit Data-Access Based Functions
- ToolKit Programming Support (*.wapp)
- Service Interface Definition File Options (*.sid)

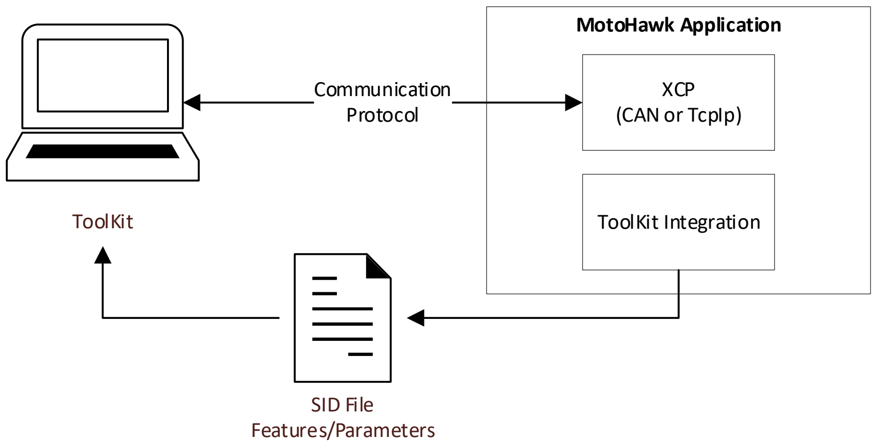
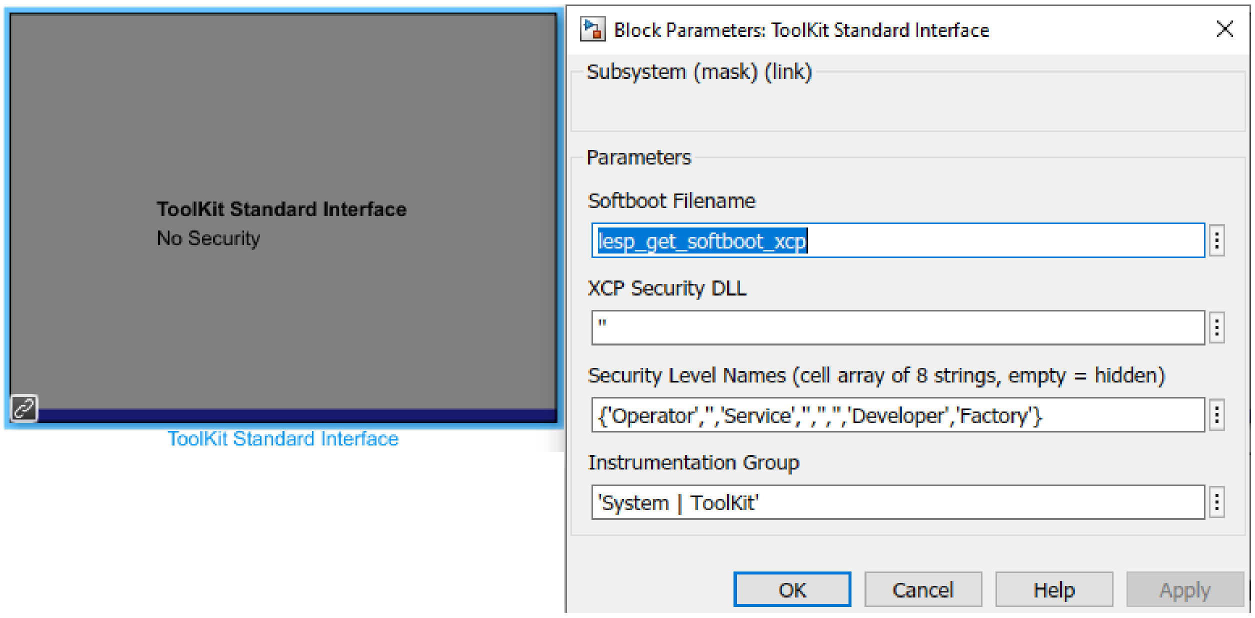


Figure 7-1. ToolKit Interface with MotoHawk Application

LESP ToolKit Standard Interface Definition

LESP implements a standard ToolKit interface definition using the ToolKit Standard Interface Definition block.


ToolKit Standard Interface
No Security
ToolKit Standard Interface
Block Parameters: ToolKit Standard Interface
OK
Cancel
Help
Apply

Subsystem (mask) (link)

Parameters

Softboot Filename: lesp_get_softboot_xcp

XCP Security DLL: ''

Security Level Names (cell array of 8 strings, empty = hidden): {'Operator','','Service','','','','Developer','Factory'}

Instrumentation Group: 'System | ToolKit'

Figure 7-2. LESP ToolKit Standard Interface Definition

This block adds the following ToolKit features:

- SID File Generation with Default Settings
- Woodward Application programming file (*.wapp) support
- DAB Functions
  - IO Lock Request
  - Save Runtime Values Request
  - Save & Reboot Values Request
- XCP Module Reset Request
- [Optional] XCP-based security
- See Diagnostics Management chapter for ToolKit Event Manager integration

SID File Generation

The SID file is automatically generated at the end of the build process and will be located by output Build/TDB directory. It's recommended to use a local build script to perform a post-build action of copying the SID File to a local common SID file repository that is referenced by ToolKit.

Woodward Application Programming File

A toolkit-based Woodward Application programming file (*.wapp) is also generated to support ToolKit flashing of the firmware. The *.wapp file is combination of the softboot programmer and the firmware application and therefore, the hardware-specific softboot file must be provided to generate the *.wapp file. By default, this softboot file is provided in the framework files and retrieved via the *lesp_get_softboot_xcp* function as illustrated in the figure above.

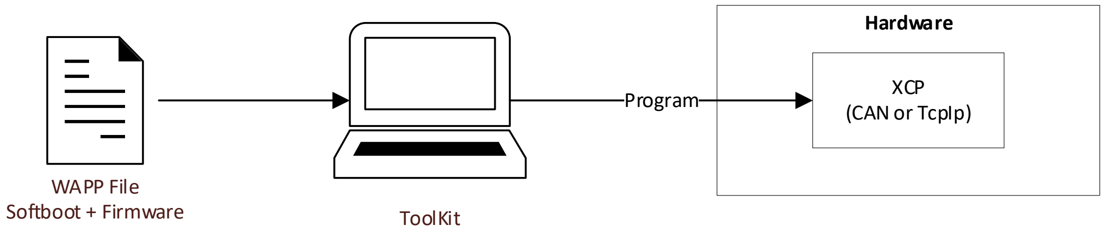


Figure 7-3. ToolKit WAPP Programming

Data-Access Based (DAB) Functions

The standard LESP ToolKit integration supports DAB functions for enabling ToolKit requests to save values and IO Lock the unit while loading settings. Any number of custom DAB functions can be supported; however, the functions marked with [Standard] are default ToolKit built-in functions. DAB functions allow the developer to implement a button event on the Service Tool. The tool sends a request to the control to execute the function and the control will return a success or error code. If the error code is returned, the function is not performed; otherwise, the function should be executed. Error codes returned can result in user-defined error messages to be reported by ToolKit to indicate why the function failed to execute.

[Standard] Save Values Function

The standard Save Values function is implemented in LESP as a “Save Values & Reboot the Control” function. The access to this function on the tool can be added manually with a DAB Function button component but is also integrated into the ribbon menu by default with the Save Values function.

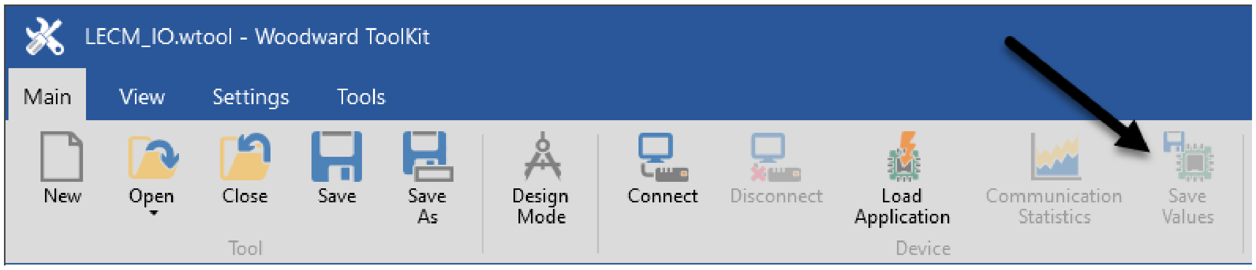


Figure 7-4. ToolKit Save Values Function

When the button is clicked in ToolKit the control sends a Save Values DAB function request to the control and triggers the SaveValuesDABFunction trigger located under the ToolKit Standard Interface definition block. When executed, this trigger sets a response code of 0 if the ToolKitSaveValuesPermissive datastore is true; otherwise, it returns a -1 error code with a message indicating the service is currently not available; however, if the service is available, it increments a ToolKitSaveRequestEvents counter. The ToolKitSaveValuesPermissive is set by user-defined application logic using the Application/Application Write (Idle) subsystem. This enables the application developer to specify conditions for which a save/reboot operation would be allowed to proceed (e.g., only allow during a zero speed condition). Lastly, a counter is incremented to indicate a request has been observed. This counter is subsequently read in the Inputs/Application/Application Read (Idle) subsystem and referenced in the Application/Application Slow/Shutdown Management subsystem to execute the request.

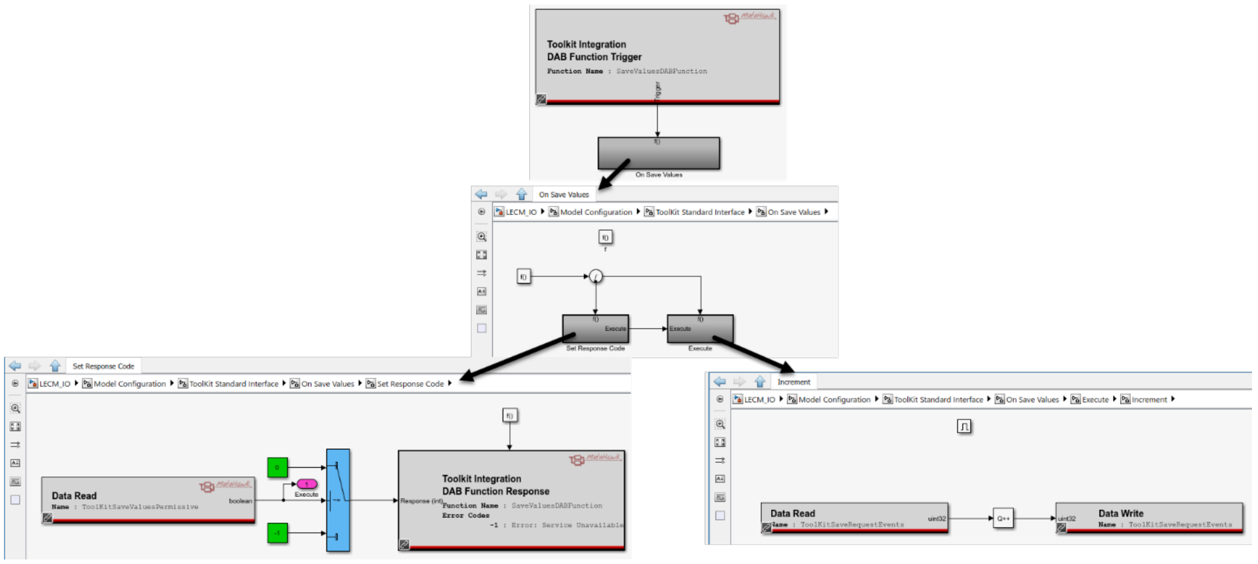

Toolkit Integration
DAB Function Trigger
Function Name : SaveValuesDABFunction
On Save Values
Set Response Code
Execute
Data Read
Name : ToolKitSaveValuesPermissive
Toolkit Integration
DAB Function Response
Function Name : SaveValuesDABFunction
Error Codes
-1 : Error: Service Unavailable
Increment
Data Read
Name : ToolKitSaveRequestEvents
Data Write
Name : ToolKitSaveRequestEvents

Figure 7-5. ToolKit Save Values DAB Function (Manage ToolKit Request)

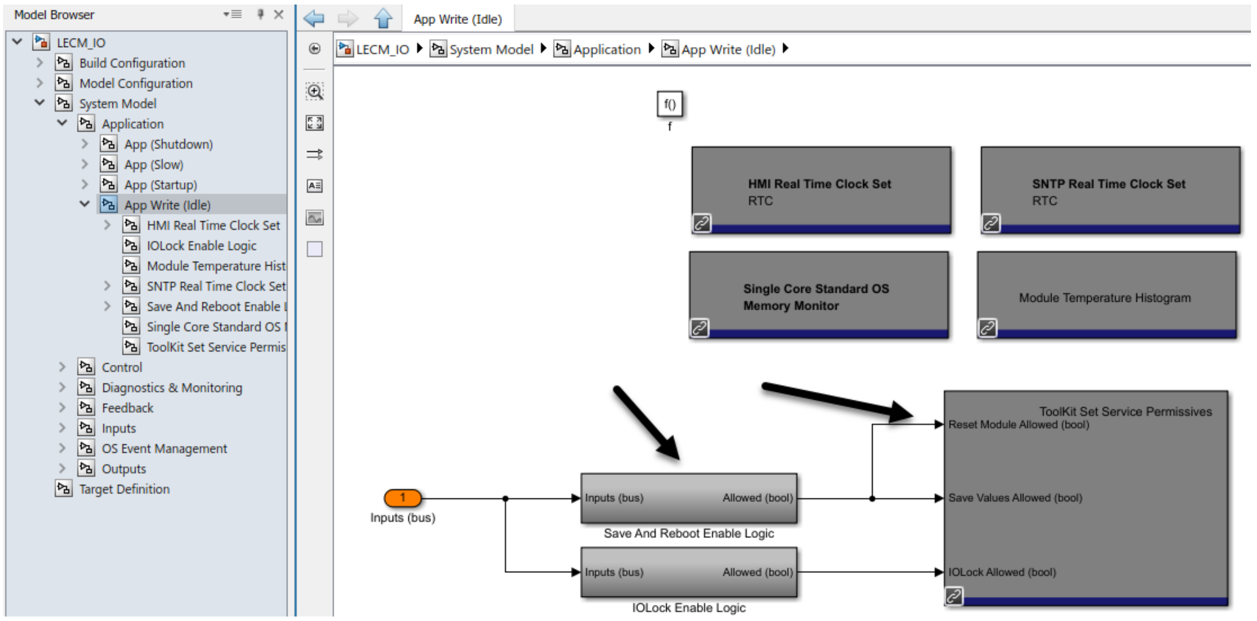


Figure 7-6. ToolKit Permissives (Manage ToolKit Request Permissives)

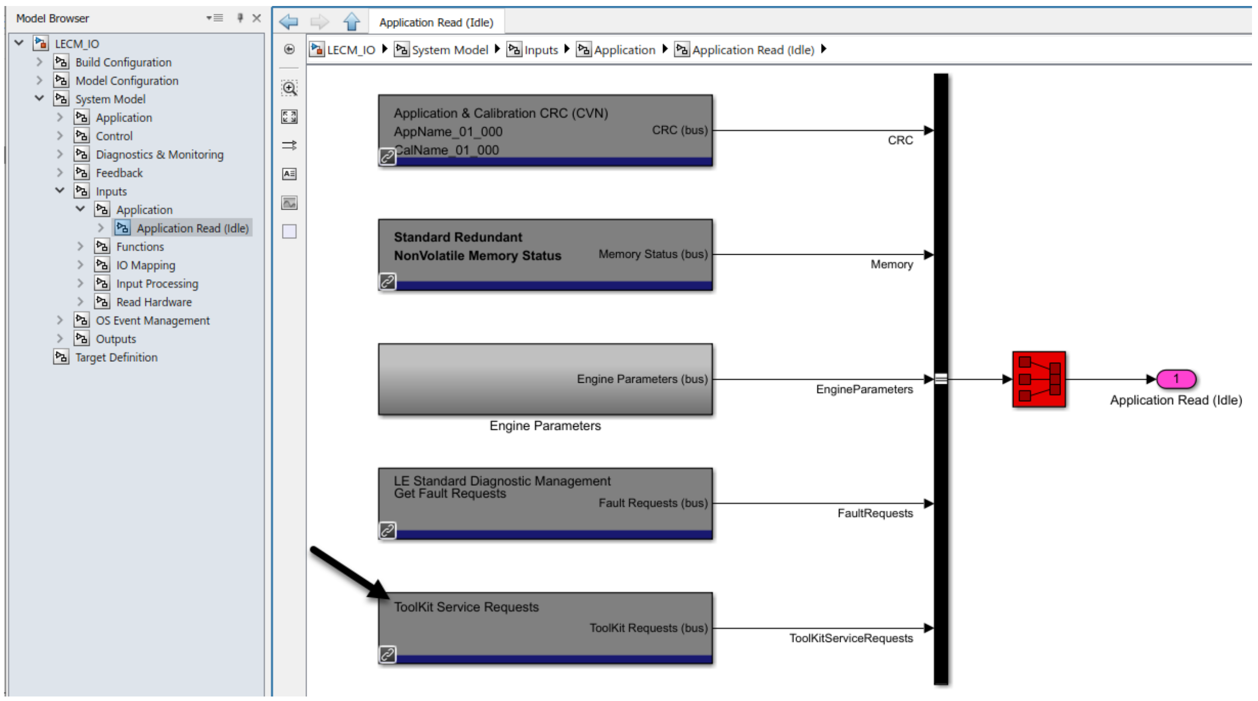


Figure 7-7. Application Read ToolKit Service Requests (Read ToolKit Requests)

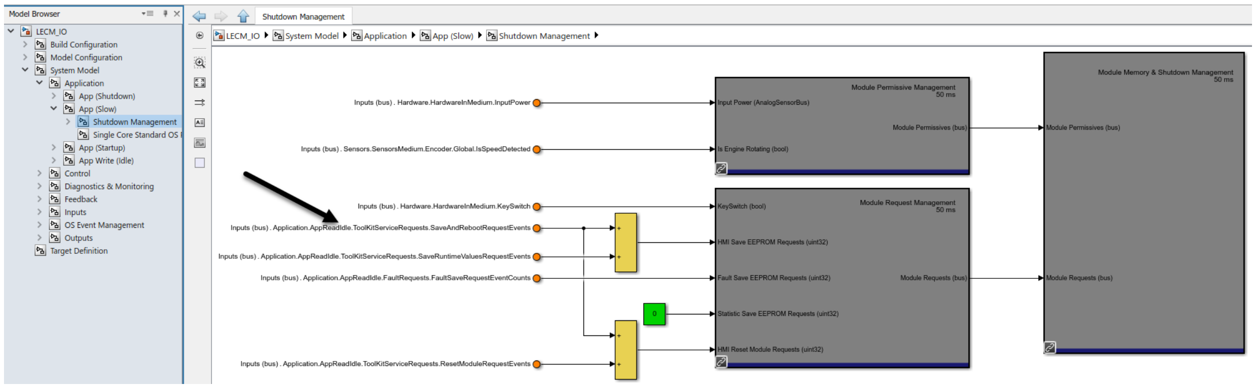

Model Browser
LECM_IO
Build Configuration
Model Configuration
System Model
Application
App (Shutdown)
App (Slow)
Shutdown Management
Single Core Standard OS
App (Startup)
App Write (Idle)
Control
Diagnostics & Monitoring
Feedback
Inputs
OS Event Management
Outputs
Target Definition
Shutdown Management
LECM_IO ▸ System Model ▸ Application ▸ App (Slow) ▸ Shutdown Management ▸
Inputs (bus) . Hardware.HardwareInMedium.InputPower
Inputs (bus) . Sensors.SensorsMedium.Encoder.Global.IsSpeedDetected
Inputs (bus) . Hardware.HardwareInMedium.KeySwitch
Inputs (bus) . Application.AppReadIdle.ToolKitServiceRequests.SaveAndRebootRequestEvents
Inputs (bus) . Application.AppReadIdle.ToolKitServiceRequests.SaveRuntimeValuesRequestEvents
Inputs (bus) . Application.AppReadIdle.FaultRequests.FaultSaveRequestEventCounts
Inputs (bus) . Application.AppReadIdle.ToolKitServiceRequests.ResetModuleRequestEvents
0
Module Permissive Management 50 ms
Input Power (AnalogSensorBus)
Is Engine Rotating (bool)
Module Permissives (bus)
Module Request Management 50 ms
KeySwitch (bool)
HMI Save EEPROM Requests (uint32)
Fault Save EEPROM Requests (uint32)
Statistic Save EEPROM Requests (uint32)
HMI Reset Module Requests (uint32)
Module Requests (bus)
Module Memory & Shutdown Management 50 ms
Module Permissives (bus)
Module Requests (bus)

Figure 7-8. Application Shutdown Management (Execute Requests)

<u>[Standard] IO Lock Request</u>

Similar to the example above, an IOLock request is also supported. This feature is built into ToolKit when it executes a settings load operation. If the IOLock feature is supported, the tool will automatically execute this request during a settings load. See similar implementation as described above.

<u>[Custom] Save Runtime Values Request</u>

Lastly, a third non-standard DAB function is supported that will execute a EEPROM save request independent of a permissive. This can be useful for saving EEPROM-based nonvolatile memory during any operational condition; however, this will not save flash-based calibrations, which requires a save/reboot operation to be executed.

XCP Reset Request

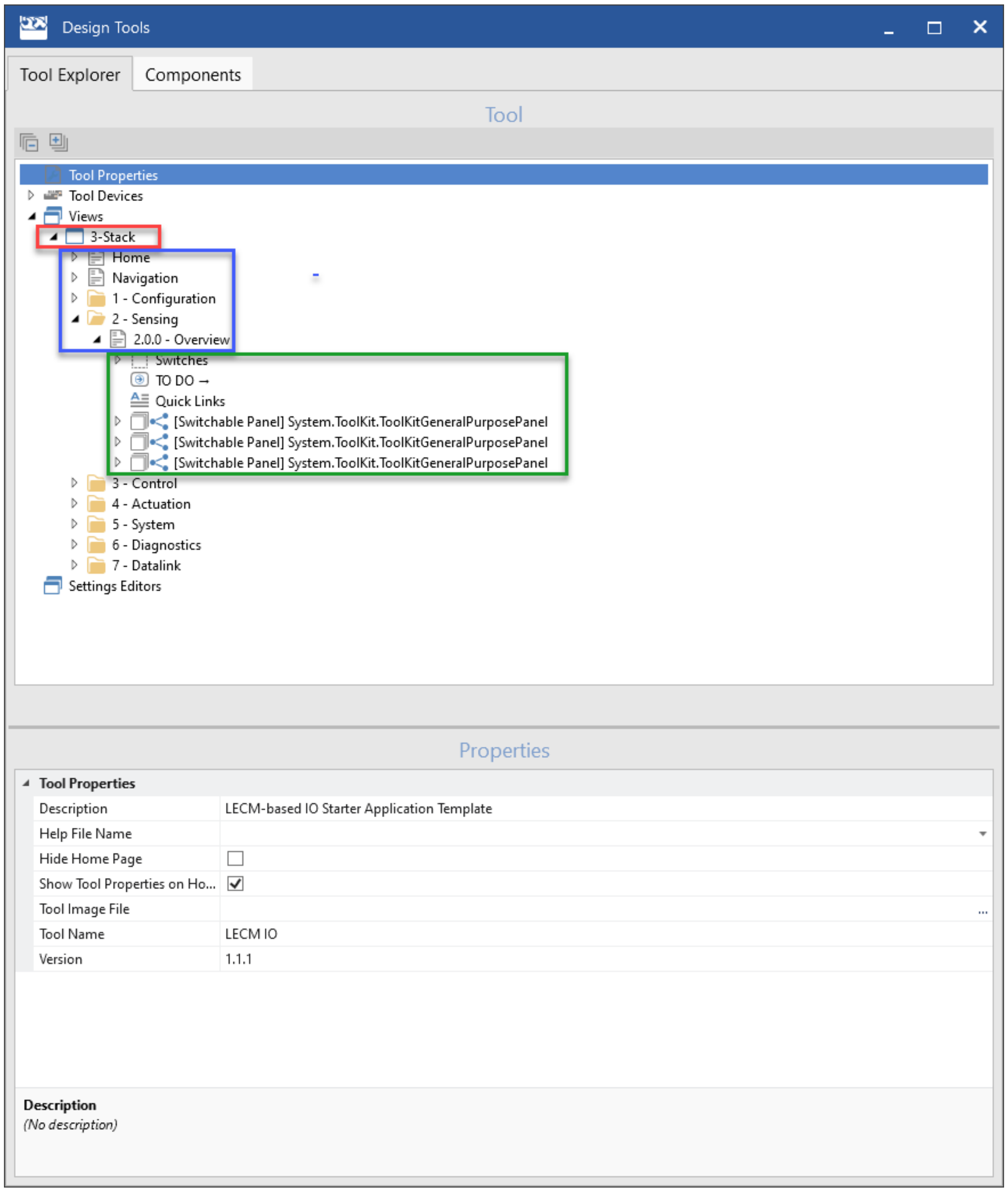
The ToolKit Standard Interface definition block implements the XCP Reset Request event. ToolKit will request this event after a firmware load is complete.

