Dataspeed Drive-By-Wire

dSPACE/Simulink Interface Blockset

User Manual

Documentation of the dSPACE Simulink blocks that interface with the Dataspeed combination drive-by-wire module.

Point of Contact:

support@dataspeedinc.com

Contents

1. Introduction

This document describes the dSPACE Simulink blockset designed to communicate with the Dataspeed combination Drive-By-Wire (DBW) hardware modules over CAN. For more details regarding the specific CAN messaging interface to the DBW hardware modules, please refer to the individual product datasheets (ThrottleBrakeDatasheet-RevA10.pdf and SteeringShifterDatasheet-RevA14.pdf).

The main Simulink library **drive_by_wire_master.slx** is shown in Figure 1. This library has the following components:

- **CAN Controller Setup** – A pre-configured block that properly sets up the dSPACE RTICAN interface to send messages to and from the Dataspeed DBW modules. All the dSPACE RTICAN blocks are pre-configured to use the **DataspeedByWire.dbc** file to automatically configure the message structure.

  IMPORTANT: When the CAN configuration block is placed in a new model, be sure to add the DBC file under the **Data File Support** tab. This assures that the correct path to the DBC file is set. The DBC file can be found in the **lib** folder in the Dataspeed DBW release package.

- **Drive-By-Wire Interface Blocks** – Double-clicking this block opens the library containing all DBW subsystem input/output interfaces, including throttle, brake, steering, shifting and turn signaling.

- **Vehicle Data Blocks** – Double-clicking this block opens the library containing CAN interfaces to vehicle data collected from OEM CAN messaging.

- **Vehicle Control Blocks** – Double-clicking this block opens the library containing blocks that provide basic control functionality over the vehicle.

- **ADAS Kit Info Blocks** – Double-clicking this block opens the library containing blocks that provide information on the ADAS Kit (firmware versions, etc.) and static vehicle information such as the VIN and platform type.

- **Demo Model** – This Simulink model demonstrates the usage of the DBW interface blocks and provides baselines from which to expand.

  Double-clicking the square box next to the model name will open a dialog box, shown in Figure 2.

  The options are:

  – **Open** – Open the demo model at its default location. It is not recommended to modify the model in this case.

  – **Copy** – This option copies the demo model into the current MATLAB working directory. Making changes to this copy will not affect the original demo model.

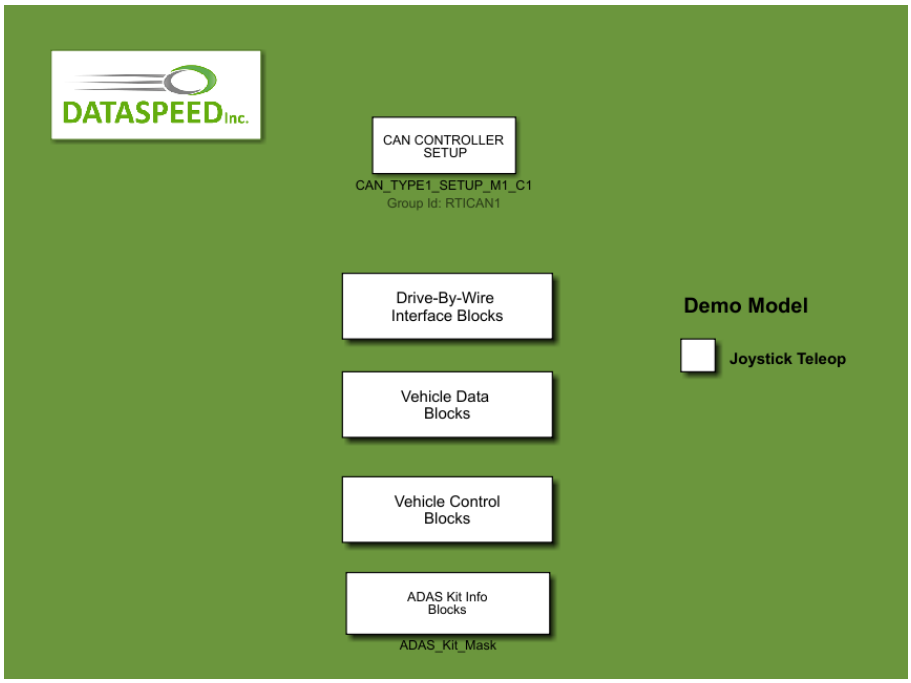


Figure 1. Main Simulink library.

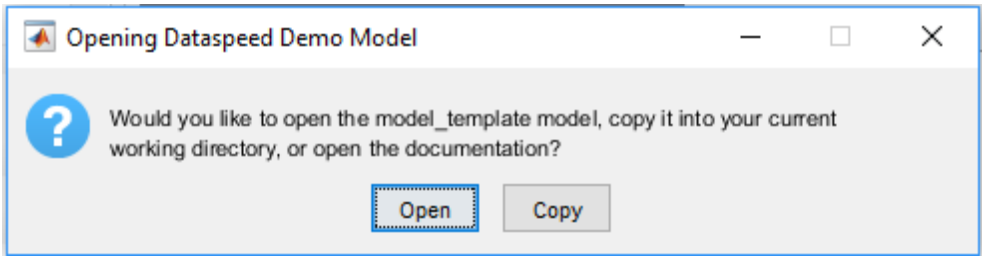


Figure 2. Demo Model dialog box.

2. Software Installation

The process of installing the DBW Simulink blockset into MATLAB is as follows.

1. Using File Explorer, copy the root DBW folder onto your hard drive in the place of your choosing.
2. Open MATLAB and change MATLAB's present working directory to be this newly added folder.
3. At the MATLAB Command Line, type "install" to run the DBW Blockset installation M-file script.
4. When prompted, hit "Enter" on your keyboard.
5. The installation script will add all appropriate DBW Simulink blockset folders to your MATLAB path.

3. Drive-by-Wire Interface Blocks

This section describes the individual DBW interface blocks. These individual blocks can be found in the **dbw_interface_blocks.slx** library, which can be opened from the main library. They are shown in Figure 3.

Figure 3. Drive-By-Wire Interface Blockset.

3.1. CAN Controller Setup

The path to the DBC file that defines CAN message structures, is hardcoded. Current dSPACE software does not support using relative paths. Therefore, after installing the Dataspeed dSPACE/Simulink Blockset library onto your PC, you must open the CAN CONTROLLER SETUP block, go to the “Data File Support” tab in the dialog box that appears. There, manually update the path by clicking “Replace Data File”, then navigating to, and selecting the file named “DataspeedByWire.dbc” in the “…/dspace/lib” subdirectory created on their PC during the installation process.