XCP Security

By default, packages do not come standard with XCP security defined. It's up to the developer to implement security if desired; however, if implemented, different levels can translate to different parameter access, and this can be integrated into ToolKit by specifying the levels and the name of the XCP security file. See MotoHawk XCP security documentation for more information on implementing a security algorithm.

ToolKit Structure

The ToolKit Service Tool design is divided into three core parts: Views, Pages, and Components. Each tool can support multiple views, each view hosting multiple pages, and each page hosting multiple different components or subpages. This is visualized in the Designer Window, where the red box outlines a View, the blue box indicates pages (some organized in folders), and the green box illustrates example components, which include views like gauges, charts, text, edit boxes, checkboxes, buttons, etc. Components are associated with runtime accessible parameters defined in the control (Calibrations, probes, Lookup Tables, Arrays, etc.).



Tool Properties

Description	LECM-based IO Starter Application Template
Help File Name	
Hide Home Page	☐
Show Tool Properties on Ho...	✓
Tool Image File	
Tool Name	LECM IO
Version	1.1.1

Description
(No description)

Figure 7-9. ToolKit Designer / Tool Hierarchy

In addition to views, each device for which the tool will connect, must be defined under the Tool Devices.

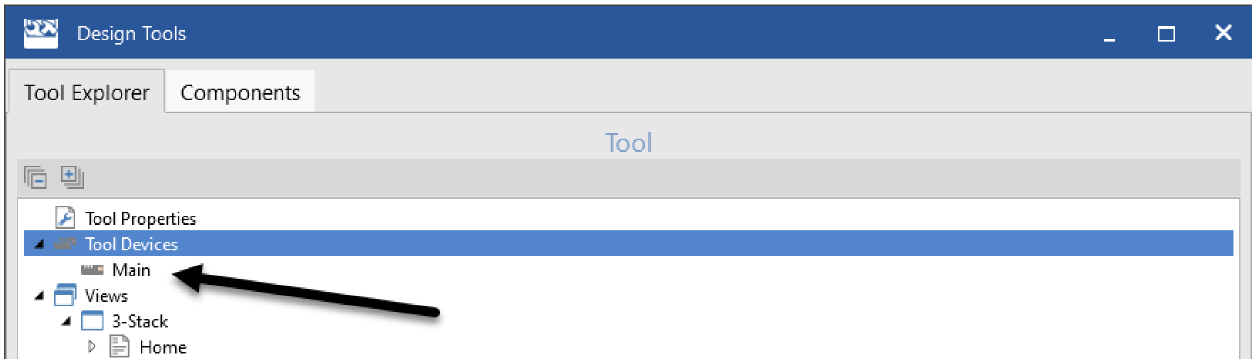


Figure 7-10. Tool Devices

IMPORTANT **Currently, ToolKit has a limitation that multiple devices must be connected on the same network device.**

Each view defined in the tool is associated with one or more SID files that defines the available parameter namespace. The SID can be updated by entering design mode, selecting the view, and changing the SID association for each device defined in the tool.

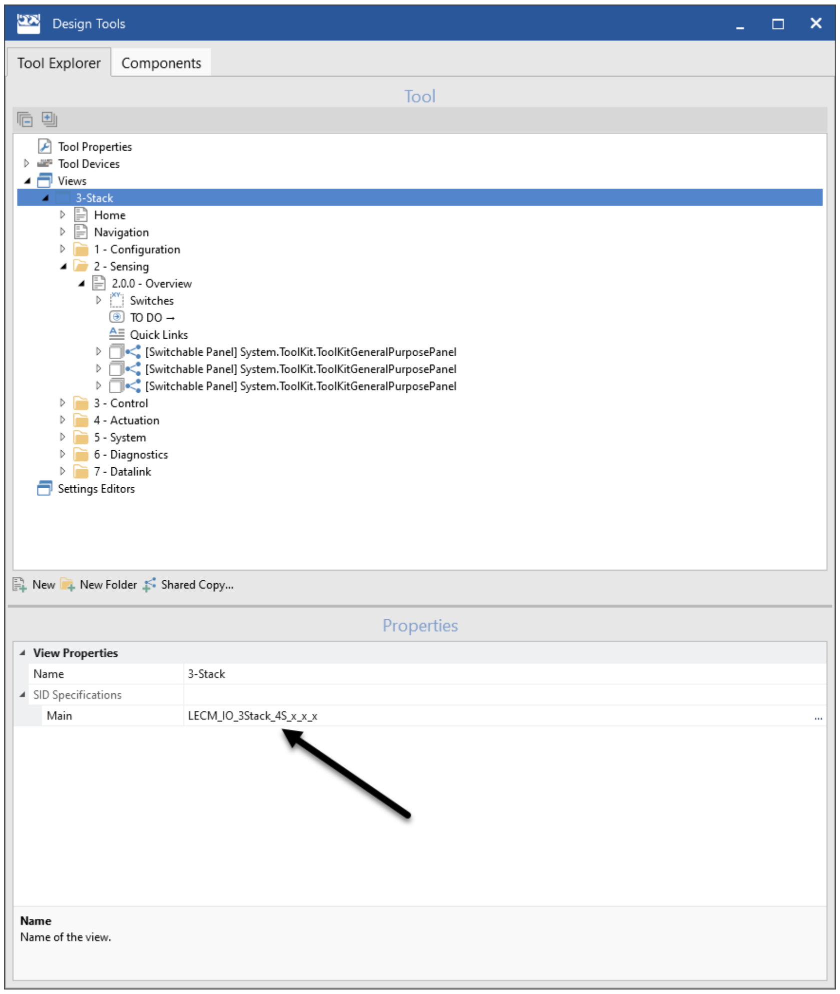
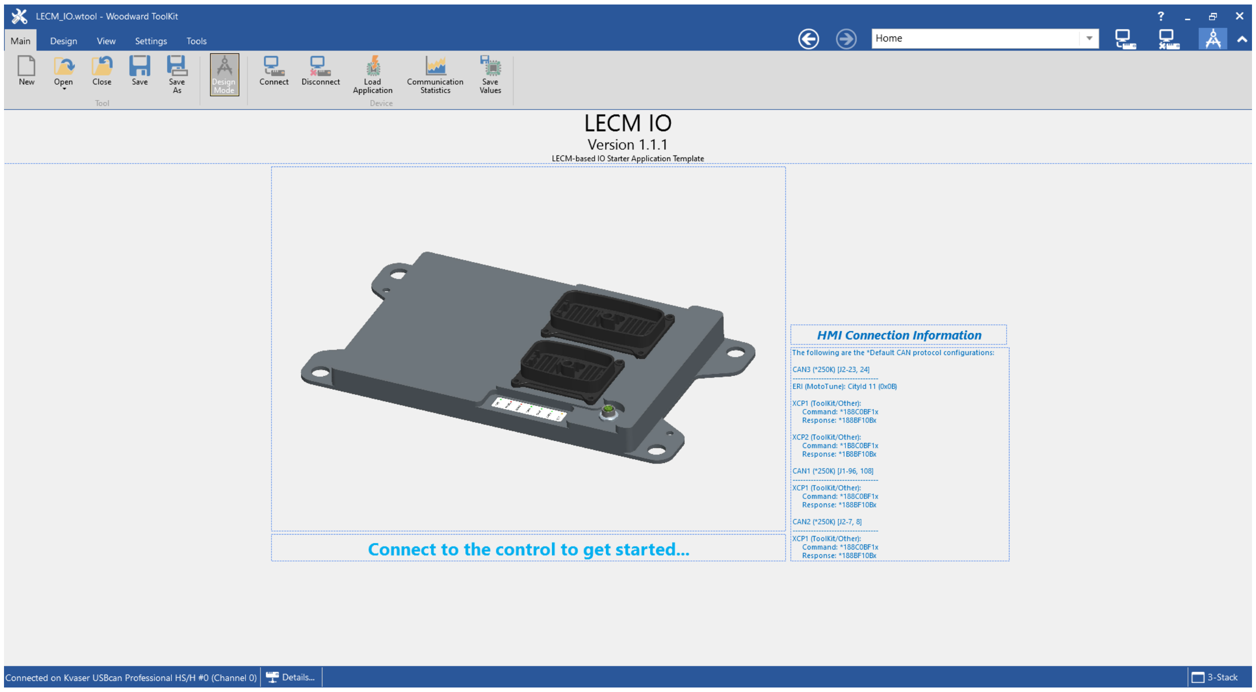


Figure 7-11. SID File Association

Each view will always contain a default HOME page. This page can only contain labels and image-based components as it is the default page made visible when the tool is first opened to the user. It's recommended to use this page as an introductory page for the tool with relevant information (e.g., connection information). This page can optionally be hidden if desired.



LECM IO
Version 1.1.1
LECM-based IO Starter Application Template

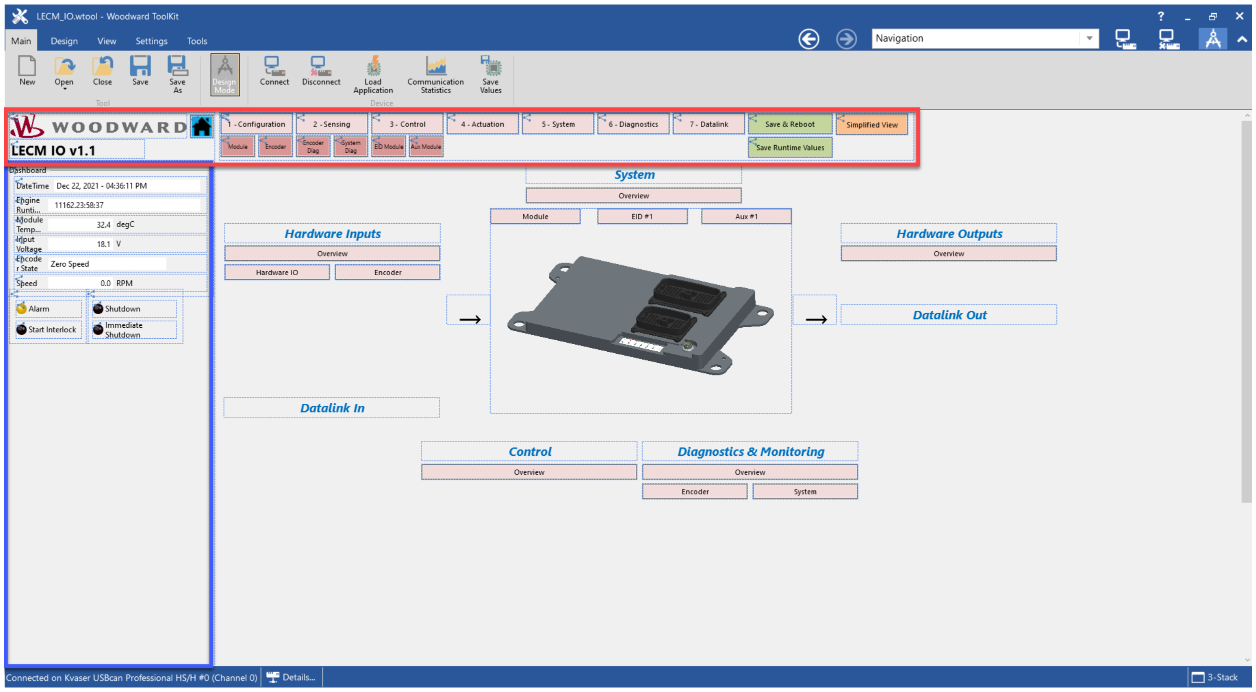
Connect to the control to get started...

HMI Connection Information

```
The following are the *Default CAN protocol configurations:

CAN3 (*250K) [J2-23, 24]
--------------------------------
ERI (MotoTune): CityId 11 (0x0B)

XCP1 (ToolKit/Other):
    Command: *188C0BF1x
    Response: *188BF10Bx

XCP2 (ToolKit/Other):
    Command: *1B8C0BF1x
    Response: *1B8BF10Bx

CAN1 (*250K) [J1-96, 108]
--------------------------------
XCP1 (ToolKit/Other):
    Command: *188C0BF1x
    Response: *188BF10Bx

CAN2 (*250K) [J2-7, 8]
--------------------------------
XCP1 (ToolKit/Other):
    Command: *188C0BF1x
    Response: *188BF10Bx
```

Figure 7-12. Example HOME Page

Each tool will contain a default set of pages for each respective view. For example, the LECM IO package is organized into folders of pages for Module Configuration, Inputs, Control, Outputs, System, Diagnostics, and Datalink. This organization is ultimately up to the application developer in order to provide a standard set of pages that interface with different software features defined in the firmware. The default pages utilize a "dashboard" (left docked pane – blue box) and a "navigation pane" (top docked pane – red box) page design. Each of these two components are shared across all ToolKit pages to provide a standard set of parameters and navigation buttons common to all pages. The remaining portion of the page is used for page-specific content.



Dashboard

DateTime	Dec 22, 2021 - 04:36:11 PM	
Engine Runti...	11162.23:58:37	
Module Temp...	32.4	degC
Input Voltage	18.1	V
Encoder State	Zero Speed	
Speed	0.0	RPM

Connected on Kvaser USBcan Professional HS/H #0 (Channel 0)

Figure 7-13. Standard Page Design

Chapter 8. Examples

Introduction

The following chapter provides a set of “how to” examples for developers to use as a reference. It’s important to note however, that there are many different approaches to code design and structure, and this chapter is only intended to provide references and general examples and not to prescribe a particular design strategy.

Input Examples

Adding Dynamically Mapped Sensors

This example demonstrates how to add a sensor that has a runtime configurable (dynamic) mapping to hardware IO. For the purposes of this exercise, we will add a 0-5V Ambient Air Pressure sensor that reports the current barometric pressure in units of kPa to the application.

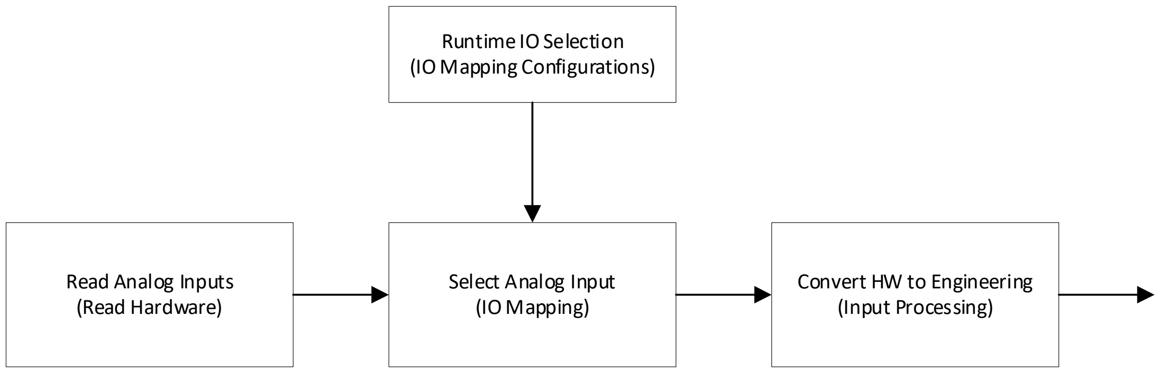
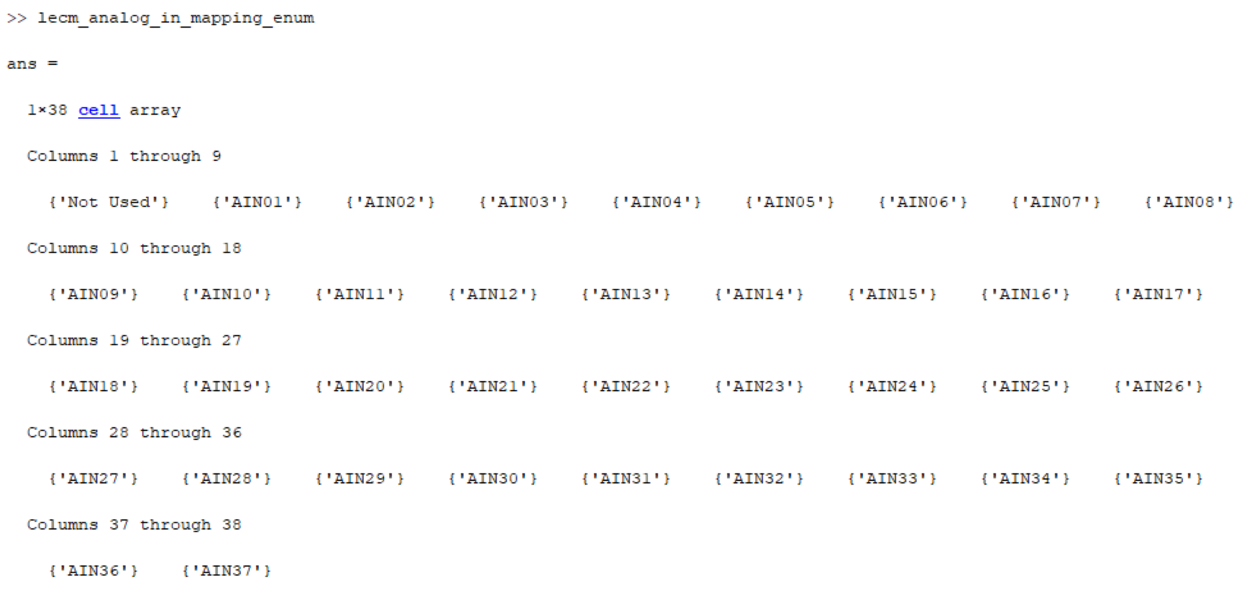


Figure 8-1. Sensor Processing Example

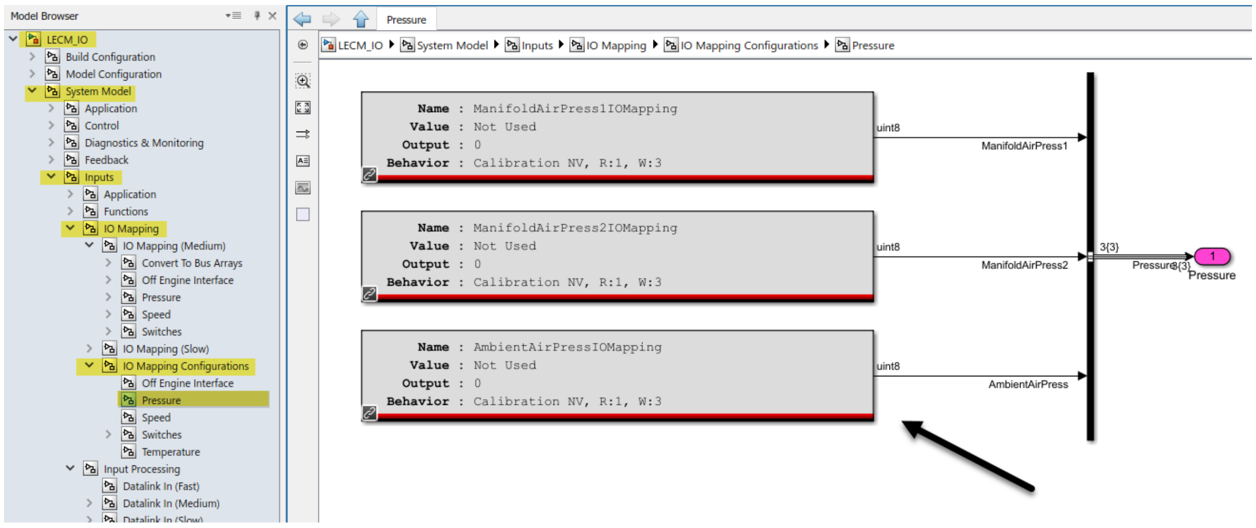
Analog Input	Sensor	Valid Sensor Range
AIN18 [0-5V] (runtime configurable)	Ambient Air Pressure	0.5 – 4.5 V

First navigate to the Inputs/IO Mapping/IO Mapping Configurations/Pressure subsystem. Inside, you should see an example bus structure with MotoHawk calibrations that contain the runtime mappings of pressure inputs. These calibrations use the standard lecm_analog_in_mapping_enum enumeration, which is defined as 0 = Not Used and 1...37 = AIN01...AIN37 respectively, thus allowing the respective sensor to be mapped to any analog input or none if not used. Change the AmbientAirPress default value to lecm_analog_in_mapping_enum(‘AIN18’) to default the calibration to using AIN18.



```
>> lecm_analog_in_mapping_enum

ans =

  1×38 cell array

  Columns 1 through 9

    {'Not Used'}    {'AIN01'}    {'AIN02'}    {'AIN03'}    {'AIN04'}    {'AIN05'}    {'AIN06'}    {'AIN07'}    {'AIN08'}

  Columns 10 through 18

    {'AIN09'}    {'AIN10'}    {'AIN11'}    {'AIN12'}    {'AIN13'}    {'AIN14'}    {'AIN15'}    {'AIN16'}    {'AIN17'}

  Columns 19 through 27

    {'AIN18'}    {'AIN19'}    {'AIN20'}    {'AIN21'}    {'AIN22'}    {'AIN23'}    {'AIN24'}    {'AIN25'}    {'AIN26'}

  Columns 28 through 36

    {'AIN27'}    {'AIN28'}    {'AIN29'}    {'AIN30'}    {'AIN31'}    {'AIN32'}    {'AIN33'}    {'AIN34'}    {'AIN35'}

  Columns 37 through 38

    {'AIN36'}    {'AIN37'}
```

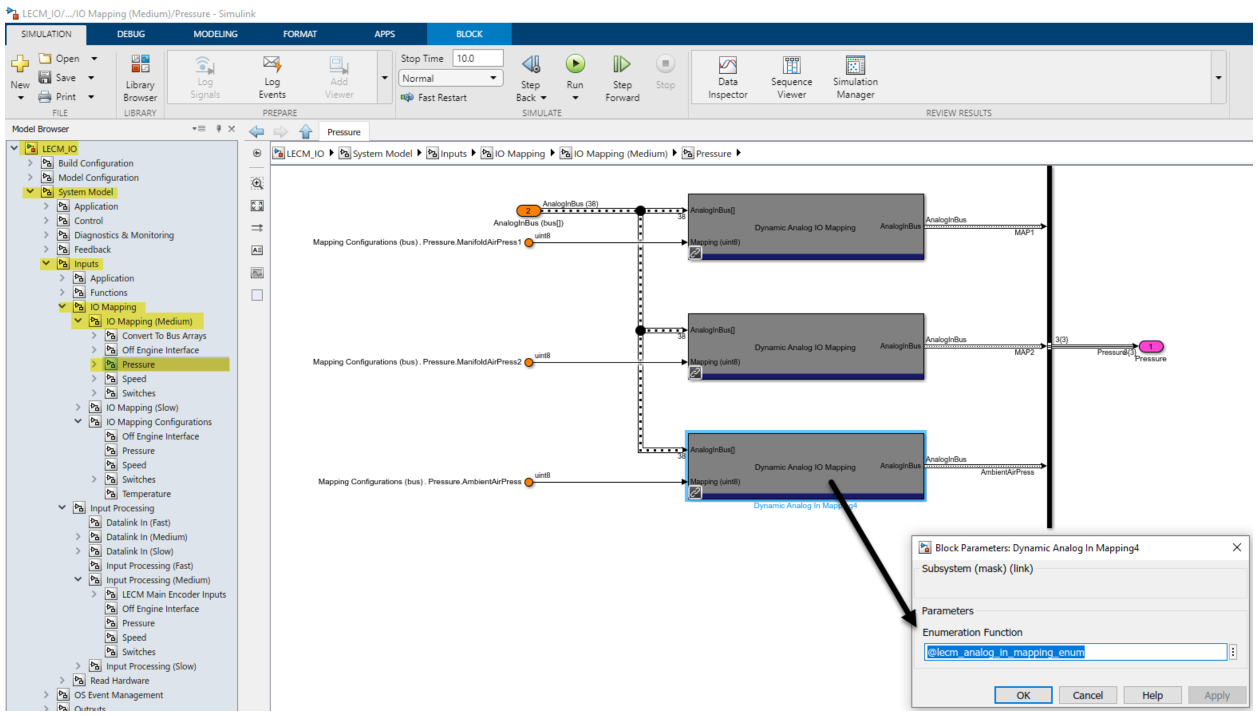
Figure 8-2. lecm_analog_in_mapping_enum



```
    Name : ManifoldAirPress1IOMapping
   Value : Not Used
  Output : 0
Behavior : Calibration NV, R:1, W:3

    Name : ManifoldAirPress2IOMapping
   Value : Not Used
  Output : 0
Behavior : Calibration NV, R:1, W:3

    Name : AmbientAirPressIOMapping
   Value : Not Used
  Output : 0
Behavior : Calibration NV, R:1, W:3
```

Figure 8-3. Ambient Air Pressure IO Mapping Configuration

Next, navigate to the Inputs/IO Mapping/IO Mapping (Medium)/Pressure subsystem. Inside, you should notice example platform blocks that implement the data selector of the respective analog input. This block accepts an AnalogInBus input array that matches the lecm_analog_in_mapping_enum and the desired mapping input. The mask of the block requires a handle to the analog input enumeration (@lecm_analog_in_mapping_enum) and will output the respective AnalogInBus object based on the input mapping.



Block Parameters: Dynamic Analog In Mapping4

Subsystem (mask) (link)

Parameters

Enumeration Function: @lecm_analog_in_mapping_enum

OK Cancel Help Apply

Figure 8-4. Dynamic IO Mapping Selector

Next navigate to the Inputs/Input Processing/Input Processing (Medium)/Pressure subsystem. This will execute the sensor logic in the medium rate execution context (10 ms in this example). Add a subsystem for Ambient Air Pressure Input Processing.

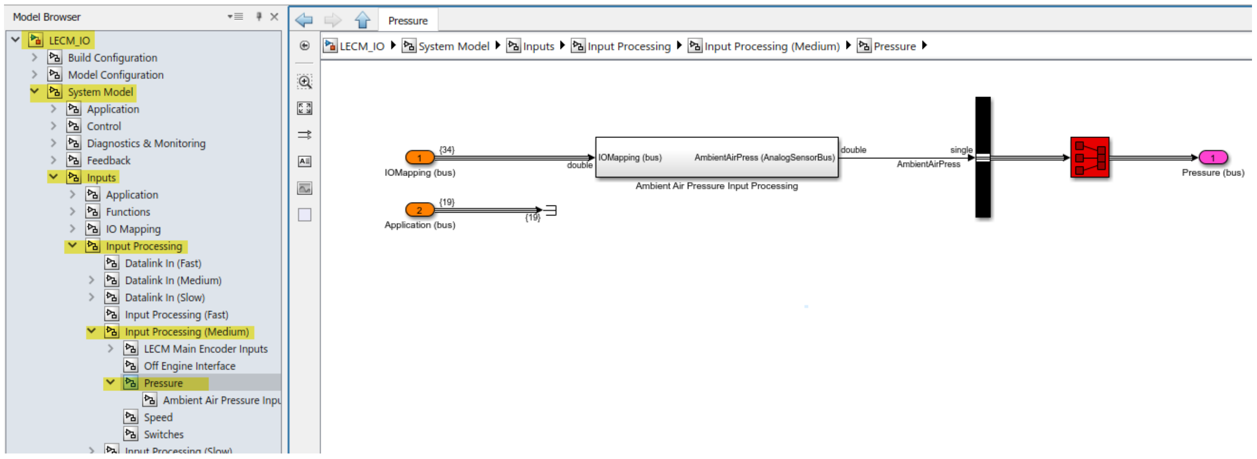


Figure 8-5. Inputs Pressure Sensor Subsystem

In this subsystem, we'll perform the input validation, convert from V to kPa, filter, and provide some simple defaulting logic. The logic will accept the AnalogInBus type and convert to an AnalogSensorBus type which contains two signals: FilteredValue and Status.

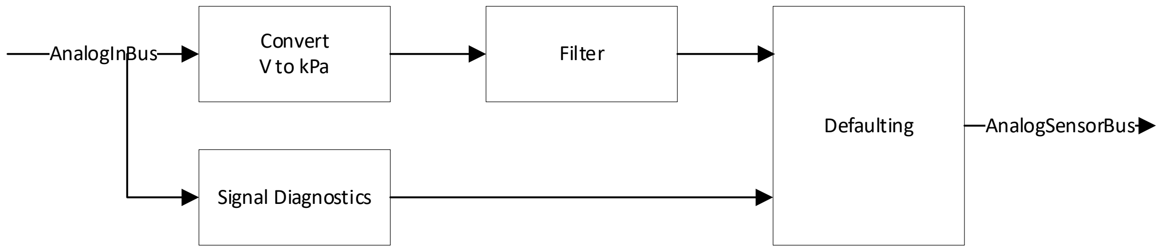


Figure 8-6. Ambient Air Pressure Input Processing Strategy

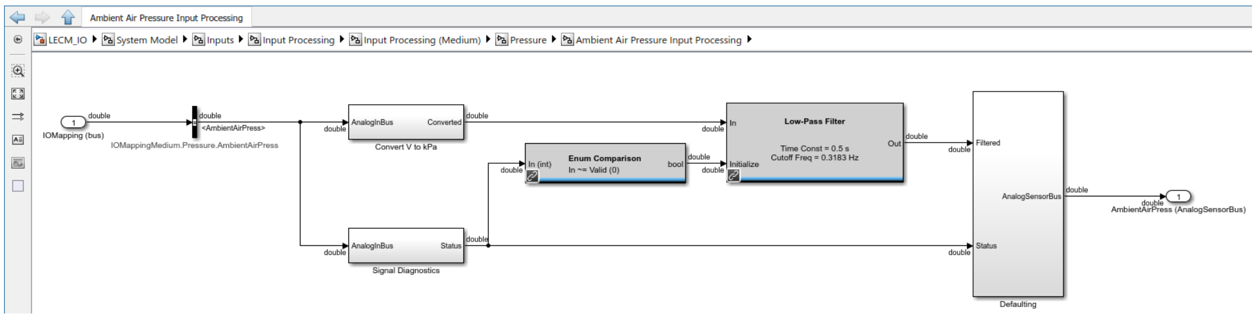

Ambient Air Pressure Input Processing
LECM_IO ▸ System Model ▸ Inputs ▸ Input Processing ▸ Input Processing (Medium) ▸ Pressure ▸ Ambient Air Pressure Input Processing ▸
IOMapping (bus)
<AmbientAirPress>
IOMappingMedium.Pressure.AmbientAirPress
AnalogInBus
Converted
Convert V to kPa
Enum Comparison
In ~= Valid (0)
Low-Pass Filter
Time Const = 0.5 s
Cutoff Freq = 0.3183 Hz
In
Initialize
Out
Filtered
AnalogSensorBus
Status
Signal Diagnostics
Defaulting
AmbientAirPress (AnalogSensorBus)

Figure 8-7. Ambient Air Pressure Input Processing Implementation

The Convert V to kPa utilizes subsystem gains and offsets the V to convert to kPa if the analog input mode is equal to 0 – 5V mode; otherwise, the sensor is considered disabled and outputs a zero value.

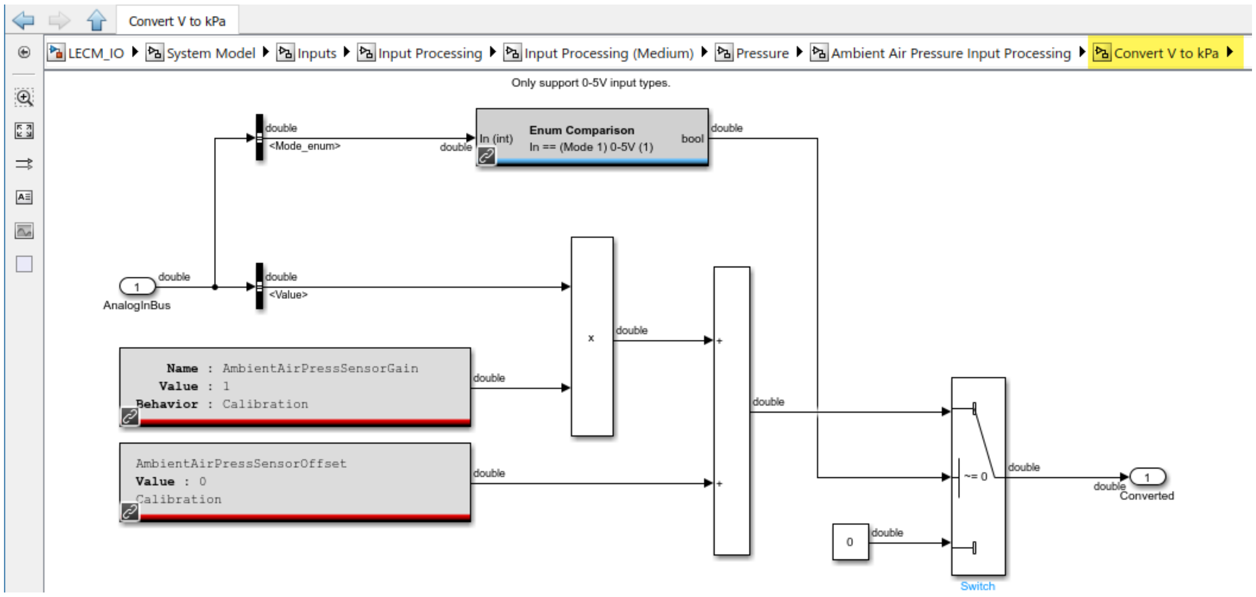

Convert V to kPa
LECM_IO ▸ System Model ▸ Inputs ▸ Input Processing ▸ Input Processing (Medium) ▸ Pressure ▸ Ambient Air Pressure Input Processing ▸ Convert V to kPa ▸
Only support 0-5V input types.
<Mode_enum>
Enum Comparison
In == (Mode 1) 0-5V (1)
AnalogInBus
<Value>
Name : AmbientAirPressSensorGain
Value : 1
Behavior : Calibration
AmbientAirPressSensorOffset
Value : 0
Calibration
~= 0
Switch
Converted

Figure 8-8. Ambient Air Pressure Input Processing Conversion

The Signal Diagnostics subsystem verifies the sensor input is valid and outputs a sensor measurement status using the lesp_measurement_status_enum. A value of Disabled (1) is used for the sensor status if the analog input mode is not equal to 0-5V. This occurs if the input is not mapped to a 0-5V input; otherwise, the nominal sensor analog input range of 0.5 to 4.5 V is checked and the corresponding SignalLow (2) and SignalHigh (3) statuses are reported if the value is outside this range respectively. Lastly, if all these conditions are not met, the sensor status will report Valid (0).

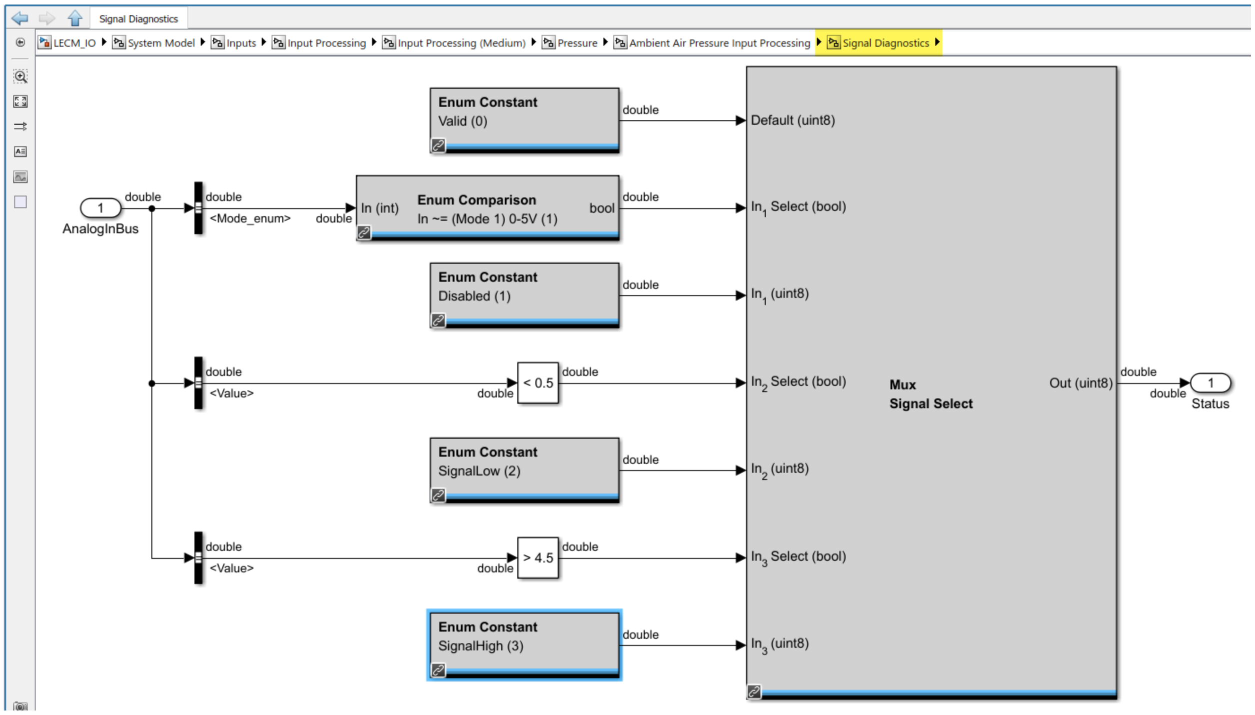

Signal Diagnostics
LECM_IO ▸ System Model ▸ Inputs ▸ Input Processing ▸ Input Processing (Medium) ▸ Pressure ▸ Ambient Air Pressure Input Processing ▸ Signal Diagnostics ▸
Enum Constant Valid (0)
double
Default (uint8)
1 AnalogInBus
double
<Mode_enum>
In (int)
Enum Comparison In ~= (Mode 1) 0-5V (1)
bool
In_1 Select (bool)
Enum Constant Disabled (1)
In_1 (uint8)
<Value>
< 0.5
In_2 Select (bool)
Mux Signal Select
Out (uint8)
1 Status
Enum Constant SignalLow (2)
In_2 (uint8)
> 4.5
In_3 Select (bool)
Enum Constant SignalHigh (3)
In_3 (uint8)

Figure 8-9. Ambient Air Pressure Input Processing Signal Diagnostics

Next, the input is filtered using an ASB Low Pass tau filter. The filter is configured to Initialize if the input is not valid to ensure invalid inputs reset the filter.

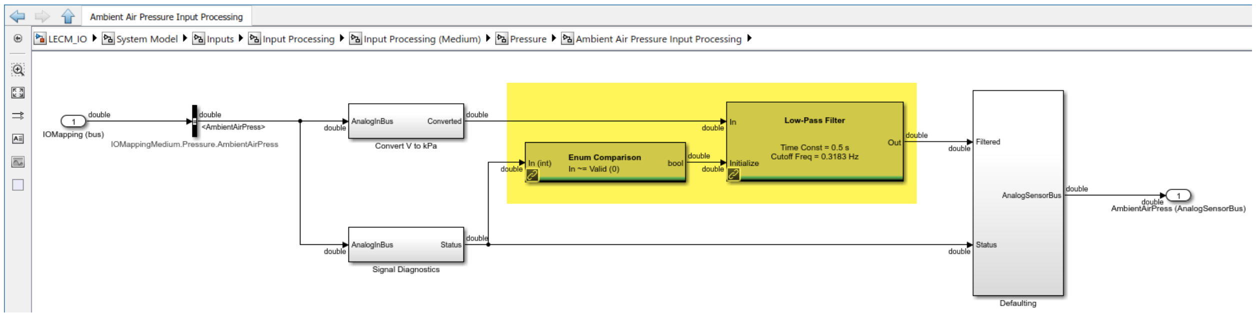


Figure 8-10. Ambient Air Pressure Input Processing Filter

Lastly, the Defaulting is used to report a default Ambient Air Pressure value if the status is not equal to Valid. This enables downstream logic to use a configured default and ignore the status if applicable. Finally, these signals are combined into an AnalogSensorBus type to be used in downstream logic.

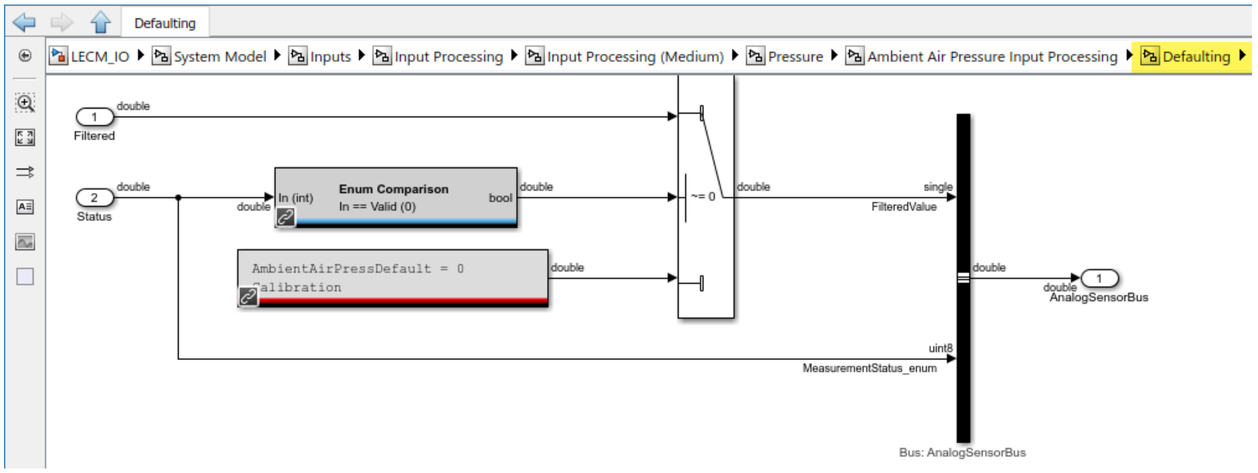

Defaulting
LECM_IO ▸ System Model ▸ Inputs ▸ Input Processing ▸ Input Processing (Medium) ▸ Pressure ▸ Ambient Air Pressure Input Processing ▸ Defaulting
1 Filtered
2 Status
Enum Comparison
In == Valid (0)
AmbientAirPressDefault = 0
Calibration
~= 0
FilteredValue
MeasurementStatus_enum
1 AnalogSensorBus
Bus: AnalogSensorBus

Figure 8-11. Ambient Air Pressure Input Processing Defaulting

IMPORTANT

Note: This logic is a relatively simple example to demonstrate the concept/process of adding a sensor input to the application. A more comprehensive input conversion strategy may be employed (e.g., ratiometric sensing options, advanced filters, hysteretic signal diagnostics, more complex defaulting logic, etc.). In addition, if dynamic IO mapping is not required, the AnalogInBus can be used directly from the Read Hardware bus structure and the IO Mapping logic can be skipped.

It's also recommended, for common sensor types, to create a local library of blocks that are parameterized to allow for re-use.

Adding Aux RTCDC Cycle Data Interfacing

This example demonstrates how to read LECM Aux RTCDC cylinder cycle data into the application. As previously discussed in Chapter 4, Aux RTCDC cylinder cycle data interfaces report cycle-based in-cylinder combustion metrics asynchronously. The asynchronous reporting therefore requires the current operational speed to be used to determine an appropriate message timeout.

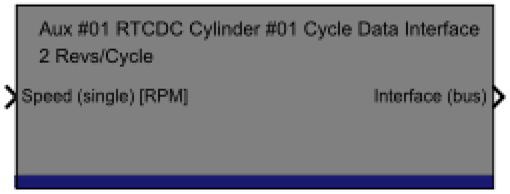


Figure 8-12. Aux RTCDC Cylinder Cycle Data Interface

In this example, let's assume our maximum engine speed is 1500 RPM (4-stroke). This equates to a maximum engine cycle time of 80 milliseconds.

$$Max\ Cycle\ Time\ \left[\frac{ms}{cycle}\right] = \frac{720\ \left[\frac{deg}{cycle}\right]}{\left(\frac{360\ deg}{1\ rev}\ x\ \frac{1500\ rev}{1\ min}\ x\ \frac{1\ min}{60000\ ms}\right)\left[\frac{deg}{ms}\right]} = 80\ \left[\frac{ms}{cycle}\right]$$

A maximum cycle time of 80 ms requires the interface to be located in a rate group faster than 80 ms. In this example, we'll add to the Datalink Slow processing rate group (50 ms); however, note the current engine speed is required as an input for proper message timeout detection. Navigate to the Input

Processing subsystem and add the Sensors Medium bus as an input to the Datalink In Slow subsystem in order to read the current speed from the encoder inputs.

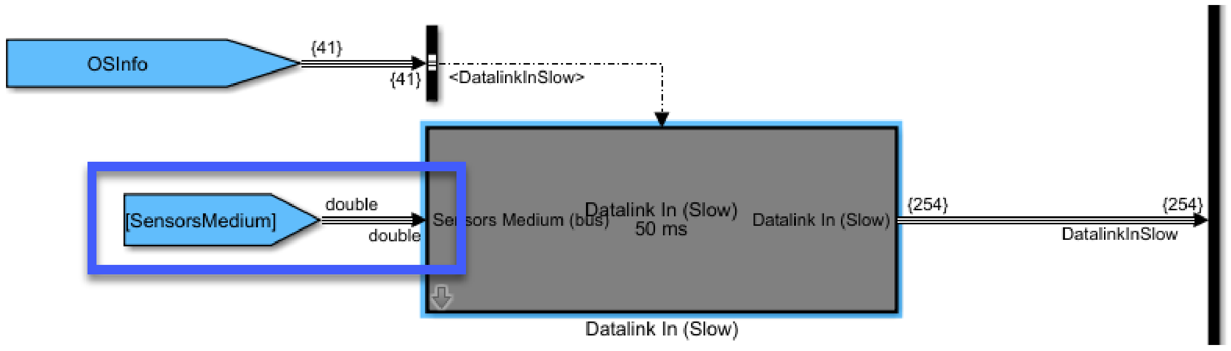


Figure 8-13. Datalink In Slow Subsystem

Next, inside of the Datalink In Slow subsystem, add a subsystem for RTCDC Cycle Data and use a bus selector to select the AverageSpeed signal from the encoder inputs.