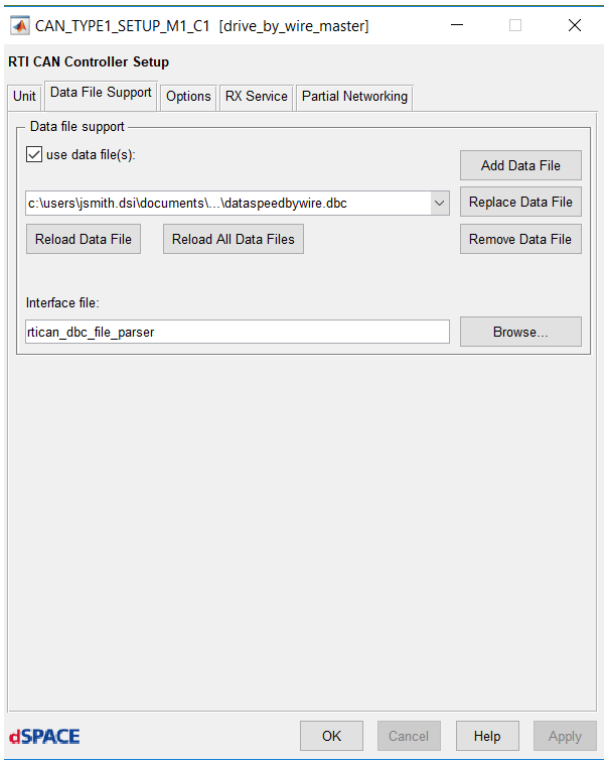
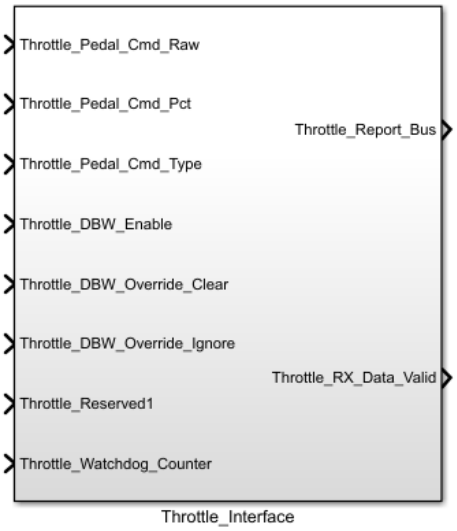


Figure 4. Setting the path to the CAN DBC file.

3.2. Throttle

The **Throttle Interface** block is shown in Figure 5, and its I/O is described in

Table 1. This block packages pre-configured dSPACE RTICAN blocks to transmit the throttle command CAN message 0x62, and receive the throttle report CAN message 0x63).



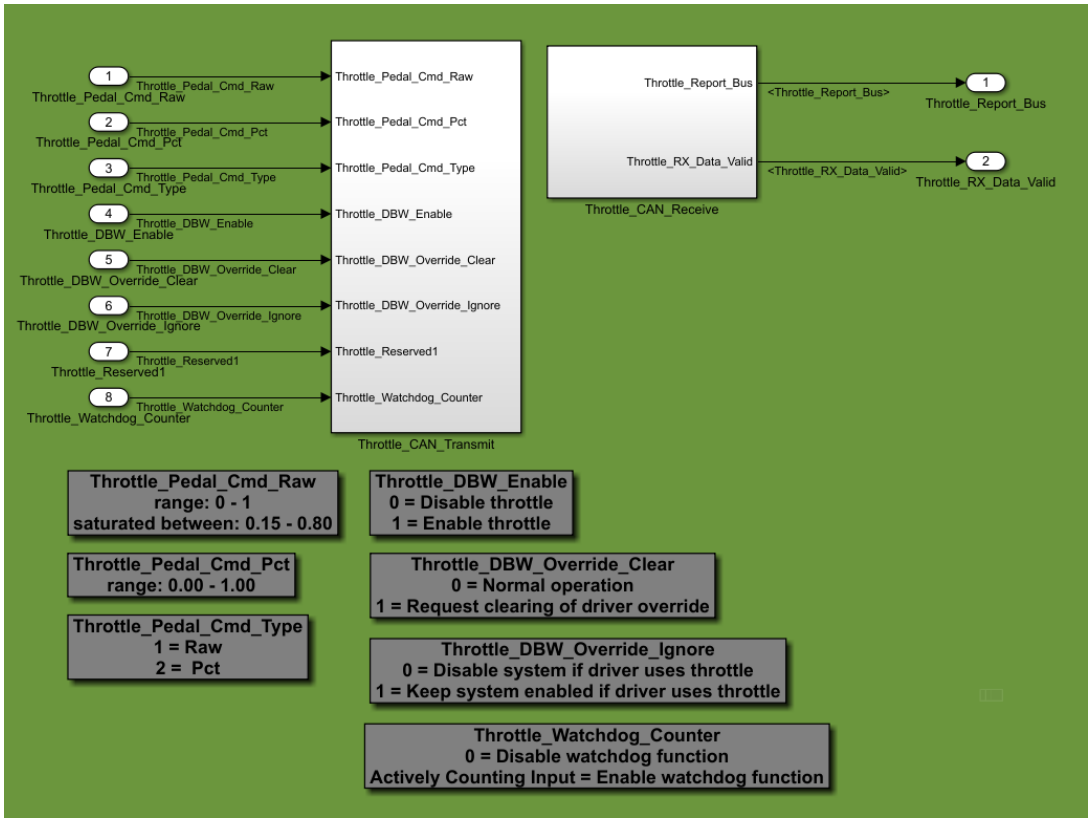


Figure 5. **Throttle_Interface** block.

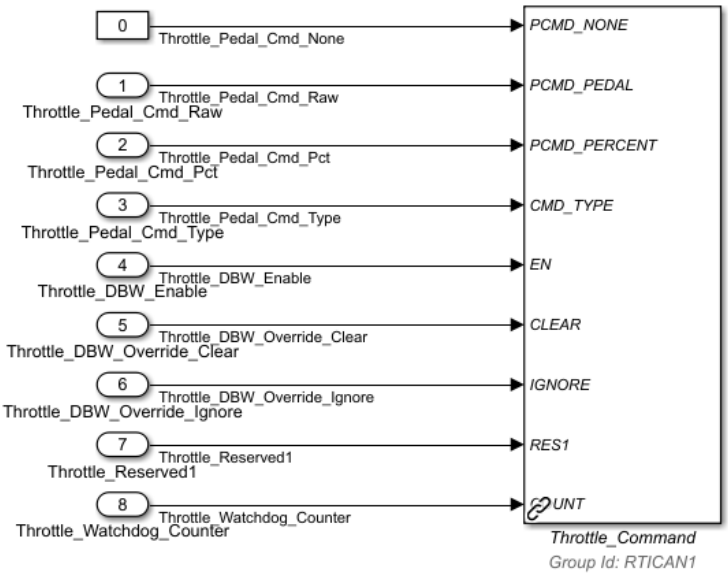


Figure 6. **Throttle_CAN_Transmit** block.