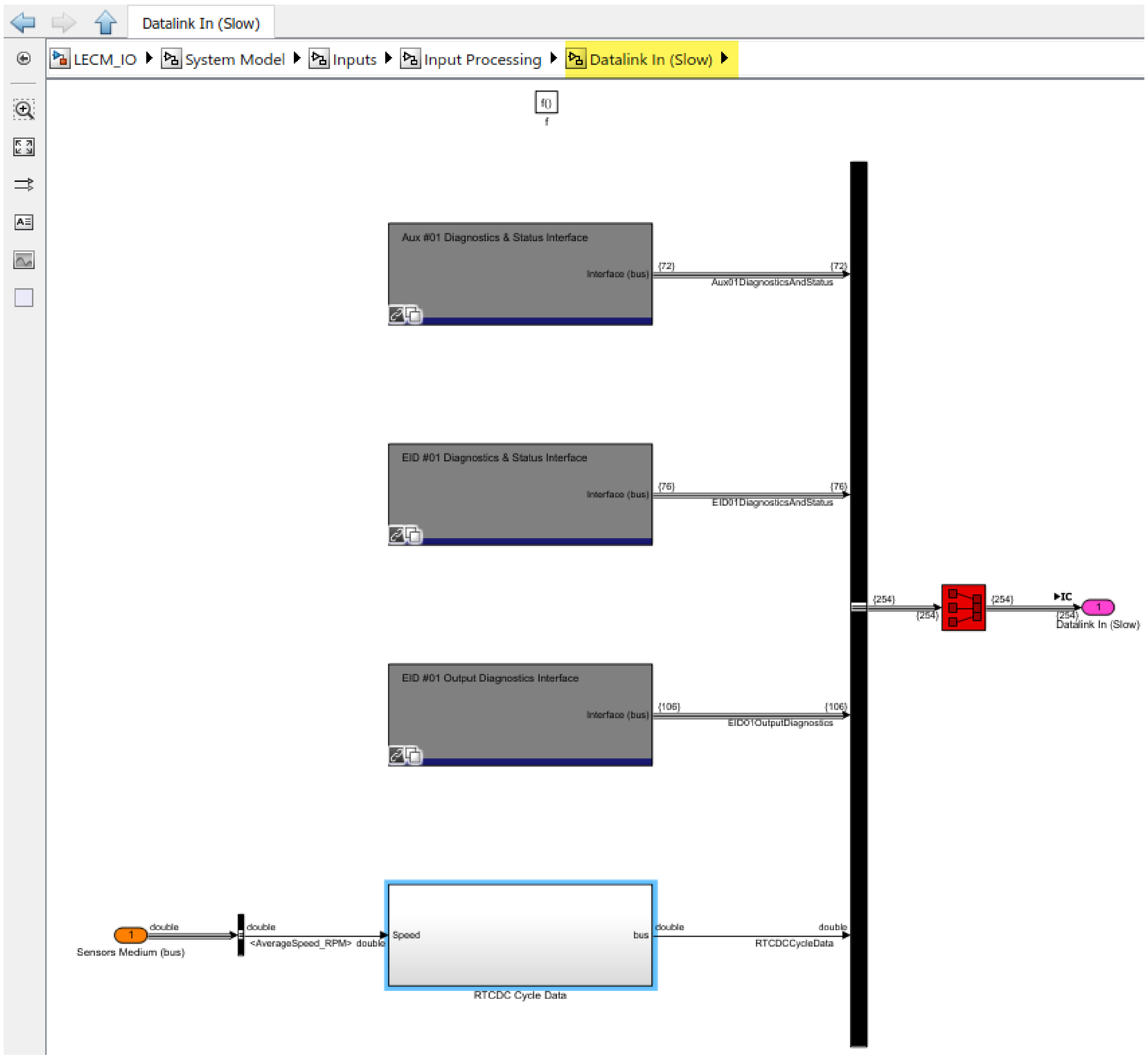


Figure 8-14. RTCDC Cycle Data Subsystem

Inside of the RTCDC Cycle Data subsystem, add the RTCDC Cycle Data interfaces for the cylinders of interest and locate these interfaces on a bus creator. The RTCDC data can then be used downstream to implement control logic and interface diagnostics.

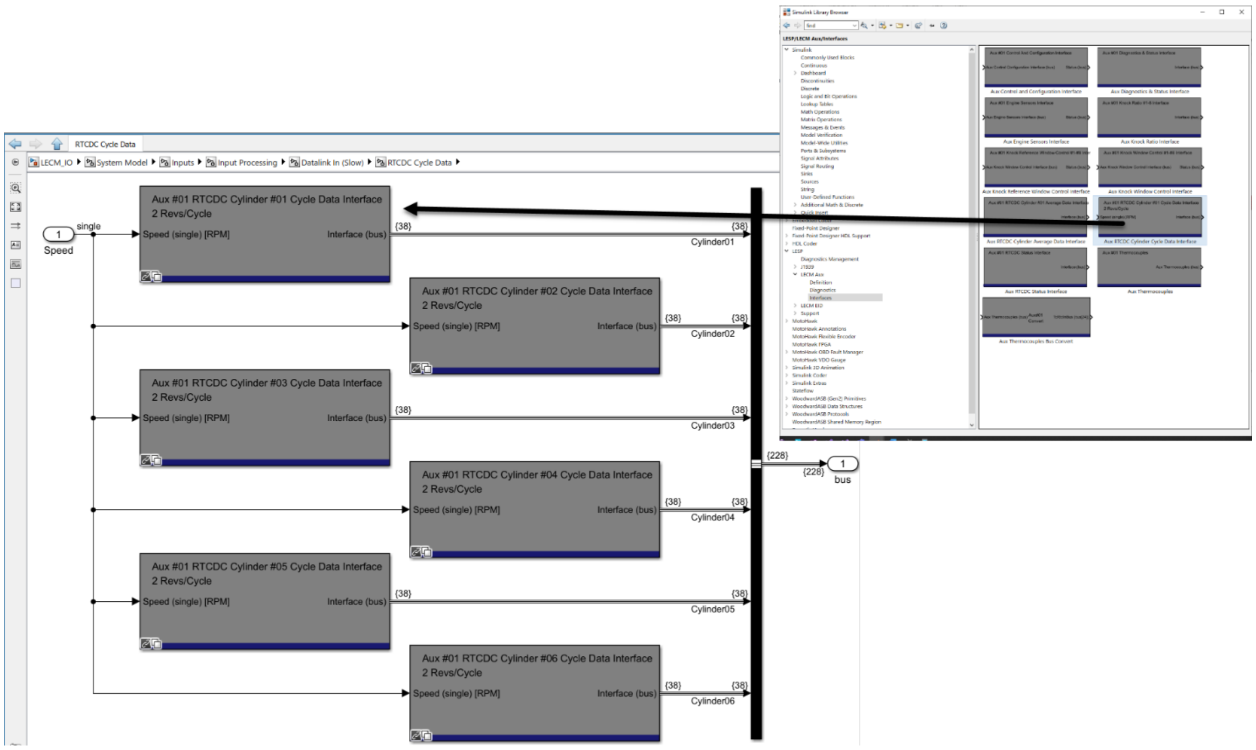
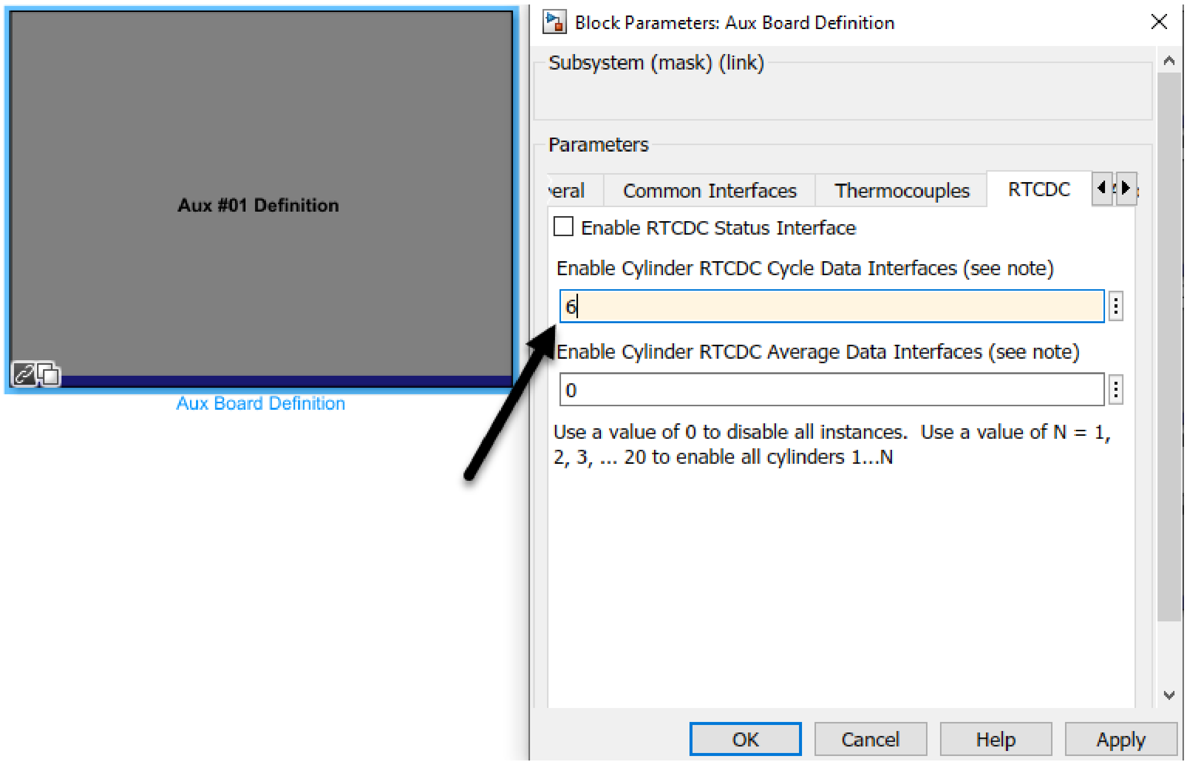

RTCDC Cycle Data
LECM_IO ▸ System Model ▸ Inputs ▸ Input Processing ▸ Datalink In (Slow) ▸ RTCDC Cycle Data
1
Speed
single
Aux #01 RTCDC Cylinder #01 Cycle Data Interface 2 Revs/Cycle
Aux #01 RTCDC Cylinder #02 Cycle Data Interface 2 Revs/Cycle
Aux #01 RTCDC Cylinder #03 Cycle Data Interface 2 Revs/Cycle
Aux #01 RTCDC Cylinder #04 Cycle Data Interface 2 Revs/Cycle
Aux #01 RTCDC Cylinder #05 Cycle Data Interface 2 Revs/Cycle
Aux #01 RTCDC Cylinder #06 Cycle Data Interface 2 Revs/Cycle
Speed (single) [RPM]
Interface (bus)
{38}
Cylinder01
Cylinder02
Cylinder03
Cylinder04
Cylinder05
Cylinder06
{228}
1
bus
Simulink Library Browser
LESP/LECM Aux/Interfaces
Aux Control and Configuration Interface
Aux Diagnostics & Status Interface
Aux Engine Sensors Interface
Aux Knock Ratio Interface
Aux Knock Reference Window Control Interface
Aux Knock Window Control Interface
Aux RTCDC Cylinder Average Data Interface
Aux RTCDC Cylinder Cycle Data Interface
Aux RTCDC Status Interface
Aux Thermocouples
Aux Thermocouples Bus Convert

Figure 8-15. RTCDC Cycle Data Interfaces

After adding the interfaces, be sure to enable them by default under the Aux Definition block.



- [ ] Enable RTCDC Status Interface

Enable Cylinder RTCDC Cycle Data Interfaces (see note): 6

Enable Cylinder RTCDC Average Data Interfaces (see note): 0

Use a value of 0 to disable all instances. Use a value of N = 1, 2, 3, ... 20 to enable all cylinders 1...N

Figure 8-16. Aux Board Definition RTCDC Cycle Data Interface Enables

Adding EID Injection Command Interfacing

This example demonstrates how to add EID Injection command interfacing to the application. In referencing the EID Injection Manual (B35175), recall the EID Multi-pulse injection interfaces are divided into two interfaces: Pulse control interface and Cylinder Trim interfaces. The pulse control interface provides the commands for a base pulse train of up to 5 pulses (base pulse timings, durations, and pulse control mode). This base pulse train is relative to each cylinder's TDC that is mapped to this interface. The trim interface enables individual cylinder adjustments of states, timing offsets, and magnitude trims. These interfaces are accessible under the LECM EID/Interfaces Simulink library browser.

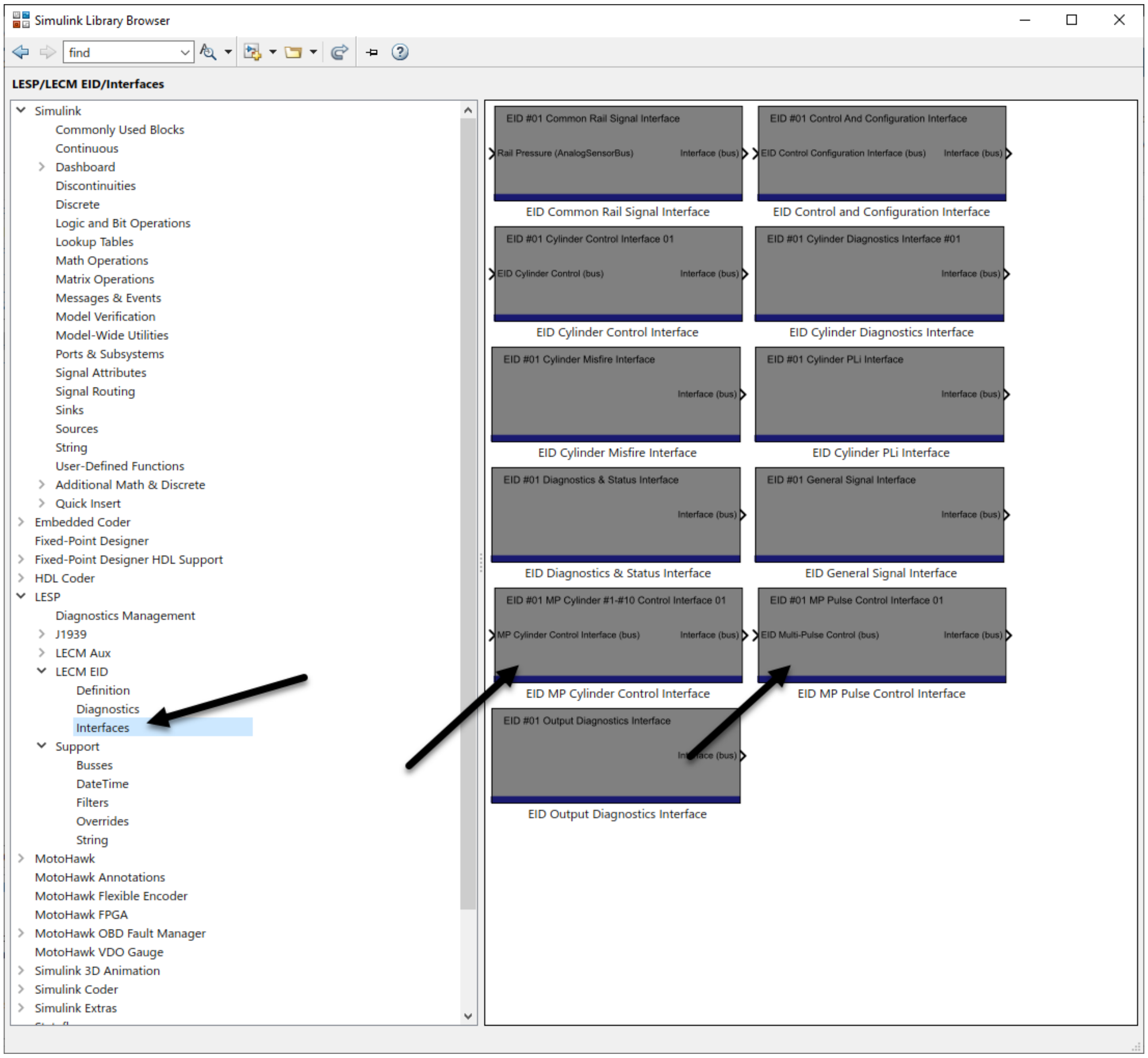

Simulink Library Browser
find
LESP/LECM EID/Interfaces
Simulink
Commonly Used Blocks
Continuous
Dashboard
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Matrix Operations
Messages & Events
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
String
User-Defined Functions
Additional Math & Discrete
Quick Insert
Embedded Coder
Fixed-Point Designer
Fixed-Point Designer HDL Support
HDL Coder
LESP
Diagnostics Management
J1939
LECM Aux
LECM EID
Definition
Diagnostics
Interfaces
Support
Busses
DateTime
Filters
Overrides
String
MotoHawk
MotoHawk Annotations
MotoHawk Flexible Encoder
MotoHawk FPGA
MotoHawk OBD Fault Manager
MotoHawk VDO Gauge
Simulink 3D Animation
Simulink Coder
Simulink Extras
EID #01 Common Rail Signal Interface
Rail Pressure (AnalogSensorBus)
Interface (bus)
EID Common Rail Signal Interface
EID #01 Control And Configuration Interface
EID Control Configuration Interface (bus)
Interface (bus)
EID Control and Configuration Interface
EID #01 Cylinder Control Interface 01
EID Cylinder Control (bus)
Interface (bus)
EID Cylinder Control Interface
EID #01 Cylinder Diagnostics Interface #01
Interface (bus)
EID Cylinder Diagnostics Interface
EID #01 Cylinder Misfire Interface
Interface (bus)
EID Cylinder Misfire Interface
EID #01 Cylinder PLi Interface
Interface (bus)
EID Cylinder PLi Interface
EID #01 Diagnostics & Status Interface
Interface (bus)
EID Diagnostics & Status Interface
EID #01 General Signal Interface
Interface (bus)
EID General Signal Interface
EID #01 MP Cylinder #1-#10 Control Interface 01
MP Cylinder Control Interface (bus)
Interface (bus)
EID MP Cylinder Control Interface
EID #01 MP Pulse Control Interface 01
EID Multi-Pulse Control (bus)
Interface (bus)
EID MP Pulse Control Interface
EID #01 Output Diagnostics Interface
Interface (bus)
EID Output Diagnostics Interface

Figure 8-17. LECM EID Interfaces Simulink Browser

Before adding the interfaces, use the bus create blocks for these respective interfaces and create the control logic in a Control/Control Fast/Fuel Control subsystem. The example below illustrates the interfaces with stubs for inputs.

- **Pulse Control Interface**
  - Mode utilizes the lecm_eid_mp_pulse_control_mode_enum. This mode defines the mode of operation for the interface and the units of the Magnitudes inport.
    - Disabled, Volume Control, or Duration Control.

- Timing Range Resolution Select input uses the lecm_eid_mp_timing_range_resolution_enum which specifies the allowable range and resolution of the Timing Offsets inport.
  - LowRange-HighResolution, MediumRange-MediumResolution, HighRange-LowResolution
- Slot Select is used to select the desired profile slot to be used.
- Pulse Enables is a 1x5 boolean array for state control of each pulse #1...#5.
- Timing Offsets is a fixed-point value representing each pulse #1...#5 relative timing to each cylinder’s TDC. Note: The multiplication factor (16, 8, 4) depends on the timing range resolution selection respectively.
- Magnitudes is used to control the volume or duration command based on the mode of operation.

- **Cylinder Trim Interface**
  - Mode must match the corresponding pulse control interface in order to properly interpret the Magnitude Trims inport.
  - Cylinder Enables allow individual cylinder state control.
  - Pulse SOI Apply Flags are used to indicate the pulses for which the SOI Trims will apply.
  - The Cylinder SOI Trims input is used to adjust the timing of each applied pulse. This resolution is always fixed to 1/8.
  - Cylinder Magnitude Trims is used to adjust the magnitude of the Main pulse (pulse #3) only for the respective cylinder.

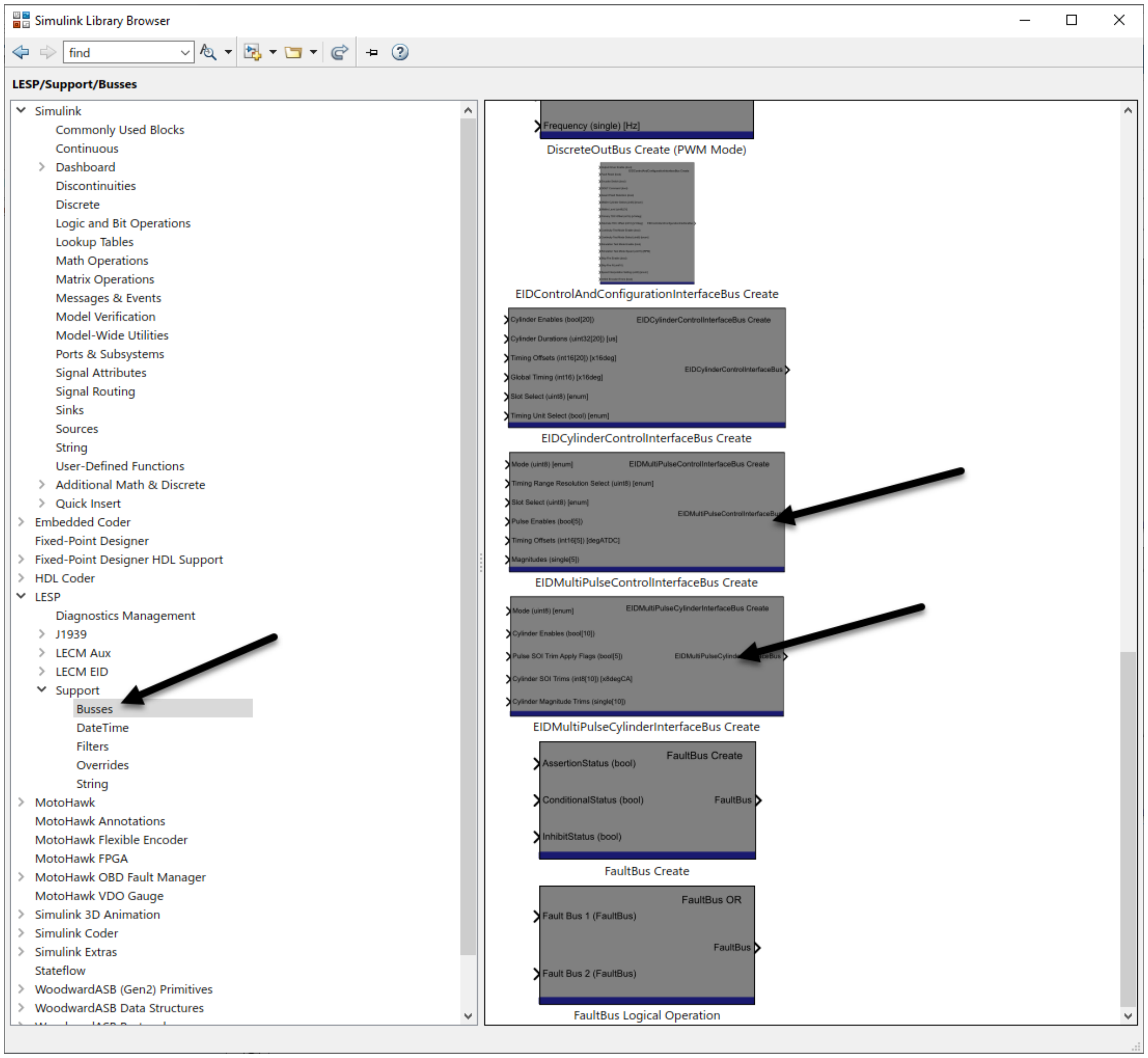

Simulink Library Browser
find
LESP/Support/Busses
Simulink
Commonly Used Blocks
Continuous
Dashboard
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Matrix Operations
Messages & Events
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
String
User-Defined Functions
Additional Math & Discrete
Quick Insert
Embedded Coder
Fixed-Point Designer
Fixed-Point Designer HDL Support
HDL Coder
LESP
Diagnostics Management
J1939
LECM Aux
LECM EID
Support
Busses
DateTime
Filters
Overrides
String
MotoHawk
MotoHawk Annotations
MotoHawk Flexible Encoder
MotoHawk FPGA
MotoHawk OBD Fault Manager
MotoHawk VDO Gauge
Simulink 3D Animation
Simulink Coder
Simulink Extras
Stateflow
WoodwardASB (Gen2) Primitives
WoodwardASB Data Structures
Frequency (single) [Hz]
DiscreteOutBus Create (PWM Mode)
EIDControlAndConfigurationInterfaceBus Create
Cylinder Enables (bool[20])
Cylinder Durations (uint32[20]) [us]
Timing Offsets (int16[20]) [x16deg]
Global Timing (int16) [x16deg]
Slot Select (uint8) [enum]
Timing Unit Select (bool) [enum]
EIDCylinderControlInterfaceBus
EIDCylinderControlInterfaceBus Create
Mode (uint8) [enum]
Timing Range Resolution Select (uint8) [enum]
Slot Select (uint8) [enum]
Pulse Enables (bool[5])
Timing Offsets (int16[5]) [degATDC]
Magnitudes (single[5])
EIDMultiPulseControlInterfaceBus
EIDMultiPulseControlInterfaceBus Create
Mode (uint8) [enum]
Cylinder Enables (bool[10])
Pulse SOI Trim Apply Flags (bool[5])
Cylinder SOI Trims (int8[10]) [x8degCA]
Cylinder Magnitude Trims (single[10])
EIDMultiPulseCylinderInterfaceBus Create
AssertionStatus (bool)
ConditionalStatus (bool)
InhibitStatus (bool)
FaultBus
FaultBus Create
Fault Bus 1 (FaultBus)
Fault Bus 2 (FaultBus)
FaultBus OR
FaultBus Logical Operation

Figure 8-18. LECM EID Interface Bus Creators

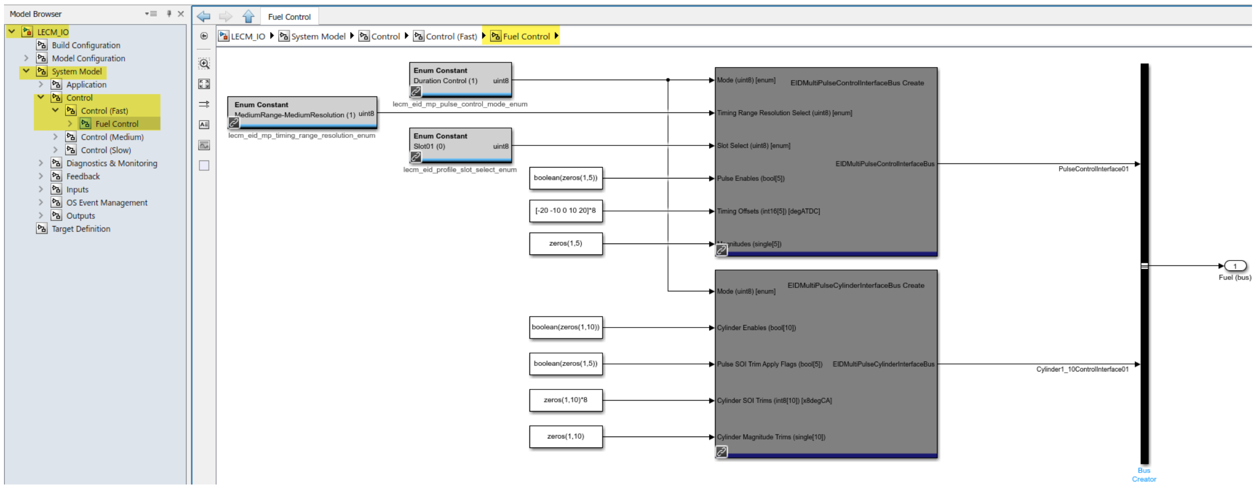

Model Browser
LECM_IO
Build Configuration
Model Configuration
System Model
Application
Control
Control (Fast)
Fuel Control
Control (Medium)
Control (Slow)
Diagnostics & Monitoring
Feedback
Inputs
OS Event Management
Outputs
Target Definition
Enum Constant Duration Control (1) uint8
lecm_eid_mp_pulse_control_mode_enum
Enum Constant MediumRange-MediumResolution (1) uint8
lecm_eid_mp_timing_range_resolution_enum
Enum Constant Slot01 (0) uint8
lecm_eid_profile_slot_select_enum
boolean(zeros(1,5))
[-20 -10 0 10 20]*8
zeros(1,5)
EIDMultiPulseControlInterfaceBus Create
EIDMultiPulseControlInterfaceBus
PulseControlInterface01
EIDMultiPulseCylinderInterfaceBus Create
EIDMultiPulseCylinderInterfaceBus
Cylinder1_10ControlInterface01
boolean(zeros(1,10))
zeros(1,10)*8
zeros(1,10)
Fuel (bus)
Bus Creator

Figure 8-19. Fuel Control Subsystem

Next, navigate to Outputs/Outputs Fast and add the EID Multi-Pulse Pulse/Cylinder control interfaces. Subsequently, interface fault monitors can be added in the Diagnostics & Monitoring to report if the interface communications fail. See Adding Faults section below for more details on how to add a new fault.

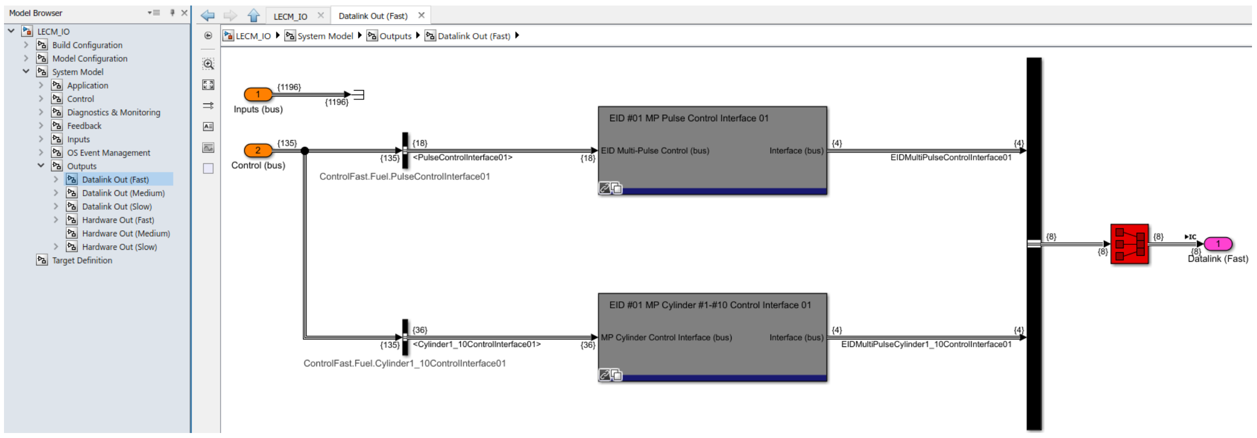
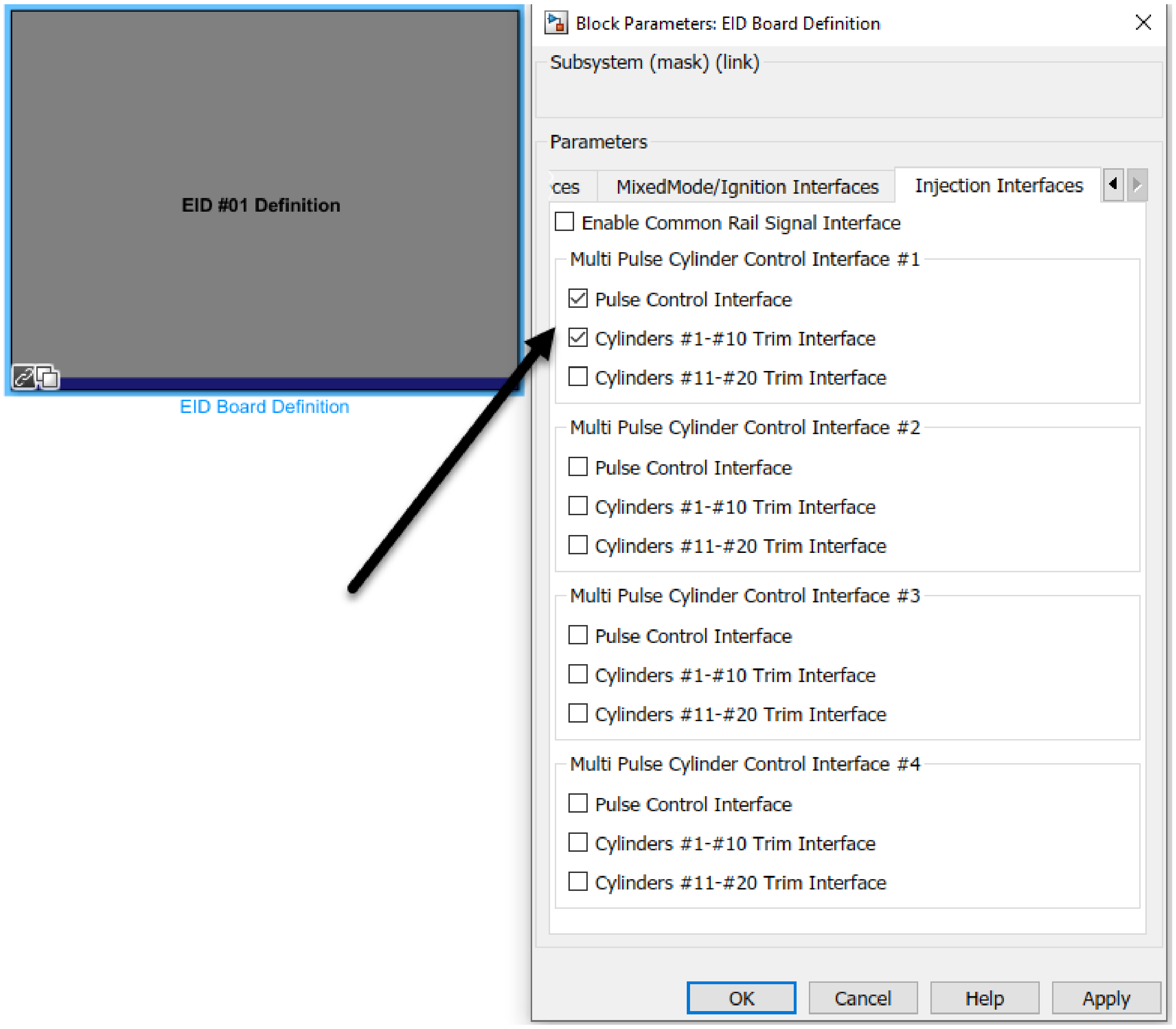

Model Browser
LECM_IO
Datalink Out (Fast)
System Model
Outputs
Datalink Out (Medium)
Datalink Out (Slow)
Hardware Out (Fast)
Hardware Out (Medium)
Hardware Out (Slow)
Inputs (bus)
Control (bus)
ControlFast.Fuel.PulseControlInterface01
ControlFast.Fuel.Cylinder1_10ControlInterface01
EID #01 MP Pulse Control Interface 01
EID Multi-Pulse Control (bus)
Interface (bus)
EIDMultiPulseControlInterface01
EID #01 MP Cylinder #1-#10 Control Interface 01
MP Cylinder Control Interface (bus)
EIDMultiPulseCylinder1_10ControlInterface01
Datalink (Fast)

Figure 8-20. EID Multi-Pulse Interfaces

After adding the interfaces, be sure to enable them by default under the EID Definition block.



Subsystem (mask) (link)

Parameters

ces | MixedMode/Ignition Interfaces | Injection Interfaces

- [ ] Enable Common Rail Signal Interface

Multi Pulse Cylinder Control Interface #1
- [x] Pulse Control Interface
- [x] Cylinders #1-#10 Trim Interface
- [ ] Cylinders #11-#20 Trim Interface

Multi Pulse Cylinder Control Interface #2
- [ ] Pulse Control Interface
- [ ] Cylinders #1-#10 Trim Interface
- [ ] Cylinders #11-#20 Trim Interface

Multi Pulse Cylinder Control Interface #3
- [ ] Pulse Control Interface
- [ ] Cylinders #1-#10 Trim Interface
- [ ] Cylinders #11-#20 Trim Interface

Multi Pulse Cylinder Control Interface #4
- [ ] Pulse Control Interface
- [ ] Cylinders #1-#10 Trim Interface
- [ ] Cylinders #11-#20 Trim Interface

Figure 8-21. EID Board Definition Injection Data Interface Enables

Diagnostic Examples

Adding Faults

This example demonstrates how to add new diagnostic faults into the application by adding sensor and range diagnostics to the AmbientAirPress sensor example demonstrated above. First, navigate to the Diagnostics (Medium) execution subsystem and add a subsystem for Pressure inputs. Inside of the Pressure subsystem, add a new AmbientAirPress subsystem.

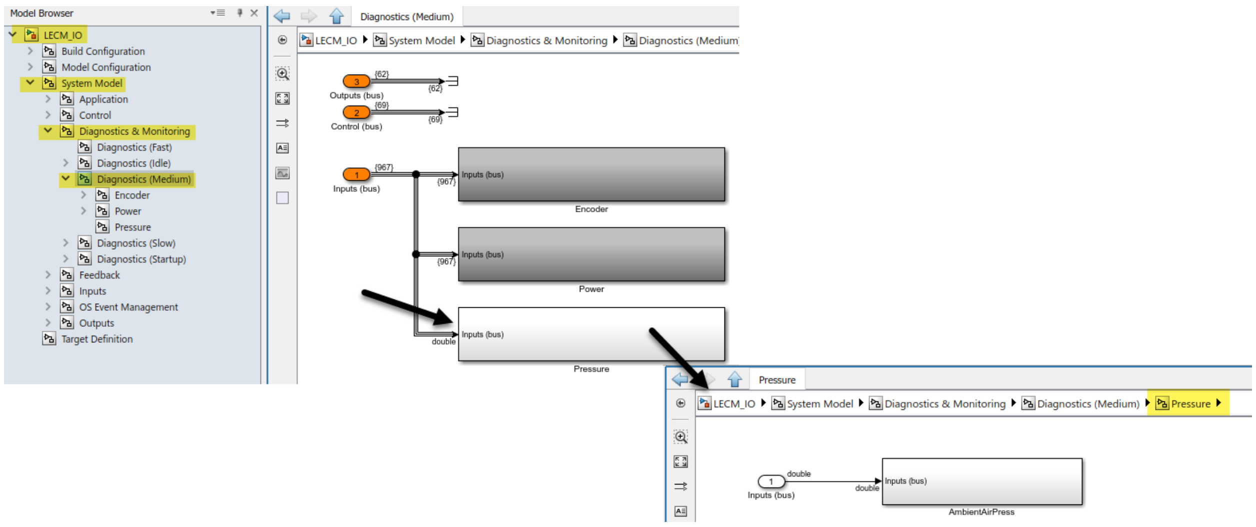


Figure 8-22. Diagnostics Medium Pressure Input Subsystem

In the AmbientAirPress subsystem, we'll create typical sensor diagnostics and range validation monitors. This will consist of a total of four faults:

- [Diagnostic] AmbientAirPressureSensorLowFault
    - Asserts when the sensor's voltage is below nominal range.
- [Diagnostic] AmbientAirPressureSensorHighFault
    - Asserts when the sensor's voltage is above the nominal range.
- [Monitor] AmbientAirPressureLowFault
    - Assert's when the engineering value of the ambient air pressure is considered too low, but sensor diagnostics are valid.
- [Monitor] AmbientAirPressureHighFault
    - Assert's when the engineering value of the ambient air pressure is considered too high, but sensor diagnostics are valid.

In this example, the sensor *diagnostics* validates the sensor hardware (V) is within range and valid. If the sensor is considered valid, then range low and high fault *monitors* (kPa) are used to monitor if the engineering value is below or above an acceptable nominal range.

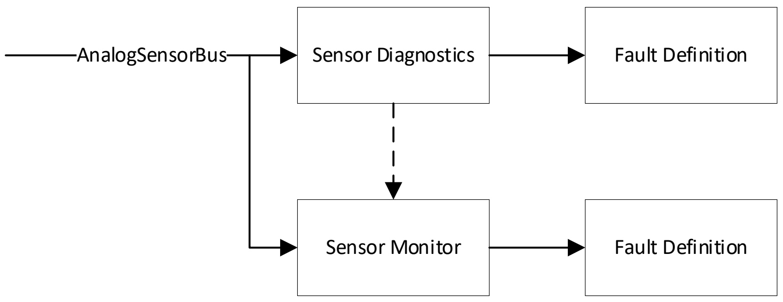


Figure 8-23. Ambient Air Pressure Diagnostics & Monitoring

Add subsystems for the sensor diagnostics and sensor monitors along with fault definitions using the example structure above as a guide.

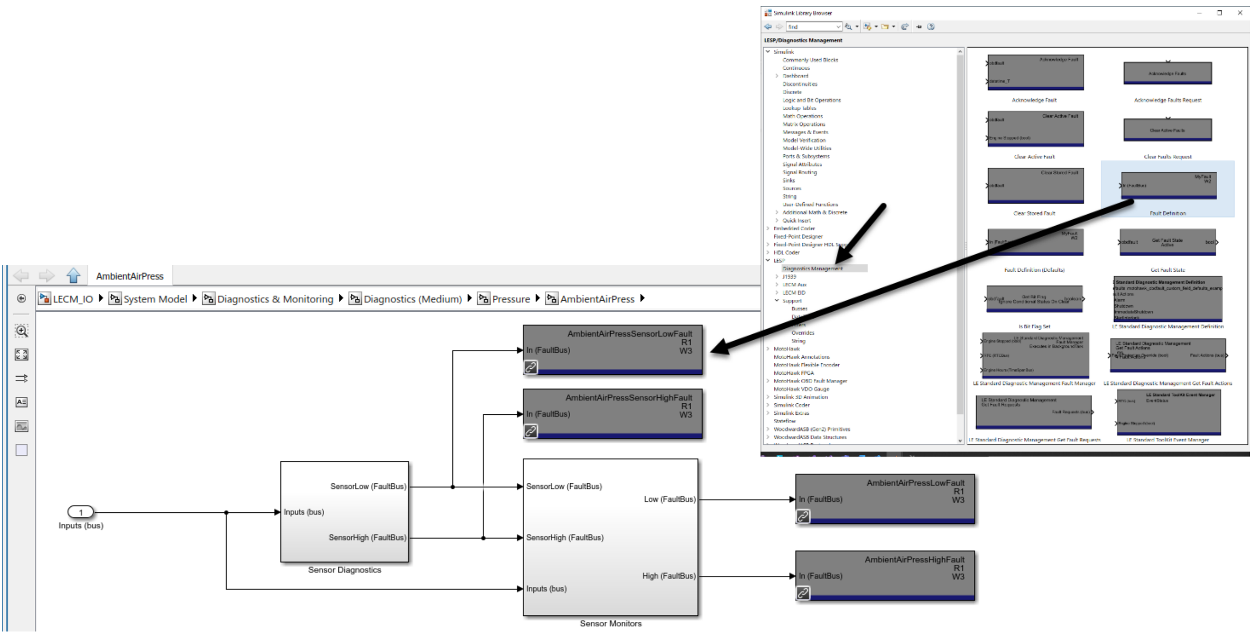


Figure 8-24. Ambient Air Pressure Diagnostics & Monitoring Implementation

Inside the Sensor Diagnostics, use a bus selector to select the Ambient Air Pressure Measurement Status. Recall from the example, the signal diagnostic checks are performed in the input processing of the sensor and reported as a status, thus the sensor diagnostic fault conditional logic can reference the signal status and assert a fault based on the status report. In this example, the SensorLowFault and SensorHighFault conditional logic is based on the measurement status reporting SignalLow and SignalHigh values respectively. These signals are tied to the FaultBus Create Conditional Status port (i.e., the condition for the fault exists). An ASB Boolean Persistence Filter is then used with the conditional status signal to provide a fault filter by ensuring the condition remains active for 250 ms continuously before asserting the fault by driving the Assertion Status port. The fault is considered inhibited if the Measurement Status reports Disabled (i.e., the sensor is not used).

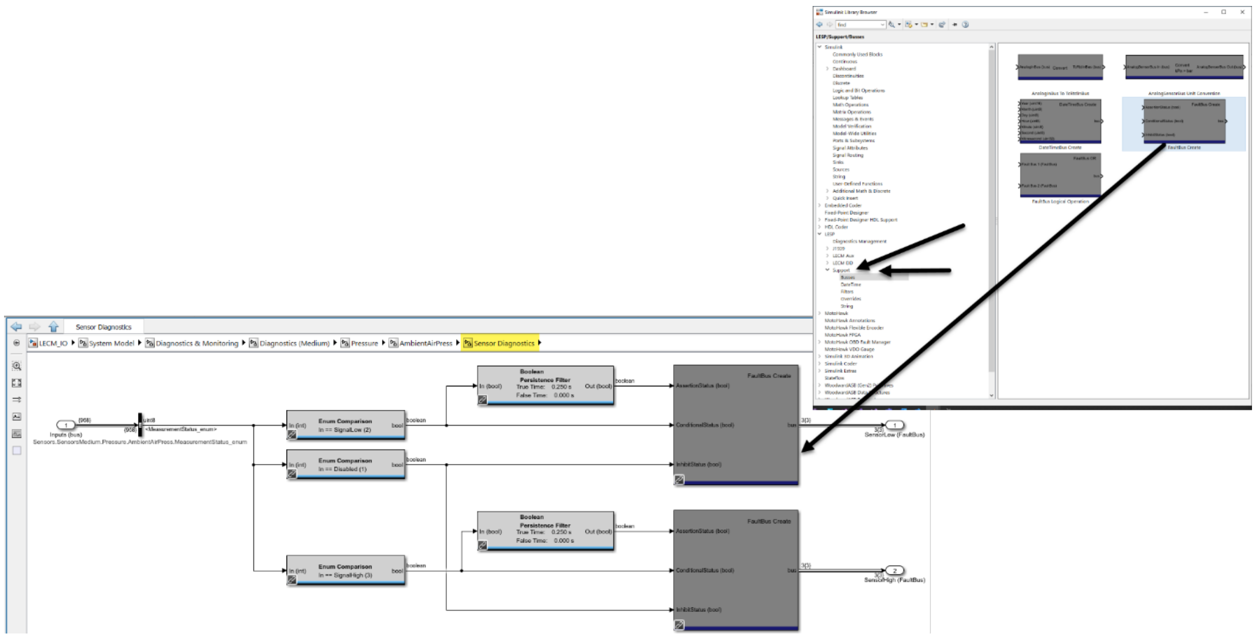

Sensor Diagnostics
LECM_IO ▸ System Model ▸ Diagnostics & Monitoring ▸ Diagnostics (Medium) ▸ Pressure ▸ AmbientAirPress ▸ Sensor Diagnostics ▸
Inputs (bus)
Sensors.SensorsMedium.Pressure.AmbientAirPress.MeasurementStatus_enum
Enum Comparison In == SignalLow (2)
Enum Comparison In == Disabled (1)
Enum Comparison In == SignalHigh (3)
Boolean Persistence Filter True Time: 0.250 s False Time: 0.000 s
FaultBus Create
AssertionStatus (bool)
ConditionalStatus (bool)
InhibitStatus (bool)
SensorLow (FaultBus)
SensorHigh (FaultBus)
Simulink Library Browser
LESP/Support/Buses
FaultBus Create
FaultBus Logical Operation

Figure 8-25. Ambient Air Pressure Sensor Diagnostics

Lastly, the Sensor Monitors subsystem implements range check logic on the filtered Ambient Air Pressure value with hysteresis using the ASB Range Comparison block. The inhibit logic is used to prevent the fault from asserting if the sensor diagnostic conditional statuses are active (i.e., sensor is not valid) or if the sensor diagnostics are inhibited (i.e., the sensor is disabled). The ASB Boolean Persistence filter is also used to drive the assertion status of the fault; however, not the input to the filter is gated with the "Not Inhibited" signal. This condition forces the filter to reset if the inhibit condition is true.