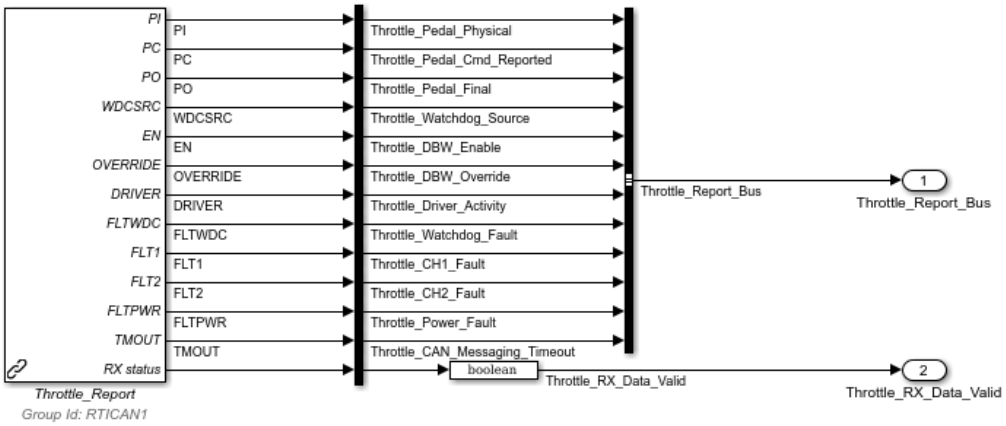


Figure 7. **Throttle_CAN_Receive** block.

Table 1. Throttle Interface Library block I/O.

	Port Name	Data Type	Range	Description
Inputs	Throttle_Pedal_Cmd_Raw	double	0.15 — 0.8	Throttle pedal command, as a direct PWM duty cycle. Values outside the range will be saturated.
	Throttle_Pedal_Cmd_Pct	double	0.00 — 1.00	Throttle pedal command, normalized from closed throttle to WOT. Values outside the range will be saturated.
	Throttle_Pedal_Cmd_Type	uint8	—	Enumeration to select which *Throttle_Pedal_Cmd* to use. 1: *Throttle_Pedal_Cmd_Raw* 2: *Throttle_Pedal_Cmd_Pct*
	Throttle_DBW_Enable	bool	—	Enable DBW throttle control. TRUE: enable FALSE: disable.
	Throttle_DBW_Clear	bool	—	Clear driver override. TRUE: request clear of driver override FALSE: normal operation.
	Throttle_DBW_Override_Ignore	bool	—	Ignore future driver overrides TRUE: ignore overrides FALSE: normal operation
	Throttle_Watchdog_Counter	uint8	0 — 255	Optional watchdog counter. 0: disables feature. See datasheet (ThrottleBrakeDatasheet-RevA10.pdf) for details
Outputs	Throttle_Report_Bus	bus	—	Simulink bus containing the data in the throttle report CAN message. See datasheet (ThrottleBrakeDatasheet-RevA10.pdf) for details.
	Throttle_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

3.3. Brake

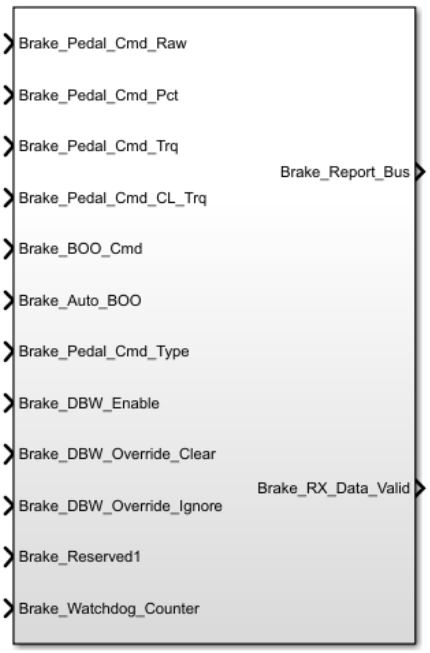
The **Brake_Interface** block is shown in Figure 8, and its I/O is described in Table 2. This block packages pre-configured dSPACE RTICAN blocks to transmit the brake command CAN message 0x60, and receive the brake report CAN message (ID = 0x61). Additionally, this block implements logic to control the **BOO** signal based on user preferences.

The **Brake_CAN_Transmit** and **Brake_CAN_Receive** subsystem blocks are shown in Figure 9 and Figure 10, respectively.

Table 2. Brake Interface Library block I/O.

	Port Name	Data Type	Range	Description
Inputs	Brake_Pedal_Cmd_Raw	double	0.15 — 0.5	Brake pedal command, as a PWM duty cycle. Values outside the range will be saturated.
	Brake_Pedal_Cmd_Pct	bool	0.00 — 1.00	Brake pedal command, as normalized torque. Values outside the range will be saturated.
	Brake_Pedal_Cmd_Trq	bool	0 — 3412	Brake pedal command, as torque (Nm). Values outside the range will be saturated.
	Brake_Pedal_Cmd_CL_Trq	bool	0 — 3412	Brake pedal command, as torque (Nm), closing the loop on reported requested torque. Values outside the range will be saturated.
	Brake_Pedal_Cmd_Type	unit8	1 —4	Enumeration to select which *Brake_Pedal_Cmd* to use. 1: Brake_*Pedal_Cmd_Raw* 2: *Brake_Pedal_Cmd_Pct* 3: *Brake_Pedal_Cmd_Trq* 4: *Brake_Pedal_Cmd_CL_Trq*
	Brake_BOO_Cmd	bool	—	Brake Light On/Off command. TRUE: On; FALSE: Off. **NOTE:** *Brake_Auto_BOO* must be set to FALSE for user to have full control over Brake Lights On/Off.
	Brake_Auto_BOO	bool	—	Enable automatic control over Brake Lights On/Off. TRUE: enable; FALSE: disable. **NOTE:** *Brake_BOO_Cmd* TRUE overrides *Brake_Auto_BOO*.
	Brake_DBW_Enable	bool	—	Enable DBW brake control. TRUE: enable, FALSE: disable.
	Brake_DBW_Override_Clear	bool	—	Clear driver override. TRUE: request clear of driver override. FALSE: normal operation.
	Brake_DBW_Override_Ignore	bool	—	Ignore future driver overrides TRUE: ignore overrides FALSE: normal operation
	Brake_Reserved1	bool	—	Strictly for internal Dataspeed use.
	Brake_Watchdog_Counter	uint8	0 — 255	Optional watchdog counter. 0: disables feature. See datasheet (ThrottleBrakeDatasheet-RevA10.pdf) for details
Outputs	Brake_Report_Bus	bus	—	Simulink bus containing the data in the brake report CAN message.

			See datasheet (ThrottleBrakeDatasheet-RevA10.pdf) for details.
Brake_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.