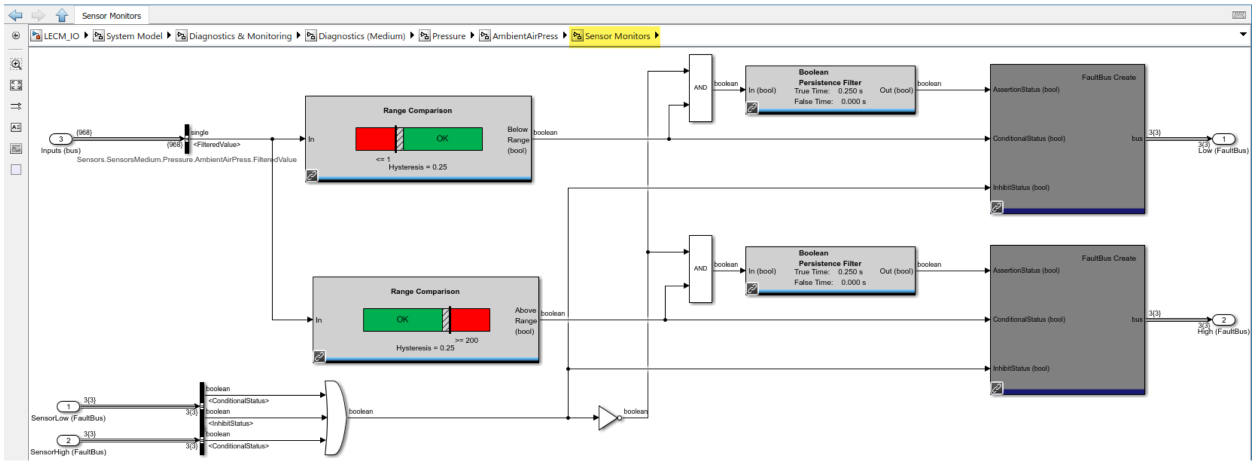


Figure 8-26. Ambient Air Pressure Monitors

Now that the four faults have been added, we need to update their Fault ID code andJ1939 SPN/FMI mappings. Navigate to the LESP Standard Diagnostics Management Definition block under Model Configuration and launch the fault property manager. The fault manager can also be launched by right-clicking on the Simulink model window and using the LESP context menu.

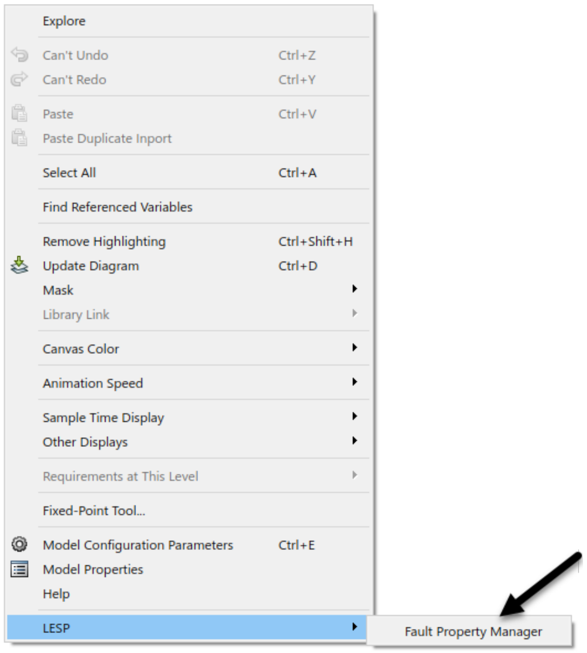
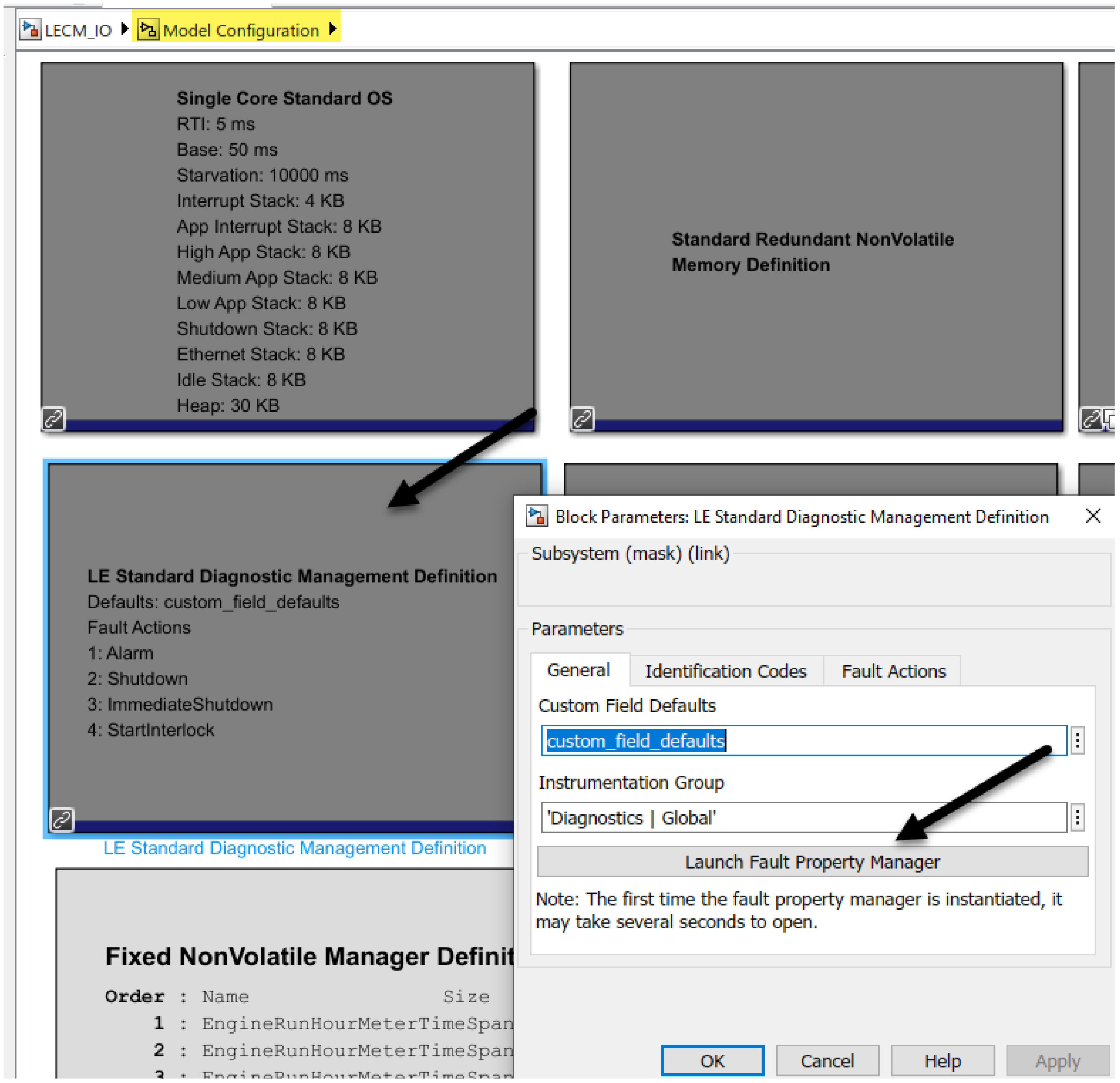


Figure 8-27. LESP Context Menu (Fault Property Manager)


LECM_IO
Model Configuration
Standard Redundant NonVolatile Memory Definition
LE Standard Diagnostic Management Definition
Block Parameters: LE Standard Diagnostic Management Definition
Subsystem (mask) (link)
Parameters
General
Identification Codes
Fault Actions
Launch Fault Property Manager
OK
Cancel
Help
Apply

Single Core Standard OS
RTI: 5 ms
Base: 50 ms
Starvation: 10000 ms
Interrupt Stack: 4 KB
App Interrupt Stack: 8 KB
High App Stack: 8 KB
Medium App Stack: 8 KB
Low App Stack: 8 KB
Shutdown Stack: 8 KB
Ethernet Stack: 8 KB
Idle Stack: 8 KB
Heap: 30 KB

LE Standard Diagnostic Management Definition
Defaults: custom_field_defaults
Fault Actions
1: Alarm
2: Shutdown
3: ImmediateShutdown
4: StartInterlock

Custom Field Defaults: custom_field_defaults
Instrumentation Group: 'Diagnostics | Global'

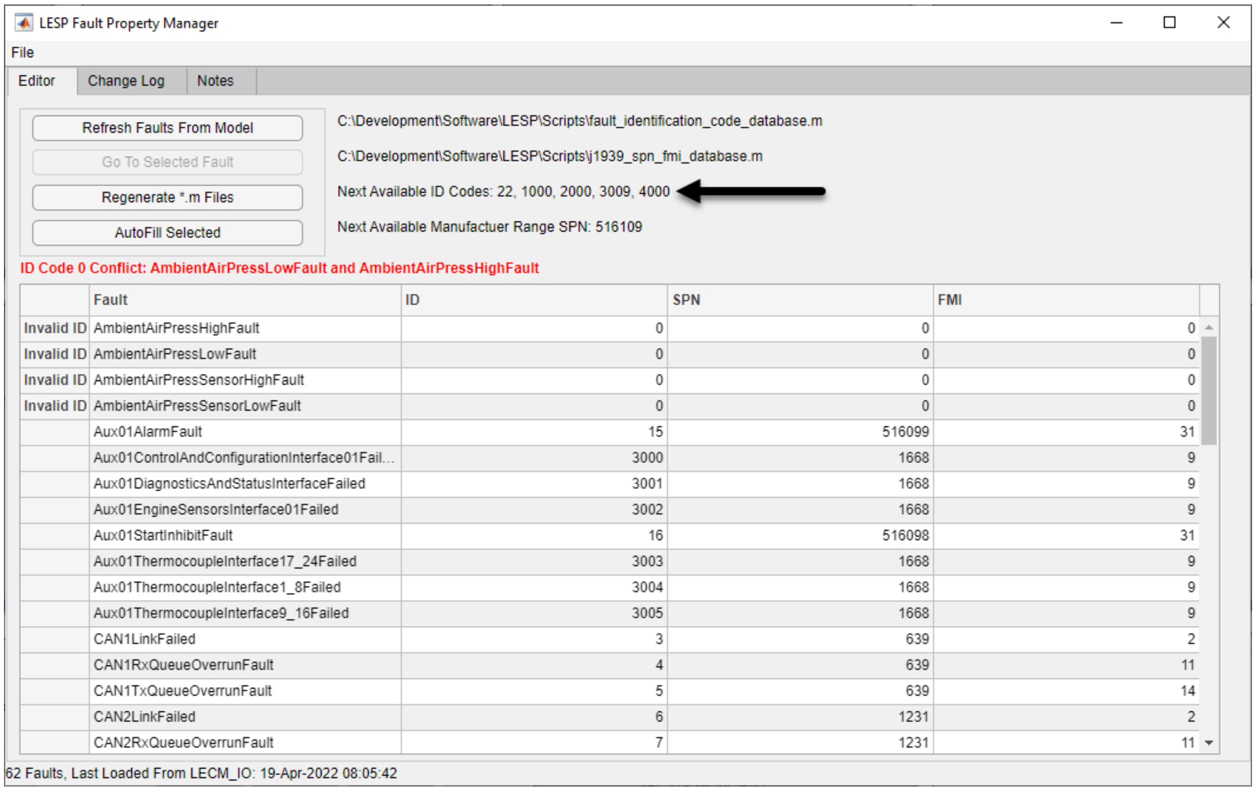
Note: The first time the fault property manager is instantiated, it may take several seconds to open.

Fixed NonVolatile Manager Definit

```
Order : Name                    Size
    1 : EngineRunHourMeterTimeSpan
    2 : EngineRunHourMeterTimeSpan
    3 : EngineRunHourMeterTimeSpan
```

Figure 8-28. Fault Property Manager

If the new faults are not visible at the top, click the ID header to sort by ID in ascending order. This will bring 0 valued codes to the top of the list.



C:\Development\Software\LESP\Scripts\fault_identification_code_database.m

C:\Development\Software\LESP\Scripts\j1939_spn_fmi_database.m

Next Available ID Codes: 22, 1000, 2000, 3009, 4000

Next Available Manufactuer Range SPN: 516109

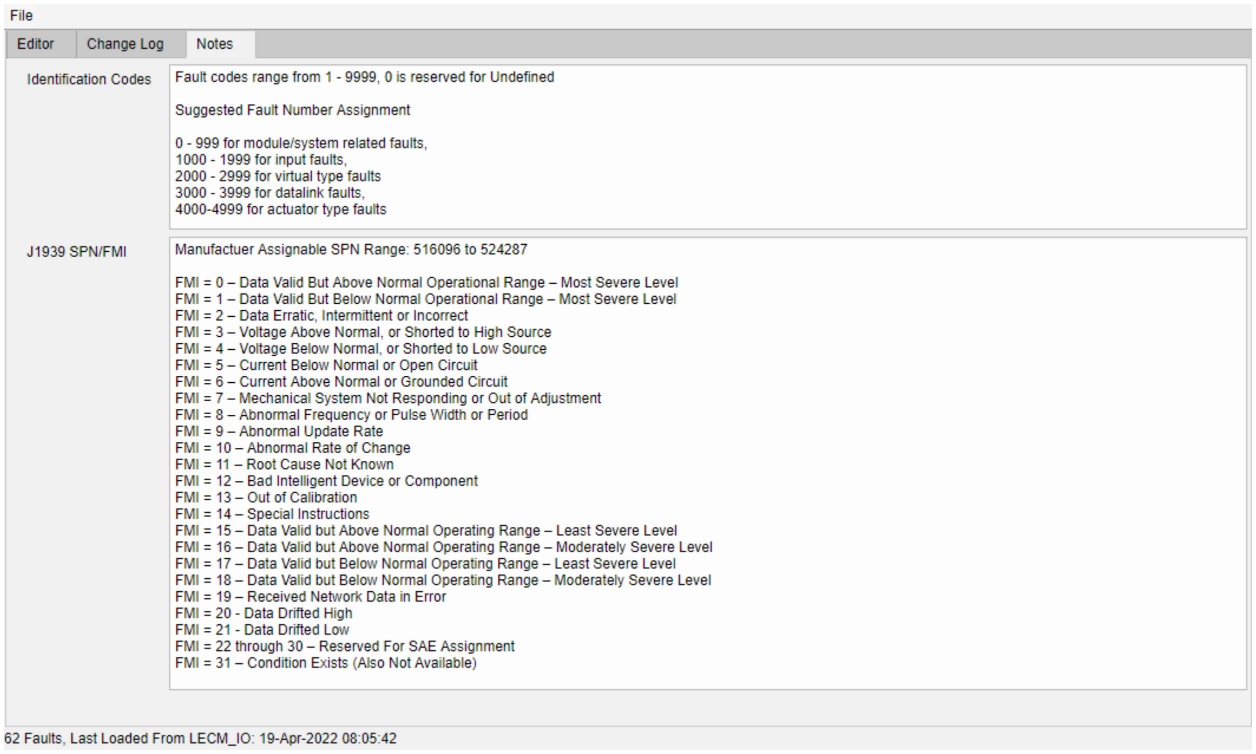
ID Code 0 Conflict: AmbientAirPressLowFault and AmbientAirPressHighFault

	Fault	ID	SPN	FMI
Invalid ID	AmbientAirPressHighFault	0	0	0
Invalid ID	AmbientAirPressLowFault	0	0	0
Invalid ID	AmbientAirPressSensorHighFault	0	0	0
Invalid ID	AmbientAirPressSensorLowFault	0	0	0
	Aux01AlarmFault	15	516099	31
	Aux01ControlAndConfigurationInterface01Fail...	3000	1668	9
	Aux01DiagnosticsAndStatusInterfaceFailed	3001	1668	9
	Aux01EngineSensorsInterface01Failed	3002	1668	9
	Aux01StartInhibitFault	16	516098	31
	Aux01ThermocoupleInterface17_24Failed	3003	1668	9
	Aux01ThermocoupleInterface1_8Failed	3004	1668	9
	Aux01ThermocoupleInterface9_16Failed	3005	1668	9
	CAN1LinkFailed	3	639	2
	CAN1RxQueueOverrunFault	4	639	11
	CAN1TxQueueOverrunFault	5	639	14
	CAN2LinkFailed	6	1231	2
	CAN2RxQueueOverrunFault	7	1231	11

62 Faults, Last Loaded From LECM_IO: 19-Apr-2022 08:05:42

Figure 8-29. Fault Property Manager (Ambient Air Pressure Defaults)

Use the fields to enter unique IDs and for each fault. The tool will display the next available ID codes in 0-999, 1000-1999, 2000-2999, 3000-3999, and 4000-4999 ranges for convenience; however, any ID code strategy can be employed. Click on the Notes tab to view suggestions and J1939 FMI codes. Use the J1939 standard to lookup an appropriate SPN for ambient air pressure signal (SPN108).



Identification Codes

Fault codes range from 1 - 9999, 0 is reserved for Undefined

Suggested Fault Number Assignment

0 - 999 for module/system related faults,
1000 - 1999 for input faults,
2000 - 2999 for virtual type faults
3000 - 3999 for datalink faults,
4000-4999 for actuator type faults

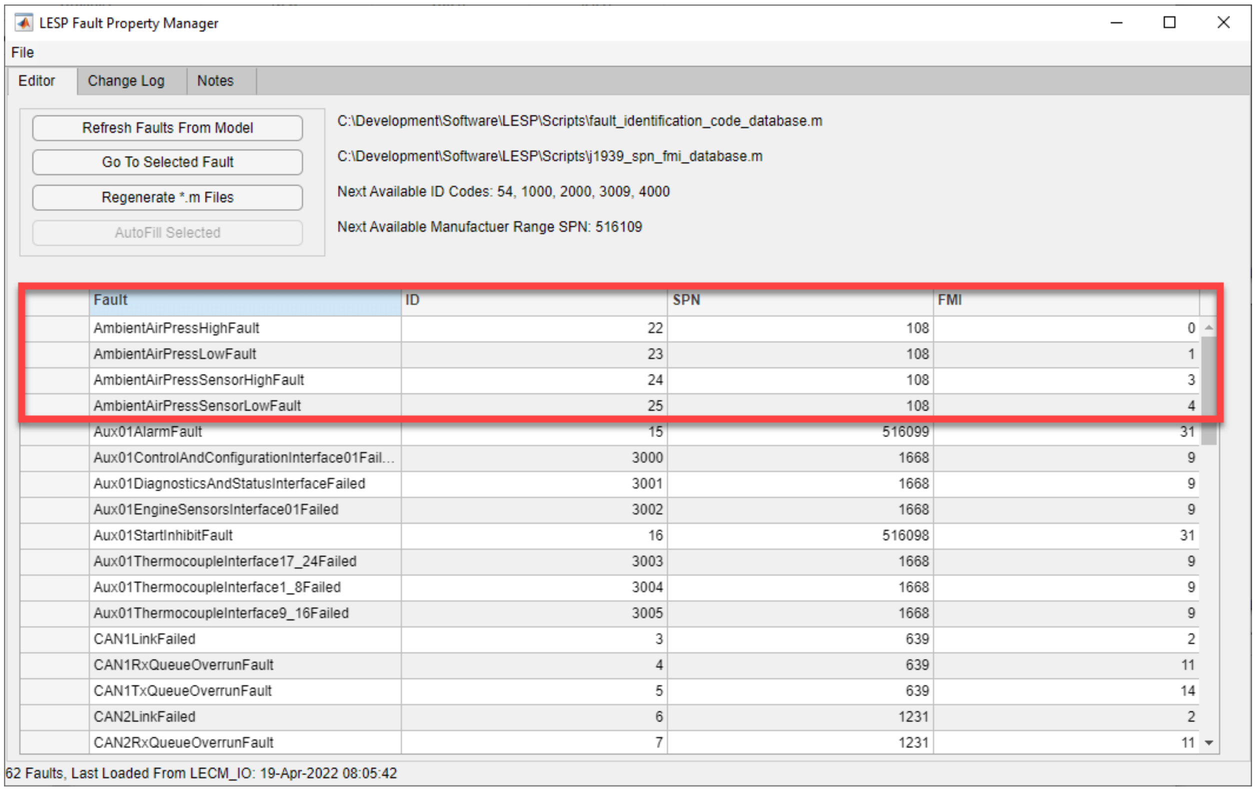
J1939 SPN/FMI

Manufactuer Assignable SPN Range: 516096 to 524287

FMI = 0 – Data Valid But Above Normal Operational Range – Most Severe Level
FMI = 1 – Data Valid But Below Normal Operational Range – Most Severe Level
FMI = 2 – Data Erratic, Intermittent or Incorrect
FMI = 3 – Voltage Above Normal, or Shorted to High Source
FMI = 4 – Voltage Below Normal, or Shorted to Low Source
FMI = 5 – Current Below Normal or Open Circuit
FMI = 6 – Current Above Normal or Grounded Circuit
FMI = 7 – Mechanical System Not Responding or Out of Adjustment
FMI = 8 – Abnormal Frequency or Pulse Width or Period
FMI = 9 – Abnormal Update Rate
FMI = 10 – Abnormal Rate of Change
FMI = 11 – Root Cause Not Known
FMI = 12 – Bad Intelligent Device or Component
FMI = 13 – Out of Calibration
FMI = 14 – Special Instructions
FMI = 15 – Data Valid but Above Normal Operating Range – Least Severe Level
FMI = 16 – Data Valid but Above Normal Operating Range – Moderately Severe Level
FMI = 17 – Data Valid but Below Normal Operating Range – Least Severe Level
FMI = 18 – Data Valid but Below Normal Operating Range – Moderately Severe Level
FMI = 19 – Received Network Data in Error
FMI = 20 - Data Drifted High
FMI = 21 - Data Drifted Low
FMI = 22 through 30 – Reserved For SAE Assignment
FMI = 31 – Condition Exists (Also Not Available)

62 Faults, Last Loaded From LECM_IO: 19-Apr-2022 08:05:42

Figure 8-30. Fault Property Manager Notes



C:\Development\Software\LESP\Scripts\fault_identification_code_database.m

C:\Development\Software\LESP\Scripts\j1939_spn_fmi_database.m

Next Available ID Codes: 54, 1000, 2000, 3009, 4000

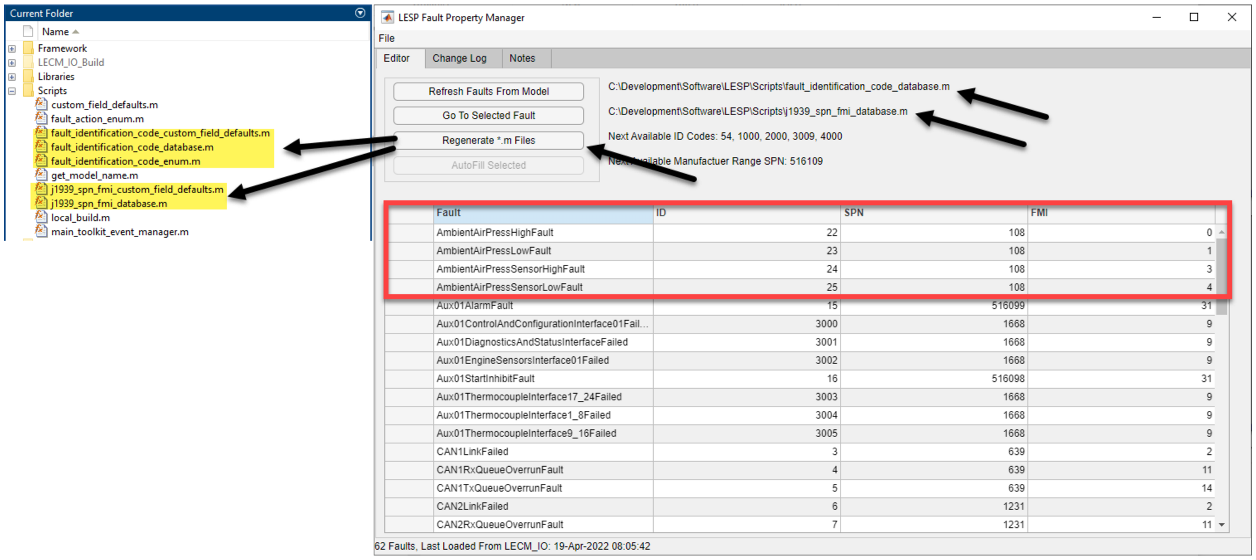
Next Available Manufactuer Range SPN: 516109

Fault	ID	SPN	FMI
AmbientAirPressHighFault	22	108	0
AmbientAirPressLowFault	23	108	1
AmbientAirPressSensorHighFault	24	108	3
AmbientAirPressSensorLowFault	25	108	4
Aux01AlarmFault	15	516099	31
Aux01ControlAndConfigurationInterface01Fail…	3000	1668	9
Aux01DiagnosticsAndStatusInterfaceFailed	3001	1668	9
Aux01EngineSensorsInterface01Failed	3002	1668	9
Aux01StartInhibitFault	16	516098	31
Aux01ThermocoupleInterface17_24Failed	3003	1668	9
Aux01ThermocoupleInterface1_8Failed	3004	1668	9
Aux01ThermocoupleInterface9_16Failed	3005	1668	9
CAN1LinkFailed	3	639	2
CAN1RxQueueOverrunFault	4	639	11
CAN1TxQueueOverrunFault	5	639	14
CAN2LinkFailed	6	1231	2
CAN2RxQueueOverrunFault	7	1231	11

62 Faults, Last Loaded From LECM_IO: 19-Apr-2022 08:05:42

Figure 8-31. Fault Property Manager (Ambient Air Pressure Fault Properties)

Lastly, verify the correct scripts are referenced and hit the Regenerate *.m Files button to recreate the new fault property files.


Current Folder
Name
Framework
LECM_IO_Build
Libraries
Scripts
custom_field_defaults.m
fault_action_enum.m
fault_identification_code_custom_field_defaults.m
fault_identification_code_database.m
fault_identification_code_enum.m
get_model_name.m
j1939_spn_fmi_custom_field_defaults.m
j1939_spn_fmi_database.m
local_build.m
main_toolkit_event_manager.m
LESP Fault Property Manager
File
Editor
Change Log
Notes
Refresh Faults From Model
Go To Selected Fault
Regenerate *.m Files
AutoFill Selected

C:\Development\Software\LESP\Scripts\fault_identification_code_database.m

C:\Development\Software\LESP\Scripts\j1939_spn_fmi_database.m

Next Available ID Codes: 54, 1000, 2000, 3009, 4000

Next Available Manufactuer Range SPN: 516109

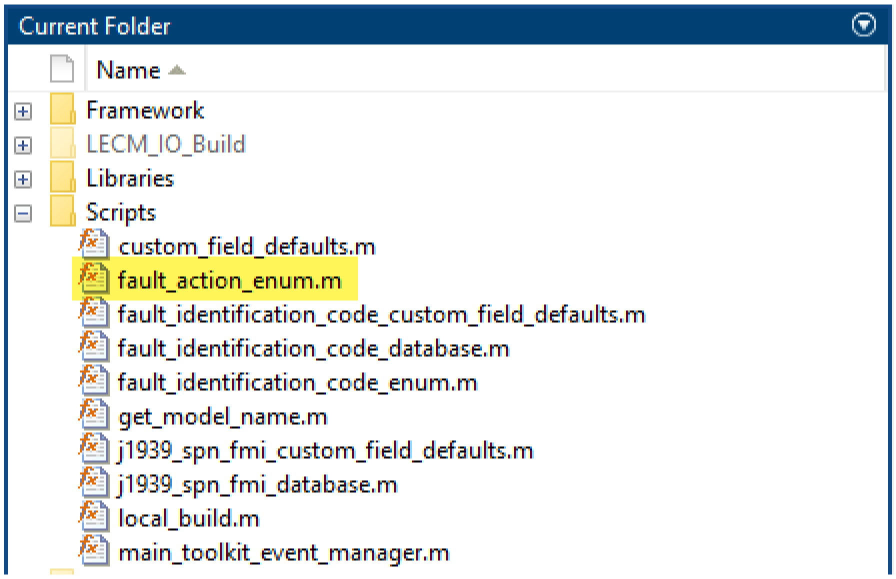
Fault	ID	SPN	FMI
AmbientAirPressHighFault	22	108	0
AmbientAirPressLowFault	23	108	1
AmbientAirPressSensorHighFault	24	108	3
AmbientAirPressSensorLowFault	25	108	4
Aux01AlarmFault	15	516099	31
Aux01ControlAndConfigurationInterface01Fail…	3000	1668	9
Aux01DiagnosticsAndStatusInterfaceFailed	3001	1668	9
Aux01EngineSensorsInterface01Failed	3002	1668	9
Aux01StartInhibitFault	16	516098	31
Aux01ThermocoupleInterface17_24Failed	3003	1668	9
Aux01ThermocoupleInterface1_8Failed	3004	1668	9
Aux01ThermocoupleInterface9_16Failed	3005	1668	9
CAN1LinkFailed	3	639	2
CAN1RxQueueOverrunFault	4	639	11
CAN1TxQueueOverrunFault	5	639	14
CAN2LinkFailed	6	1231	2
CAN2RxQueueOverrunFault	7	1231	11

62 Faults, Last Loaded From LECM_IO: 19-Apr-2022 08:05:42

Figure 8-32. Fault Property Manager (Regenerate Files)

Adding Fault Actions

This example demonstrates how to add new diagnostic fault actions into the application. Fault actions are defined in the local fault_actions_enum.m script located under ./Scripts folder.



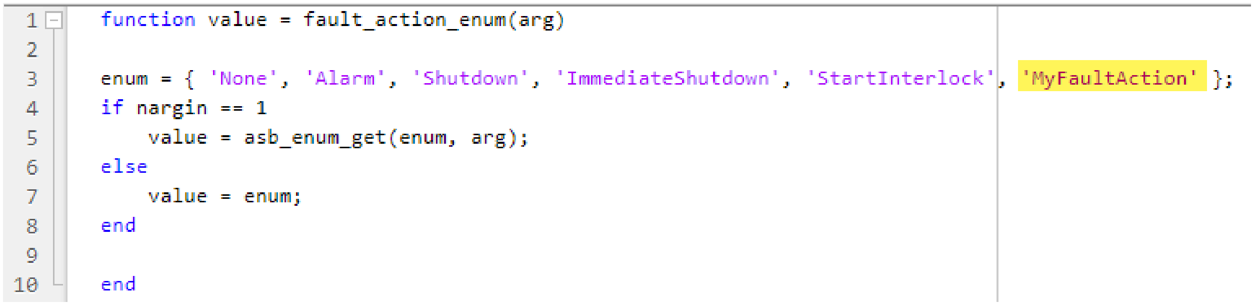
Current Folder

Name

- Framework
- LECM_IO_Build
- Libraries
- Scripts
  - custom_field_defaults.m
  - fault_action_enum.m
  - fault_identification_code_custom_field_defaults.m
  - fault_identification_code_database.m
  - fault_identification_code_enum.m
  - get_model_name.m
  - j1939_spn_fmi_custom_field_defaults.m
  - j1939_spn_fmi_database.m
  - local_build.m
  - main_toolkit_event_manager.m

Figure 8-33. fault_action_enum Script

Open the script for editing and append a new fault action. After adding the fault action, any dependent fault action enumeration blocks (e.g., LESP Standard Diagnostics Management Definition, Get Fault Actions blocks) need to be refreshed.

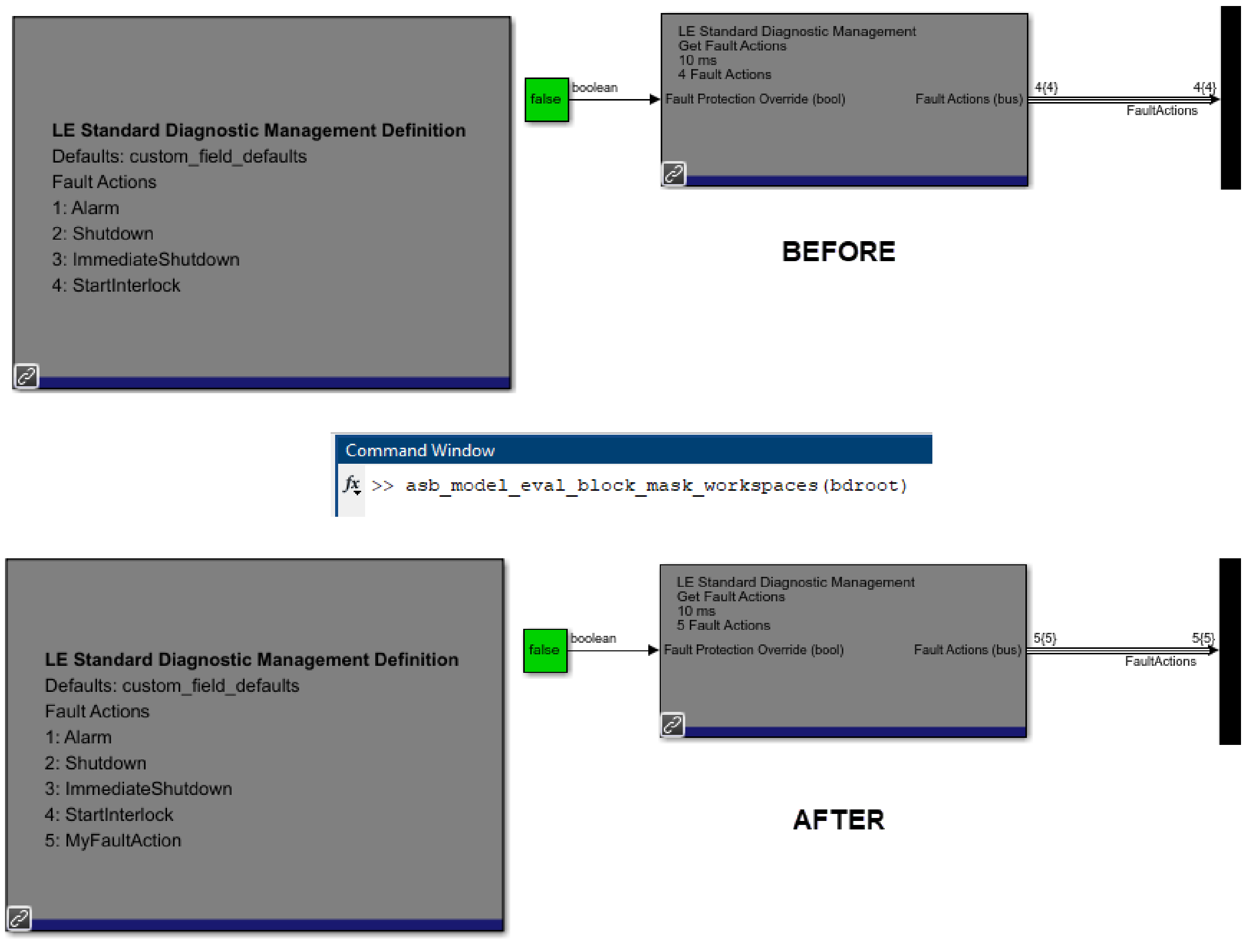


```
function value = fault_action_enum(arg)

enum = { 'None', 'Alarm', 'Shutdown', 'ImmediateShutdown', 'StartInterlock', 'MyFaultAction' };
if nargin == 1
    value = asb_enum_get(enum, arg);
else
    value = enum;
end

end
```

Figure 8-34. fault_action_enum Script (MyFaultAction)

This can be accomplished by executing the asb_model_eval_block_mask_workspaces script and passing in the name of the model (or bdroot if the model is in the foreground). After the refresh is complete, verify the model updates without errors. At this point, the new fault action will be available to bind to faults defined in the model and the new fault action boolean signal should be available in the FaultActions bus that can be used in application logic to drive the appropriate control behaviors.

IMPORTANT **Do not skip the step of asb_model_eval_block_mask_workspaces; otherwise, the blocks may report errors during a model update.**



```
>> asb_model_eval_block_mask_workspaces(bdroot)
```

Figure 8-35. Refresh Block Workspaces

Output Examples

Adding LSO Discrete/PWM Control

The LESP platform supports hardware-dependent discrete output manager blocks that accept DiscreteOutputBus types and dynamically map the input busses to the assigned discrete output resource (LSOx, HSOx, etc.). In addition, the mappings array is output and used with the Discrete Output Mapping Conflict diagnostic block to assert a fault if a mapping conflict arises. This block enables control logic to support dynamic IO mapping of outputs if desired. If dynamic mapping is not required, this block can be removed, and direct mapping implemented.

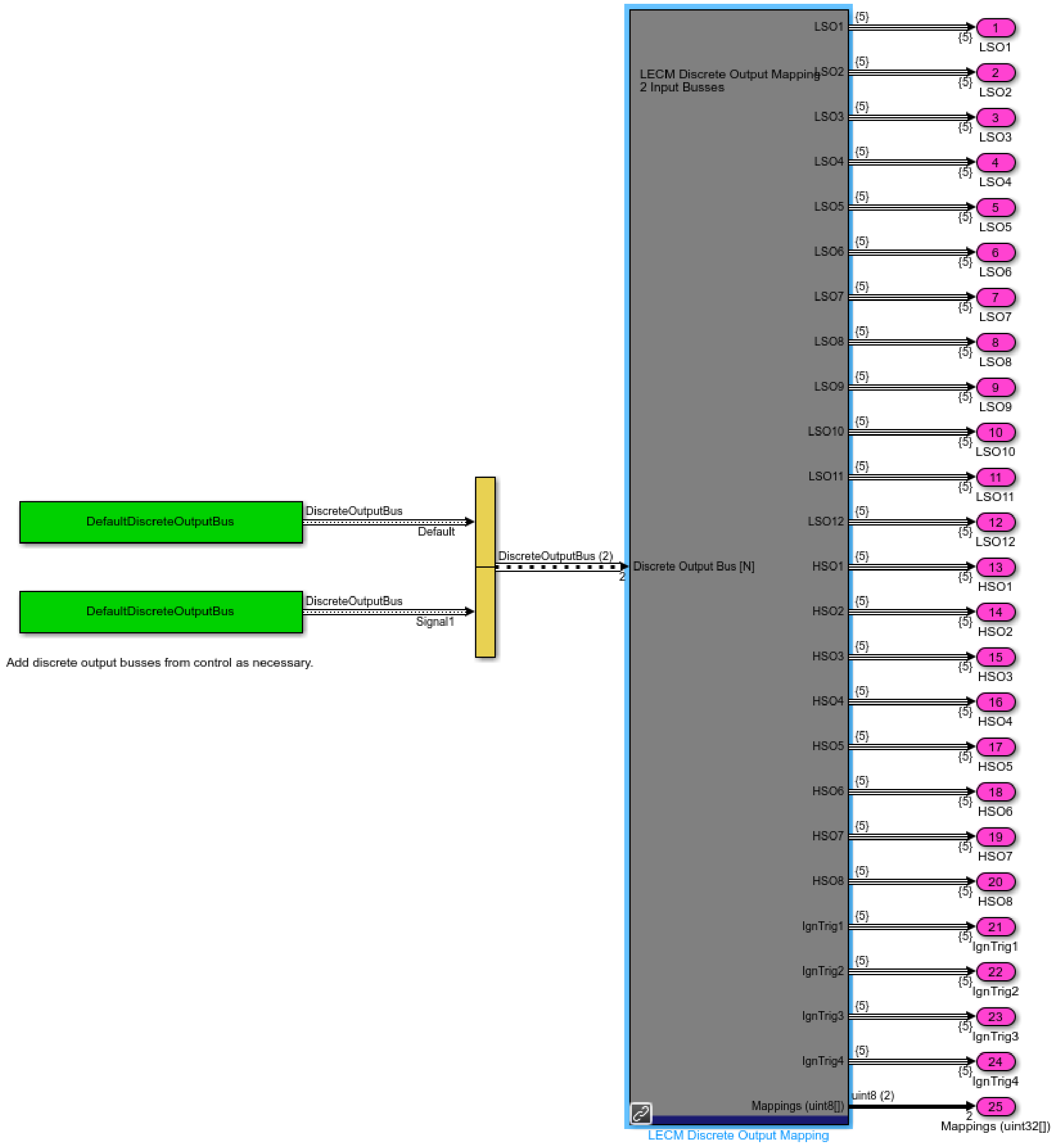


Figure 8-36. LECM-based Discrete Output Mapping Block

In this example, we'll add two different discrete output controllers: Discrete (on/off) and PWM. The first will drive HSO1 to an active (on) state when the ImmediateShutdown fault action is active. The second, will pulse-width-modulate HSO2 at a frequency of 100 Hz and a duty cycle that is a function of the current engine speed (0...1800 RPM → 0...100% duty cycle). First, navigate to the Control Medium subsystem and add a subsystem for HSO1 and HSO2 controls with an Inputs bus inport.

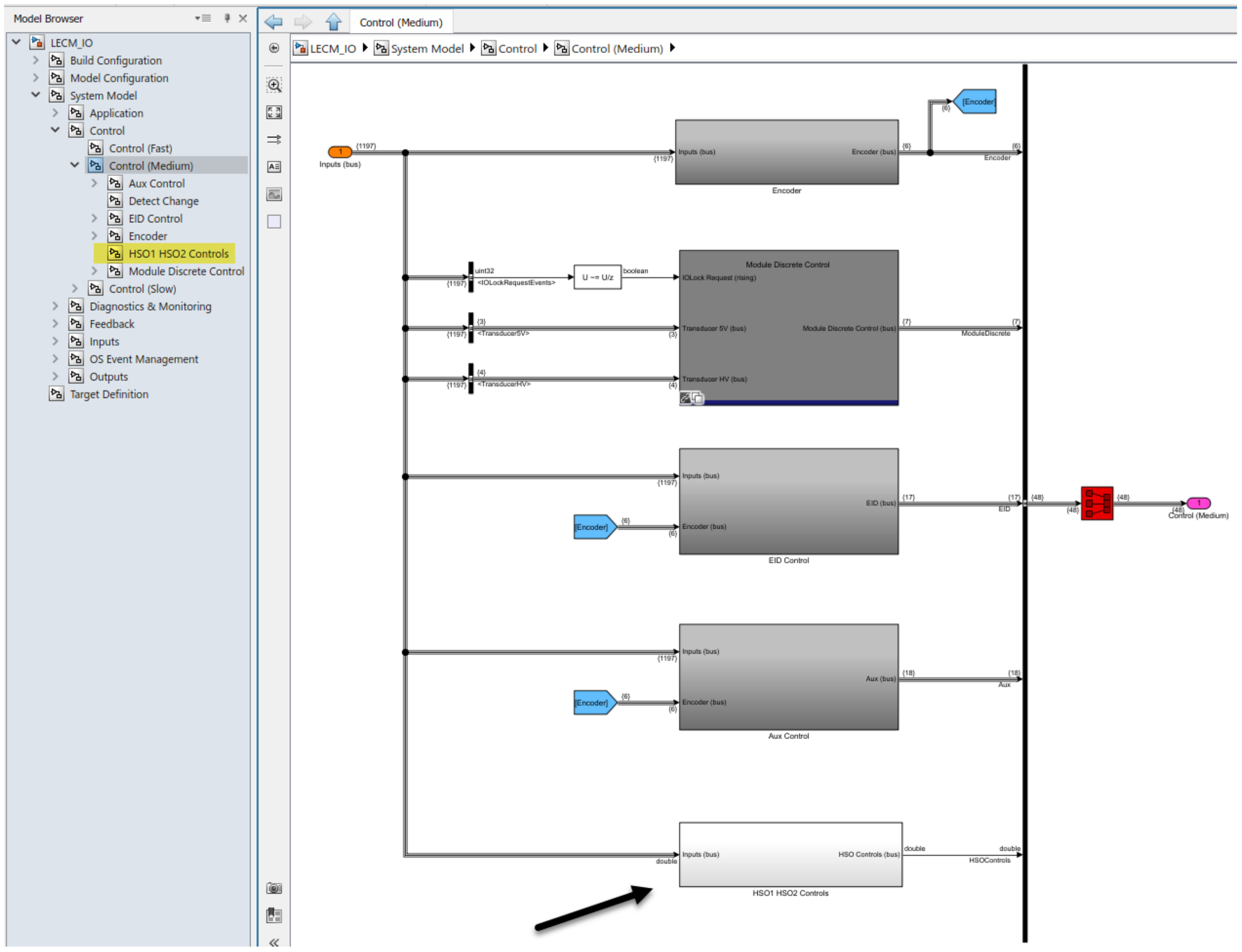


Figure 8-37. HSO Controls Subsystem

Inside of the subsystem, open the Simulink browser and add a copy of the DiscreteOutBus (Discrete Mode) and DiscreteOutBus (PWM Mode) blocks. Use the lecm_discrete_out_io_select_enum to select the respective HSO1 and HSO2 resources and associated with the IO Select inports. For HSO1, select the ImmediateShutdown fault action signal and connect to the Activate port. For HSO2, use a MotoHawk 1D lookup table to convert from speed to duty cycle and connect with the Duty Cycle inport along with a fix valued Enable and Frequency ports. Note: MotoHawk calibrations or other logic can be used instead of constants or for the IO Select ports if runtime IO mapping is desired. Locate the DiscreteOutputBus types on a bus creator to be used in the output logic.

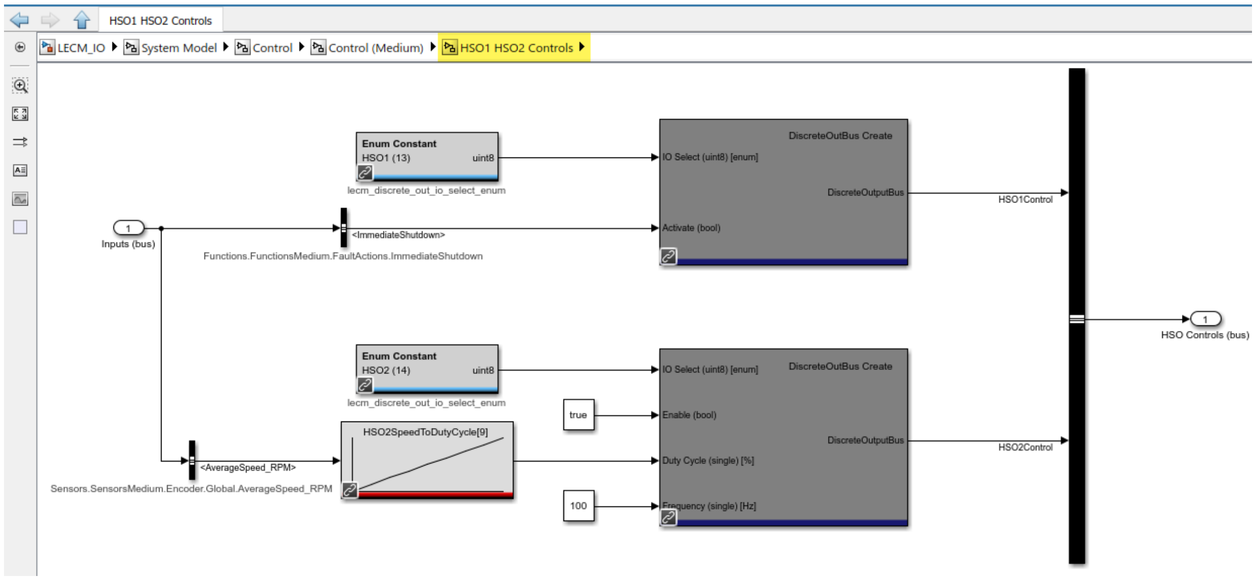

HSO1 HSO2 Controls
LECM_IO ▸ System Model ▸ Control ▸ Control (Medium) ▸ HSO1 HSO2 Controls ▸
Enum Constant HSO1 (13) uint8
lecm_discrete_out_io_select_enum
DiscreteOutBus Create
IO Select (uint8) [enum]
DiscreteOutputBus
Activate (bool)
HSO1Control
1 Inputs (bus)
<ImmediateShutdown>
Functions.FunctionsMedium.FaultActions.ImmediateShutdown
1 HSO Controls (bus)
Enum Constant HSO2 (14) uint8
lecm_discrete_out_io_select_enum
true
Enable (bool)
HSO2SpeedToDutyCycle[9]
<AverageSpeed_RPM>
Sensors.SensorsMedium.Encoder.Global.AverageSpeed_RPM
Duty Cycle (single) [%]
100
Frequency (single) [Hz]
HSO2Control

Figure 8-38. HSO1/HSO2 Controls Subsystem Logic

Lastly, navigate to the Outputs/Hardware Fast/Discrete Output Mapping subsystem. Use a bus selector to select the HSO1 and HSO2 control busses from the Control bus. Use a Signal Conversion block to convert the bus to a NonVirtual type and expand the vector concatenate block to three ports. Also change the mask parameter of the LECM Discrete Mapping block to three. Note, the first "Default" input must be maintained as this is the signal selected when the IO is not used. Finally, navigate to Diagnostics & Monitoring Idle subsystem and change the Output Mapping Fault mask width parameter to a value of three as well. Verify the model updates without errors. This process can be used to expand to any number of different discrete/PWM-based controls and commands.