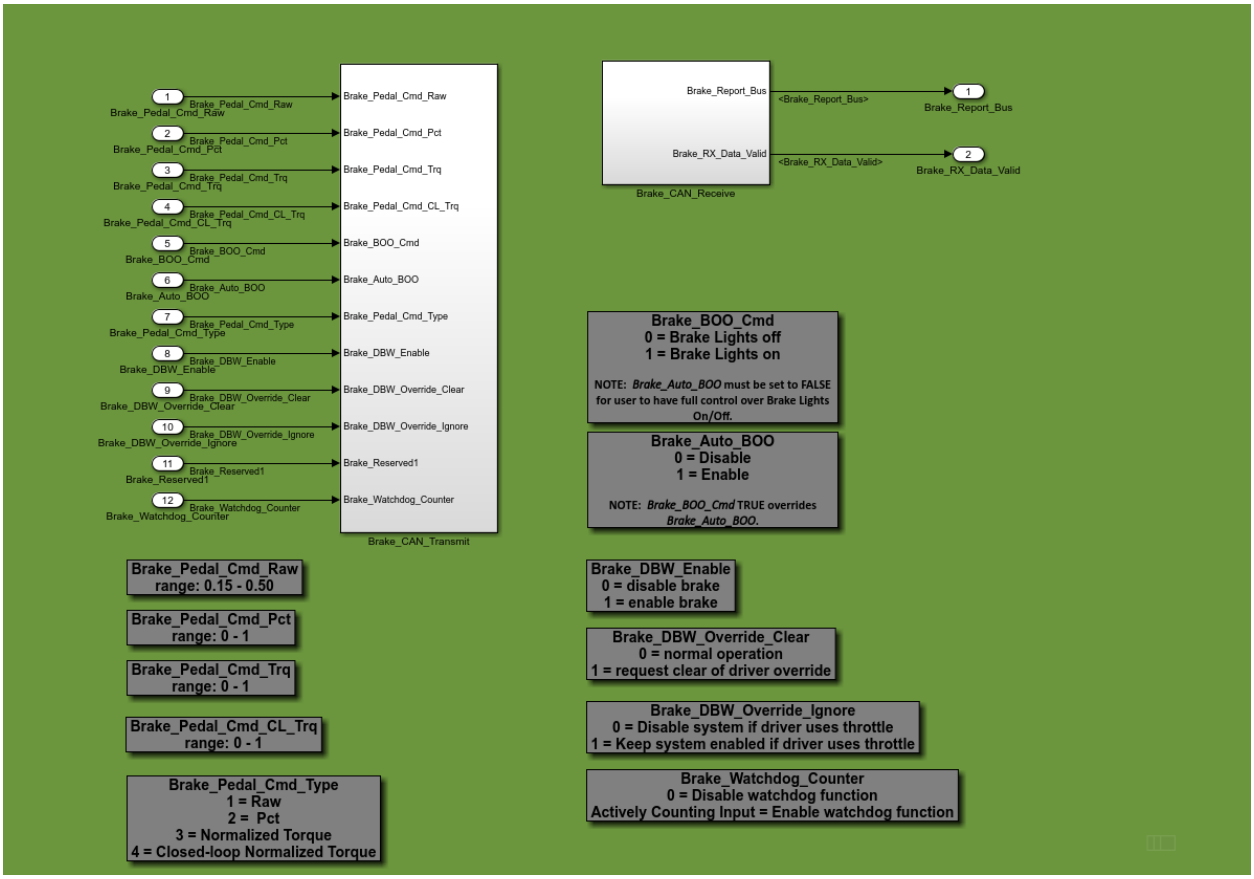


Figure 8. **Brake_Interface** block.

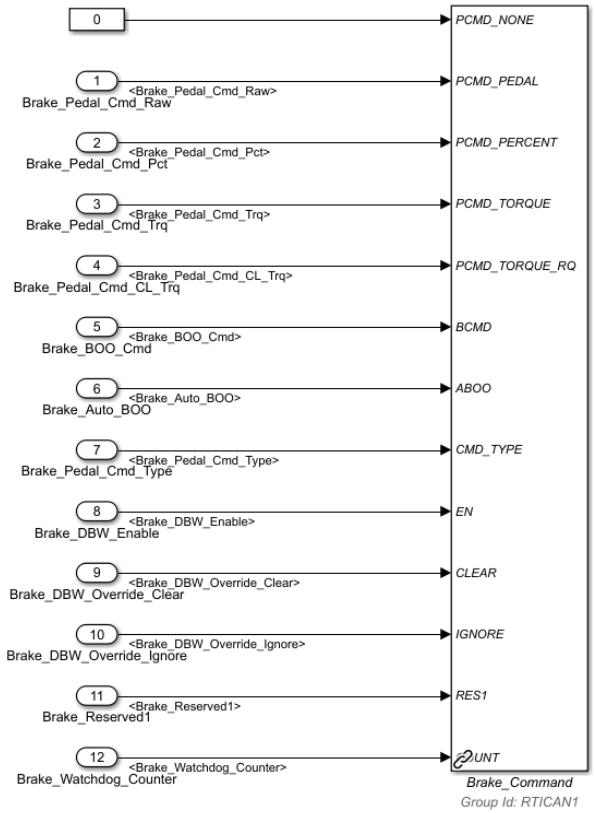


Figure 9. **Brake_CAN_Transmit** block.

.

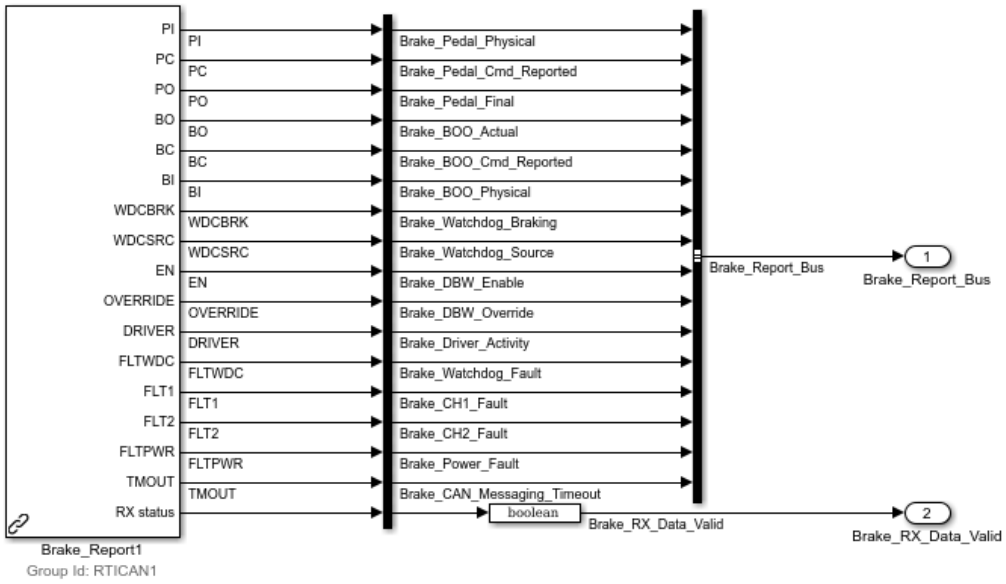
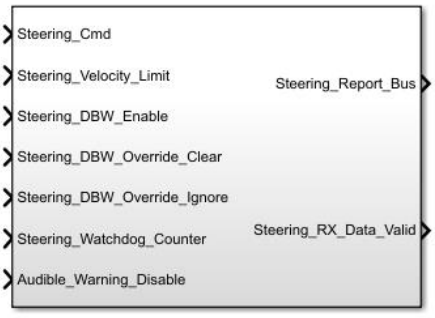


Figure 10. **Brake_CAN_Receive** block.

3.4. Steering

The **Steering_Interface** block is shown in Figure 11, and its I/O is described in Table 3. This block packages pre-configured dSPACE RTICAN blocks to transmit the steering command CAN message 0x64, and receive the steering report CAN message 0x65.