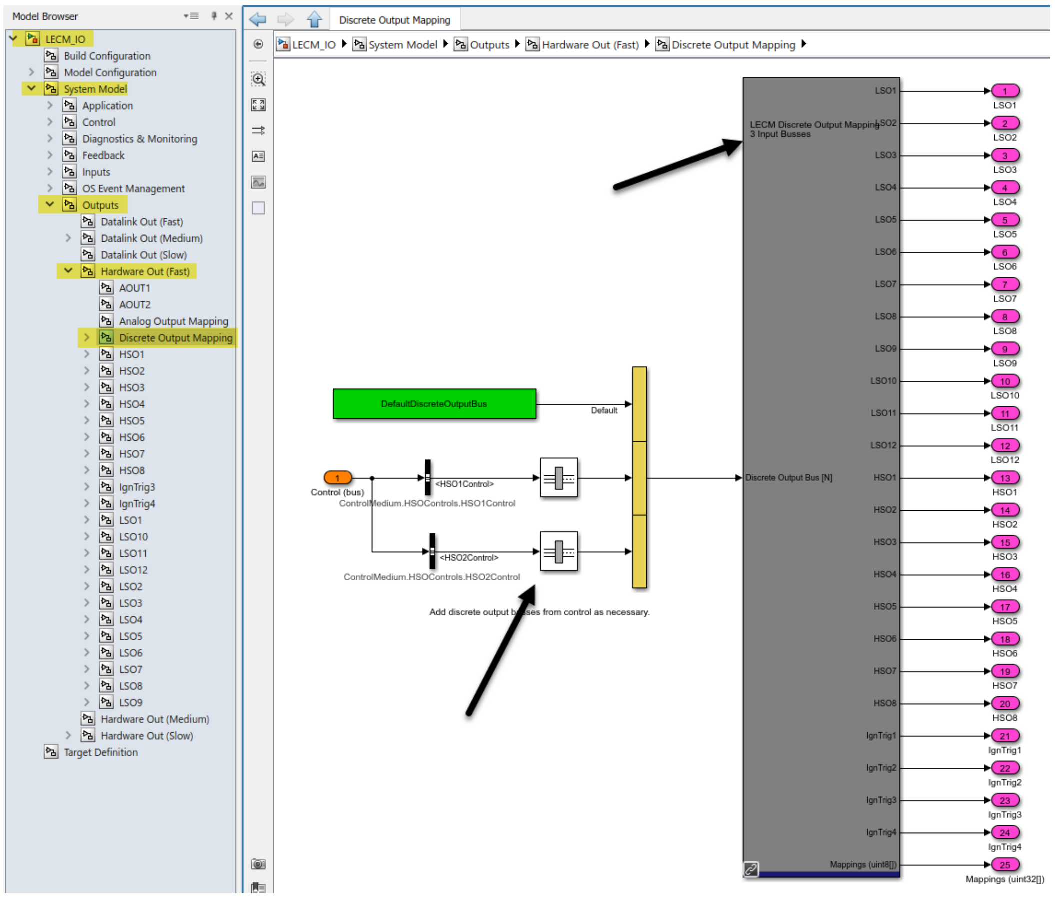


Figure 8-39. LECM Discrete Output Mapping Interface Update

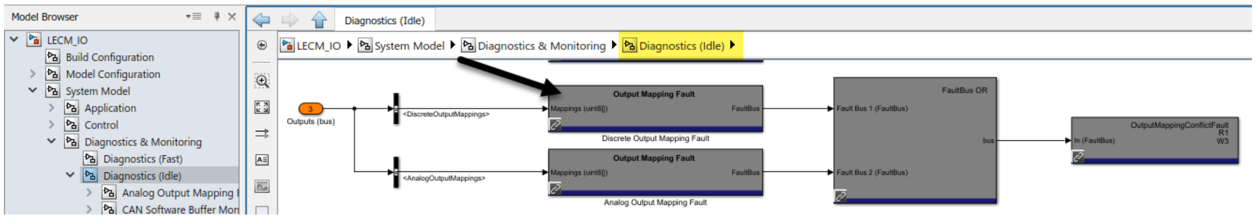


Figure 8-40. Discrete Output Mapping Conflict Fault Detection Logic

Adding Analog Output Control

The LESP platform supports hardware-dependent analog output manager blocks that accept AnalogOutputBus types and dynamically maps the input busses to the assigned analog output resource (e.g., AOUT1). In addition, the mappings array is output and used with the Output Mapping Conflict diagnostic block to assert a fault if a mapping conflict arises. This block enables control logic to support dynamic IO mapping of outputs if desired. If dynamic mapping is not required, this block can be removed, and direct mapping implemented.

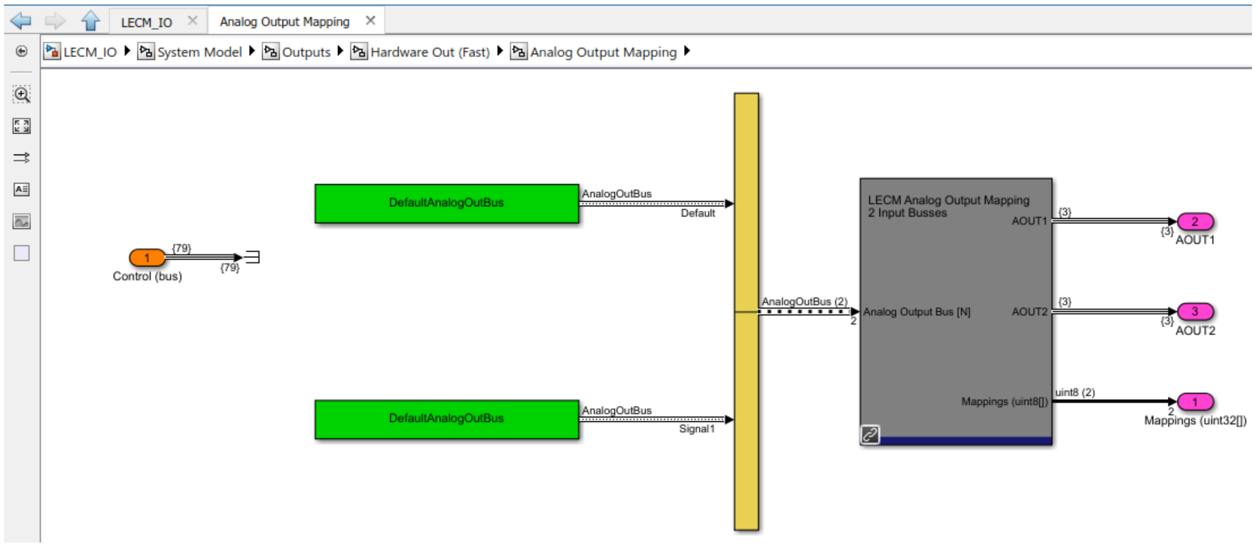


Figure 8-41. LECM-based Analog Output Mapping Block

In this example, we’ll add an Analog output that uses AOUT1 and maps the current engine speed to an AOUT1 command (0...1800 RPM → 4...20 mA). First, navigate to the Control Medium subsystem and add a subsystem for AOUT1 control with an Inputs bus inport.

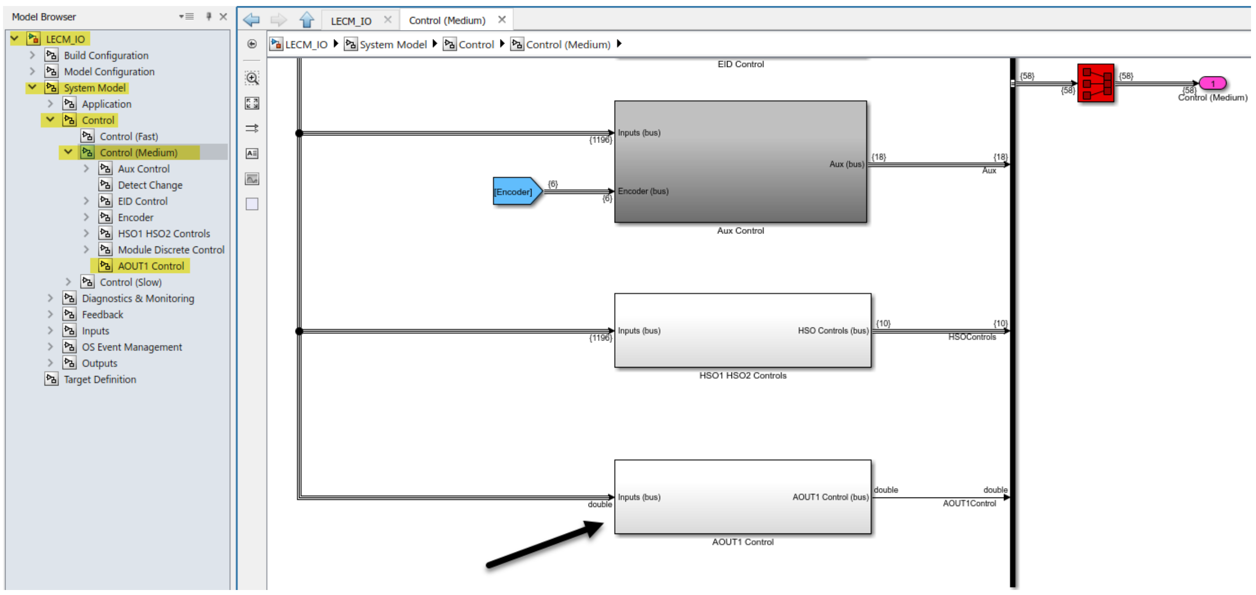


Figure 8-42. AOUT1 Control Subsystem

Inside of the subsystem, open the Simulink browser and add a copy of the AnalogOutBus block. Use the lecm_analog_out_io_select_enum to select the respective AOUT1 resource and associate with the IO Select inport. Use a MotoHawk 1D lookup table to convert from speed to mA command and associate with the Current Demand inport along with a fix valued Enable port. Note: MotoHawk calibrations can be used instead of constants or for the IO Select ports if runtime IO mapping is desired.

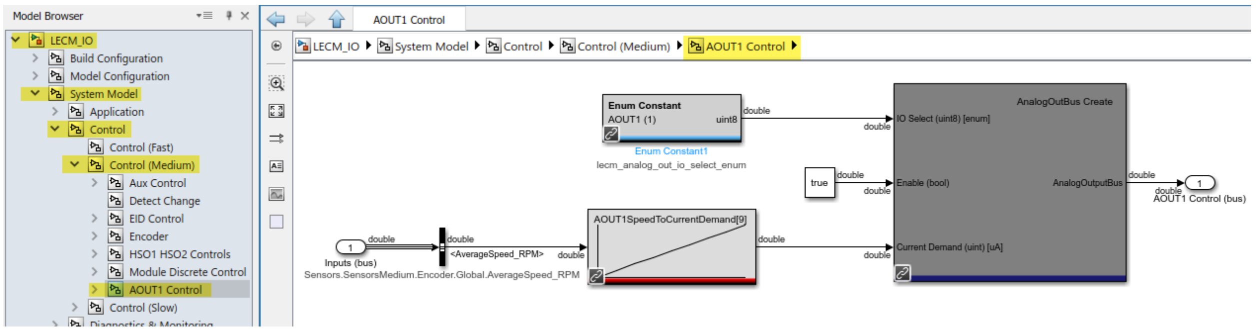


Figure 8-43. AOUT1 Control Subsystem Logic

Lastly, navigate to the Outputs/Hardware Fast/Analog Output Mapping subsystem. Use a bus selector to select the AOUT1 bus from the Control bus. Use a Signal Conversion block to convert the bus to a NonVirtual type and expand the vector concatenate block more ports if required. Verify the mask parameter of the LECM Analog Mapping block matches the input width. Note, the first “Default” input must be maintained as this is the bus selected when the IO is not used. Finally, navigate to Diagnostics & Monitoring Idle subsystem and change the Analog Output Mapping Fault mask width parameter to match as well. Verify the model updates without errors.

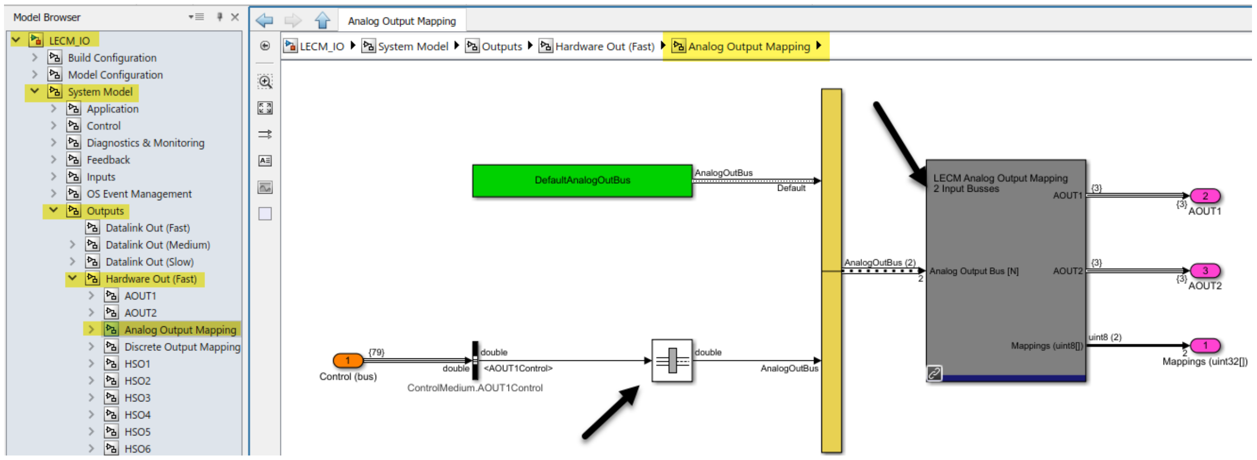


Figure 8-44. LECM Analog Output Mapping Interface Update

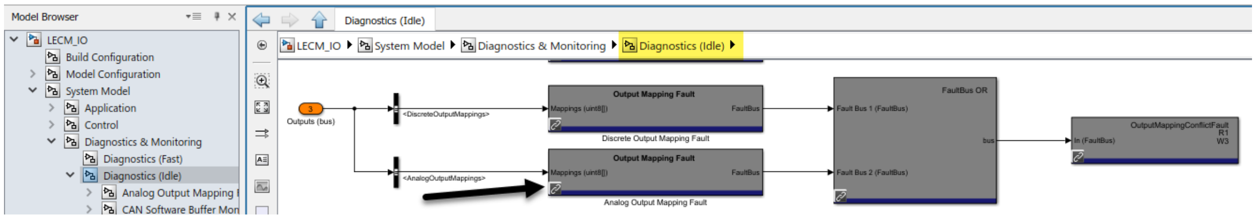


Figure 8-45. Analog Output Mapping Conflict Fault Detection Logic

Chapter 9. Product Support and Service Options

Product Support Options

If you are experiencing problems with the installation, or unsatisfactory performance of a Woodward product, the following options are available:

1. Consult the troubleshooting guide in the manual.
2. Contact the **OE Manufacturer or Packager** of your system.
3. Contact the **Woodward Business Partner** serving your area.
4. Contact Woodward technical assistance via email (**EngineHelpDesk@Woodward.com**) with detailed information on the product, application, and symptoms. Your email will be forwarded to an appropriate expert on the product and application to respond by telephone or return email.
5. If the issue cannot be resolved, you can select a further course of action to pursue based on the available services listed in this chapter.

OEM or Packager Support: Many Woodward controls and control devices are installed into the equipment system and programmed by an Original Equipment Manufacturer (OEM) or Equipment Packager at their factory. In some cases, the programming is password-protected by the OEM or packager, and they are the best source for product service and support. Warranty service for Woodward products shipped with an equipment system should also be handled through the OEM or Packager. Please review your equipment system documentation for details.

Woodward Business Partner Support: Woodward works with and supports a global network of independent business partners whose mission is to serve the users of Woodward controls, as described here:

- A **Full-Service Distributor** has the primary responsibility for sales, service, system integration solutions, technical desk support, and aftermarket marketing of standard Woodward products within a specific geographic area and market segment.
- An **Authorized Independent Service Facility (AISF)** provides authorized service that includes repairs, repair parts, and warranty service on Woodward's behalf. Service (not new unit sales) is an AISF's primary mission.
- A **Recognized Engine Retrofitter (RER)** is an independent company that does retrofits and upgrades on reciprocating gas engines and dual-fuel conversions, and can provide the full line of Woodward systems and components for the retrofits and overhauls, emission compliance upgrades, long term service contracts, emergency repairs, etc.

A current list of Woodward Business Partners is available at www.woodward.com/local-partner.

Product Service Options

Depending on the type of product, the following options for servicing Woodward products may be available through your local Full-Service Distributor or the OEM or Packager of the equipment system.

- Replacement/Exchange (24-hour service)
- Flat Rate Repair
- Flat Rate Remanufacture

Replacement/Exchange: Replacement/Exchange is a premium program designed for the user who is in need of immediate service. It allows you to request and receive a like-new replacement unit in minimum time (usually within 24 hours of the request), providing a suitable unit is available at the time of the request, thereby minimizing costly downtime.

This option allows you to call your Full-Service Distributor in the event of an unexpected outage, or in advance of a scheduled outage, to request a replacement control unit. If the unit is available at the time of the call, it can usually be shipped out within 24 hours. You replace your field control unit with the like-new replacement and return the field unit to the Full-Service Distributor.

Flat Rate Repair: Flat Rate Repair is available for many of the standard mechanical products and some of the electronic products in the field. This program offers you repair service for your products with the advantage of knowing in advance what the cost will be.

Flat Rate Remanufacture: Flat Rate Remanufacture is very similar to the Flat Rate Repair option, with the exception that the unit will be returned to you in “like-new” condition. This option is applicable to mechanical products only.

Returning Equipment for Repair

If a control (or any part of an electronic control) is to be returned for repair, please contact your Full-Service Distributor in advance to obtain Return Authorization and shipping instructions.

When shipping the item(s), attach a tag with the following information:
- Return number
- Name and location where the control is installed
- Name and phone number of contact person
- Complete Woodward part number(s) and serial number(s)
- Description of the problem
- Instructions describing the desired type of repair

Packing a Control

Use the following materials when returning a complete control:
- Protective caps on any connectors
- Antistatic protective bags on all electronic modules
- Packing materials that will not damage the surface of the unit
- At least 100 mm (4 inches) of tightly packed, industry-approved packing material
- A packing carton with double walls
- A strong tape around the outside of the carton for increased strength

NOTICE

To prevent damage to electronic components caused by improper handling, read and observe the precautions in Woodward manual 82715, *Guide for Handling and Protection of Electronic Controls, Printed Circuit Boards, and Modules*.

Replacement Parts

When ordering replacement parts for controls, include the following information:
- The part number(s) (XXXX-XXXX) that is on the enclosure nameplate
- The unit serial number, which is also on the nameplate.

Engineering Services

Woodward's Full-Service Distributors offer various Engineering Services for our products. For these services, you can contact the Distributor by telephone or by email.

- Technical Support
- Product Training
- Field Service

Technical Support is available from your equipment system supplier, your local Full-Service Distributor, or from many of Woodward's worldwide locations, depending upon the product and application. This service can assist you with technical questions or problem solving during the normal business hours of the Woodward location you contact.

Product Training is available as standard classes at many Distributor locations. Customized classes are also available, which can be tailored to your needs and held at one of our Distributor locations or at your site. This training, conducted by experienced personnel, will assure that you will be able to maintain system reliability and availability.

Field Service engineering on-site support is available, depending on the product and location, from one of our Full-Service Distributors. The field engineers are experienced both on Woodward products as well as on much of the non-Woodward equipment with which our products interface.

For information on these services, please contact one of the Full-Service Distributors listed at www.woodward.com/local-partner.

Contacting Woodward's Support Organization

For the name of your nearest Woodward Full-Service Distributor or service facility, please consult our worldwide directory at www.woodward.com/support, where you may also find the most current product support and contact information.

You can also contact the Woodward Customer Service Department at one of the following Woodward facilities to obtain the address and phone number of the nearest facility at which you can obtain information and service.

Products Used in Electrical Power Systems

Facility	Phone Number
Brazil	+55 (19) 3708 4800
China	+86 (512) 8818 5515
Germany:	+49 (711) 78954-510
India	+91 (124) 4399500
Japan	+81 (43) 213-2191
Korea	+82 (32) 422-5551
Poland	+48 (12) 295 13 00
United States	+1 (970) 482-5811

Products Used in Engine Systems

Facility	Phone Number
Brazil	+55 (19) 3708 4800
China	+86 (512) 8818 5515
Germany	+49 (711) 78954-510
India	+91 (124) 4399500
Japan	+81 (43) 213-2191
Korea	+ 82 (32) 422-5551
The Netherlands	+31 (23) 5661111
United States	+1 (970) 482-5811

Products Used in Industrial Turbomachinery Systems

Facility	Phone Number
Brazil	+55 (19) 3708 4800
China	+86 (512) 8818 5515
India	+91 (124) 4399500
Japan	+81 (43) 213-2191
Korea	+ 82 (32) 422-5551
The Netherlands	+31 (23) 5661111
Poland	+48 (12) 295 13 00
United States	+1 (970) 482-5811

Technical Assistance

If you need to contact technical assistance, you will need to provide the following information. Please write it down here before contacting the Engine OEM, the Packager, a Woodward Business Partner, or the Woodward factory:

General	
Your Name	
Site Location	
Phone Number	
Fax Number	
Prime Mover Information	
Manufacturer	
Engine Model Number	
Number of Cylinders	
Type of Fuel (gas, gaseous, diesel, dual-fuel, etc.)	
Power Output Rating	
Application (power generation, marine, etc.)	
Control/Governor Information	
Control/Governor #1	
Woodward Part Number & Rev. Letter	
Control Description or Governor Type	
Serial Number	
Control/Governor #2	
Woodward Part Number & Rev. Letter	
Control Description or Governor Type	
Serial Number	
Control/Governor #3	
Woodward Part Number & Rev. Letter	
Control Description or Governor Type	
Serial Number	
Symptoms	
Description	

If you have an electronic or programmable control, please have the adjustment setting positions or the menu settings written down and with you at the time of the call.

Revision History

Changes in Revision —

- New manual