The **Steering_CAN_Transmit** and **Steering_CAN_Receive** subsystem blocks are shown in Figure 12 and Figure 13, respectively.



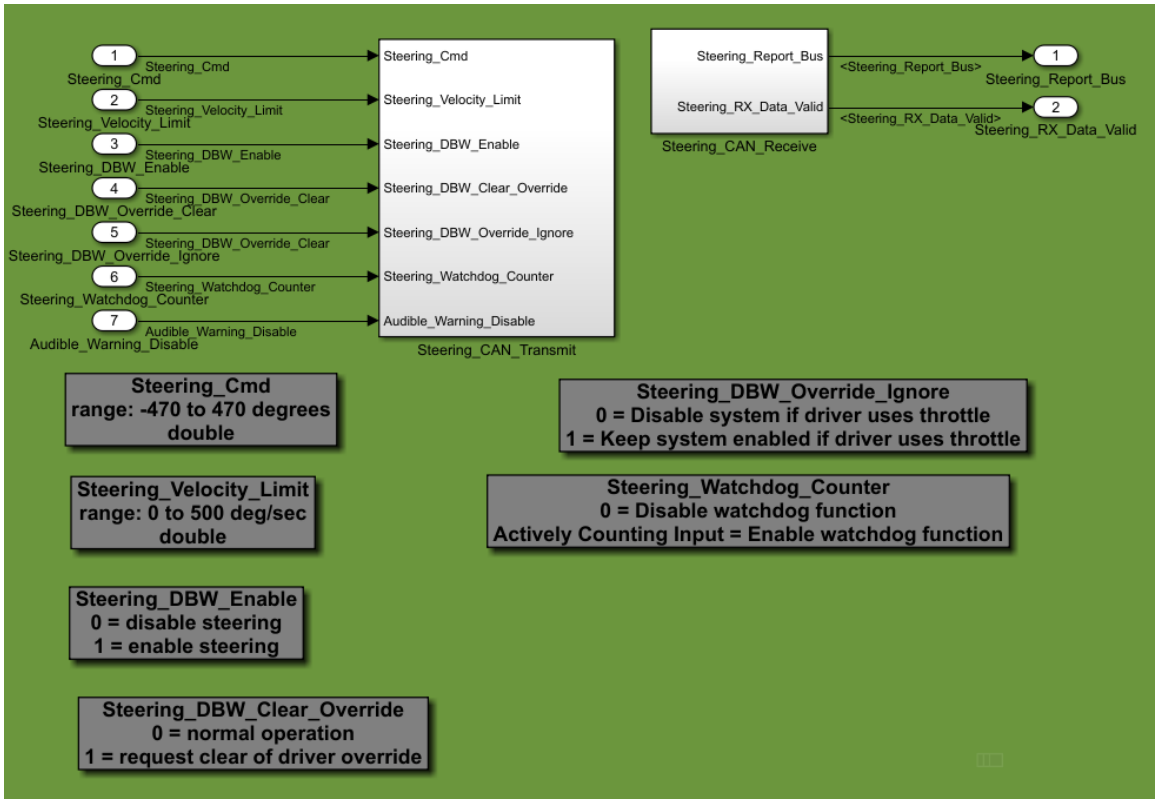


Figure 11. **Steering_Interface** block.

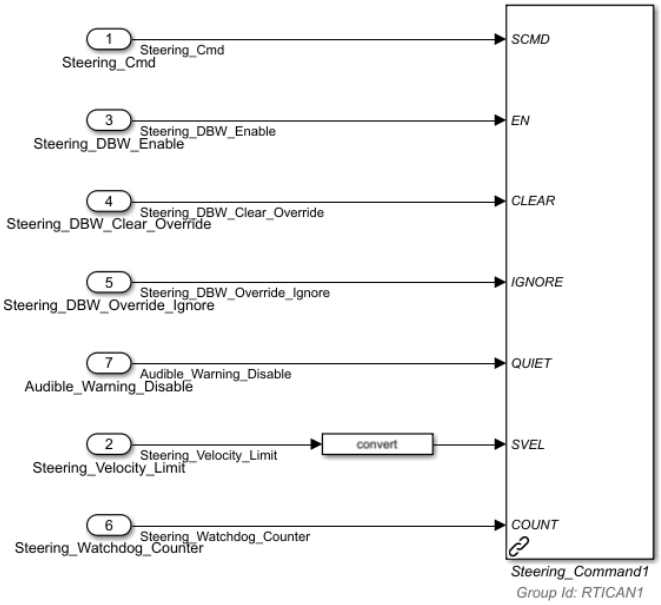


Figure 12. **Steering_CAN_Transmit** block.

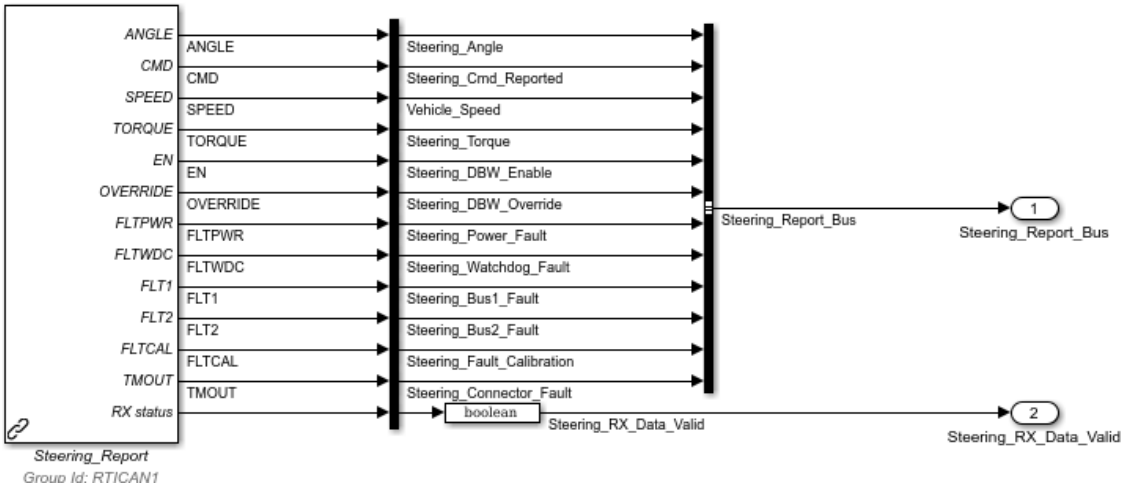


Figure 13. **Steering_CAN_Receive** block.

Table 3. **Steering_Interface** library block I/O.

	Port Name	Data Type	Range	Description
Inputs	Steering_Cmd	double	-470 — 470	Steering wheel angle command in degrees. Values outside the range will be saturated.
	Steering_Velocity		0 – 500	Maximum velocity in *deg/sec* to use while moving steering wheel to desired angle. Setting to zero will default to maximum of 500 *deg/sec*.
	Steering_DBW_Enable	bool	—	Enable DBW steering control. TRUE: enable FALSE: disable.
	Steering_DBW_Override_Clear	bool	—	Clear driver override. TRUE: request clear of driver override. FALSE: normal operation.
	Steering_DBW_Override_Ignore	bool	—	Ignore future driver overrides TRUE: ignore overrides FALSE: normal operation
	Steering_Watchdog_Counter	uint8	0 — 255	Optional watchdog counter. 0: disables feature. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
Outputs	Steering_Report_Bus	bus	—	Simulink bus containing the data in the steering report CAN message. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
	Steering_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

3.5. Shifter

The **Shifter_Interface** block is shown in Figure 14, and its I/O is described in Table 4. This block packages pre-configured dSPACE RTICAN blocks to transmit the gear command CAN message 0x66 and receive the gear report CAN message 0x67.

The **Gear_CAN_Transmit** and **Gear_CAN_Receive** subsystem blocks are shown in Figure 15 and Figure 16, respectively.

The transmit block does the following:

- Passes the gear command into the **GCMD** filed of the gear command CAN message.
- Passes the clear signal to the **CLEAR** bit of the gear command CAN message.

- If the gear command is equal to zero, the hardware module will ignore the command.
- If the command is between 1 and 5, then the corresponding gear is selected.

The receive block does the following:

- Parses the gear report message and combines the received data into a Simulink bus.
- The **CAN_RX_Data_Valid** flag is used to indicate if the report message is being received.

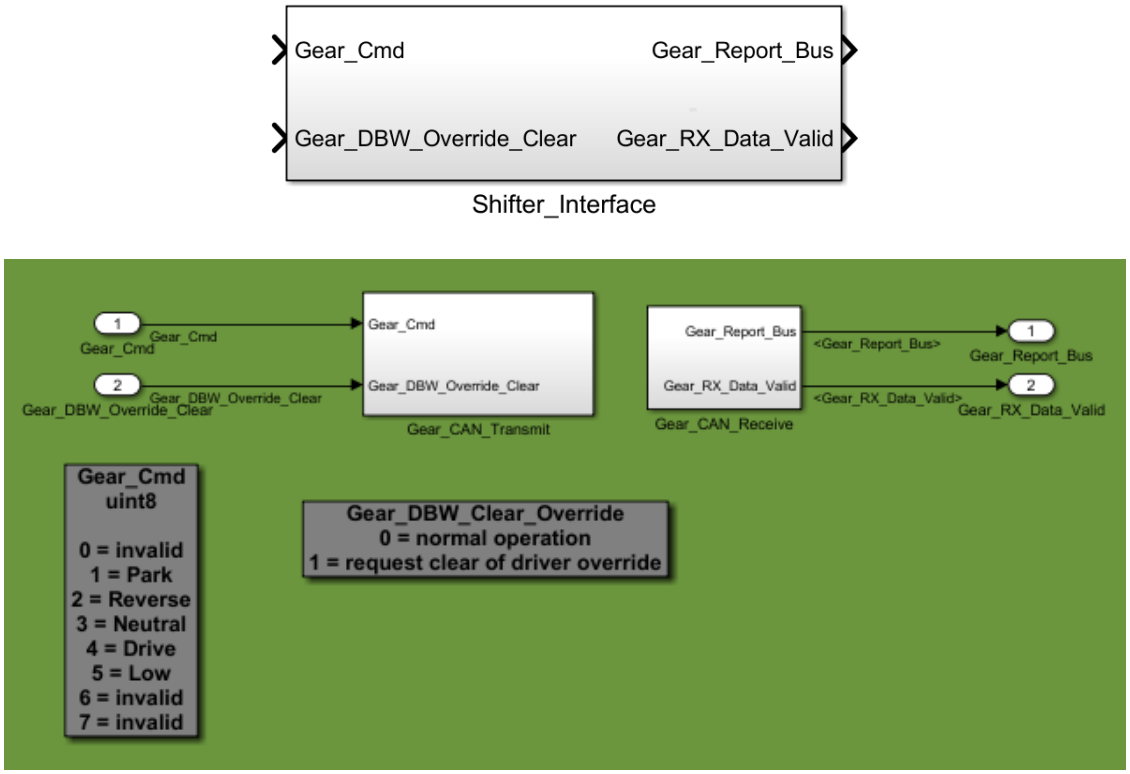


Figure 14. **Shifter_Interface** block.

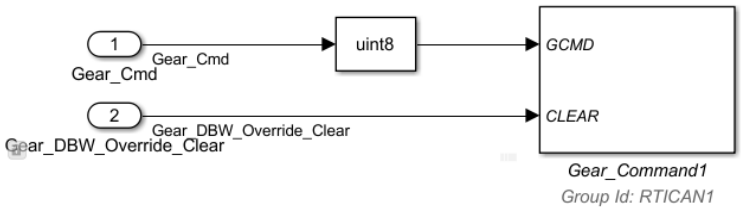


Figure 15. **Gear_CAN_Transmit** block.

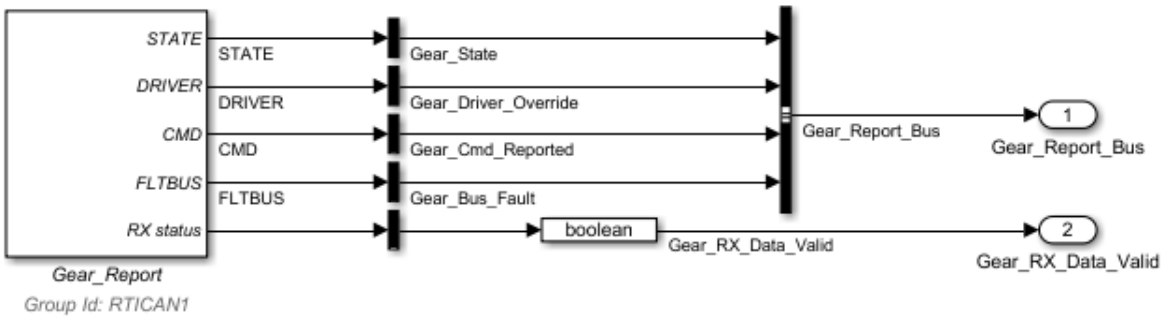


Figure 16. **Gear_CAN_Receive** block.

Table 4. **Shifter_Interface** library block I/0.

	Port Name	Data Type	Range	Description
Inputs	Gear_Cmd	uint8	0 — 5	Enumeration of desired shifter position. 0: None 1 — 5: P, R, N, D, L, respectively.
	Clear	bool	—	Clear driver override. TRUE: request clear of driver override. FALSE: normal operation.
Outputs	Report	bus	—	Simulink bus containing the data in the gear report CAN message. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
	Gear_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

3.6. Turn Signal

The **Turn_Signal_Interface** block is shown in Figure 17, and its I/O is described in Table 5. This block packages a pre-configured dSPACE RTICAN block to transmit the turn signal command CAN message 0x68. The turn signal report status can be accessed in the **Misc_CAN_Receive** block. See Section 4.7 for details.

The **Signal_CAN_Transmit** block is shown in Figure 18.

The transmit block does the following:

- Passes the turn signal command into the **TRNCMD** field of the turn signal command CAN message.
- If the command is 0, all signals turn off.
- If the command is 1, the left signal turns on.
- If the command is 2, the right signal turns on.

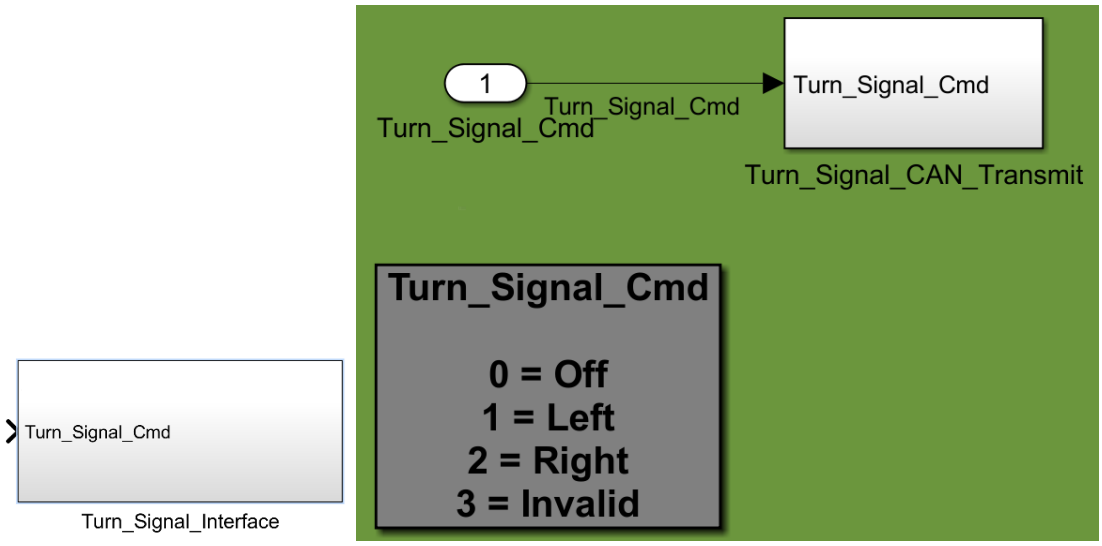


Figure 17. **Turn_Signal_Interface** block

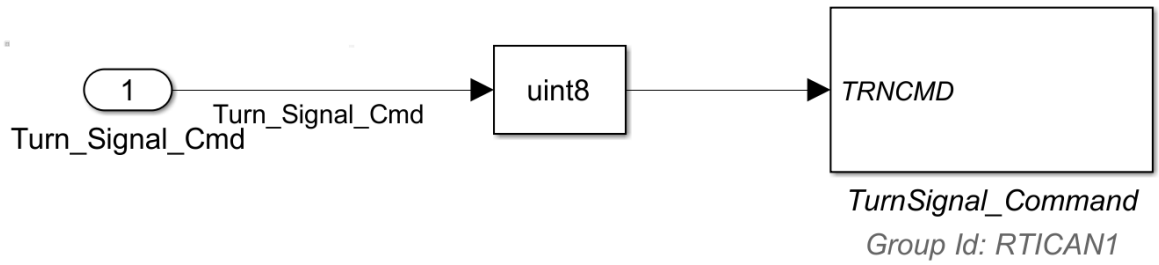


Figure 18. **Signal_CAN_Transmit** block

Table 5. **Turn_Signal_Interface** library block I/O.

	Port Name	Data Type	Range	Description
Inputs	Signal_Cmd	uint8	0 — 2	Enumeration of desired shifter position. 0: None 1: Left 2: Right

4. Vehicle Data Blocks

The DBW hardware module listens to data that is available on the main vehicle CAN networks and re-transmits the data on the DBW CAN network.

The vehicle data that is re-transmitted includes:

- **Wheel Speed** – The four individual wheel speed measurements in *rad/s*.
- **Gyro** – The roll and yaw rate measurements in *rad/s*.
- **Acceleration** – Longitudinal, lateral and vertical acceleration measurements in m/s^2.
- **Misc Data** – Miscellaneous data containing turn signal, wiper, and high-beam status, as well as the state of many of the buttons on the steering wheel.

Blocks to access the re-transmitted data are provided in the **vehicle_data_blocks.slx** library, which is shown in Figure 19. This library can also be opened from the main **drive_by_wire_master.slx** library.

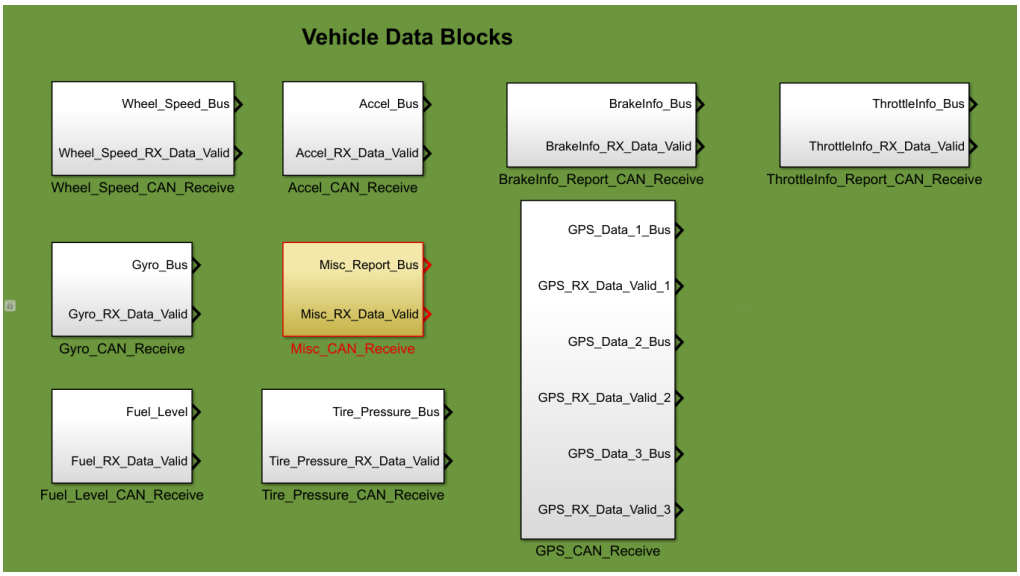


Figure 19. Vehicle Data Library.

4.1. Wheel Speed

The **Wheel_Speed_CAN_Receive** block is shown in Figure 20, and its I/O is described in Table 6. CAN message ID 0x6A.

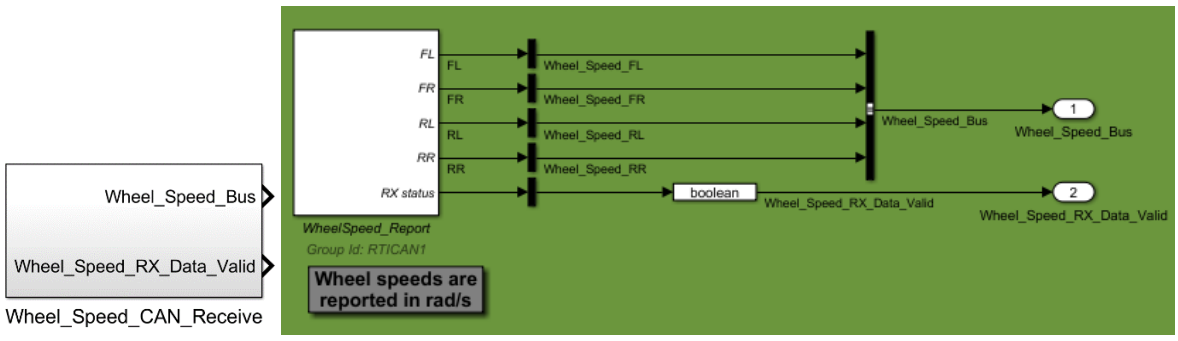


Figure 20. **Wheel_Speed_CAN_Receive** block.

Table 6. **Wheel_Speed_CAN_Receive** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Wheel_Speed_Bus	bus	—	Simulink bus containing the four individual wheel speeds in *rad/s*.
	Wheel_Speed_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.2. Gyro

The **Gyro_CAN_Receive** block is shown in Figure 21, and its I/O is described in Table 7. CAN message ID 0x6C.

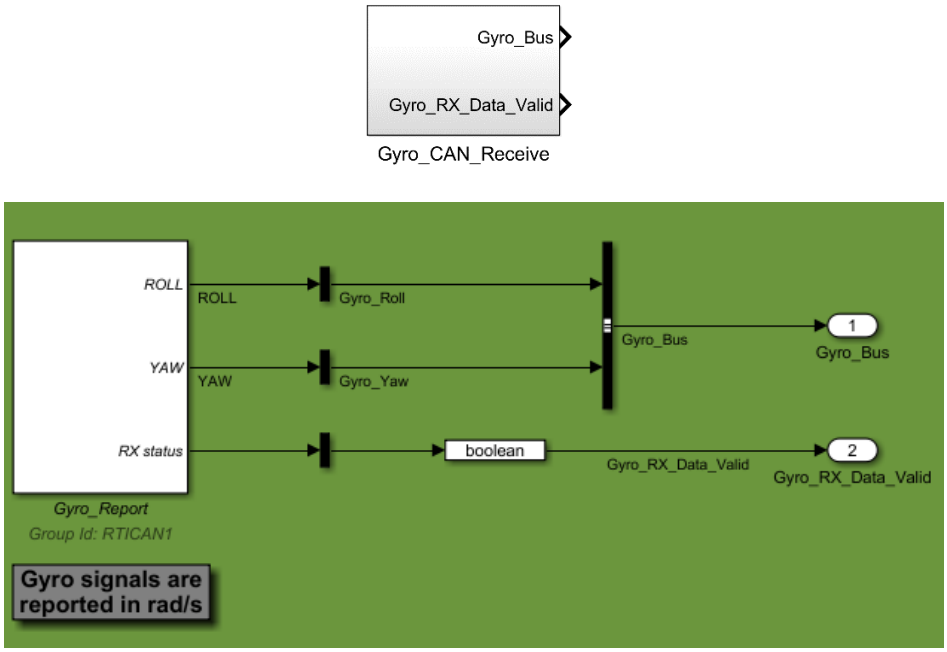


Figure 21. **Gyro_CAN_Receive** block.

Table 7. **Gyro_CAN_Receive** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Gyro_Bus	bus	—	Simulink bus containing the vehicle roll and yaw rates in *rad/s*.
	Gyro_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.3. Acceleration

The **Accel_CAN_Receive** block is shown in Figure 22, and its I/O is described in Table 8. CAN message ID 0x6B.

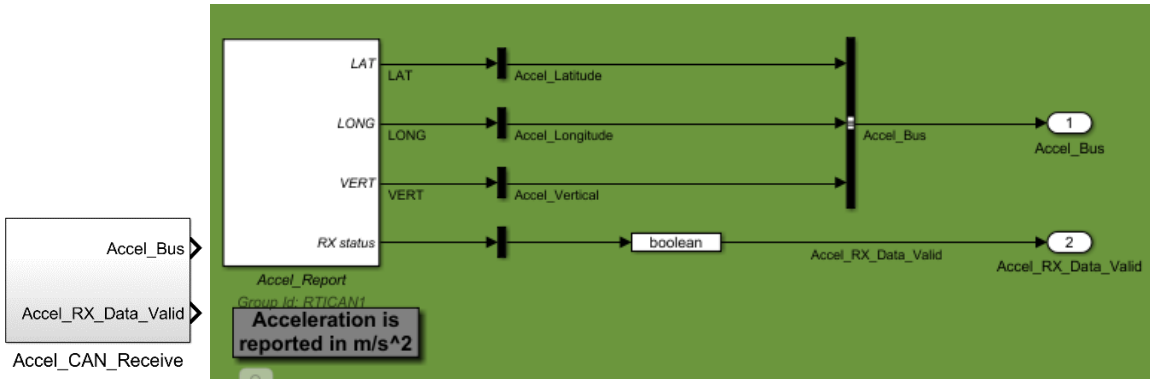


Figure 22. **Accel_CAN_Receive** block.

Table 8: **Accel_CAN_Receive** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Accel_Bus	bus	—	Simulink bus containing the longitudinal, lateral and vertical acceleration in m/s^2.
	Accel_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.4. Fuel Level

The **Fuel_Level_CAN_Receive** block is shown in Figure 23, and its I/O is described in Table 9. CAN message ID 0x72.

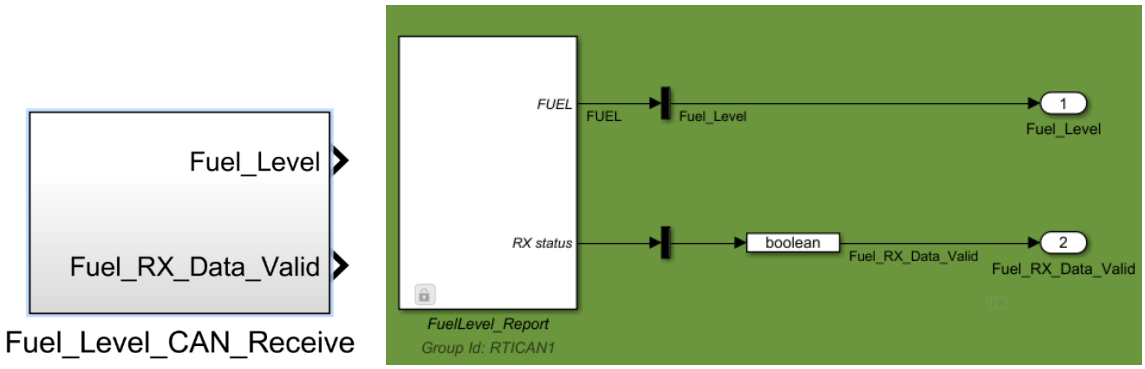


Figure 23. **Fuel_Level_CAN_Receive** block.

Table 9. Fuel level data library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Fuel_Level	double	0 — 100	Current fuel level percentage.
	Fuel_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.5. Tire Pressure

The **Tire_Pressure_CAN_Receive** block is shown in Figure 24, and its I/O is described in Table 10. CAN message ID 0x71.

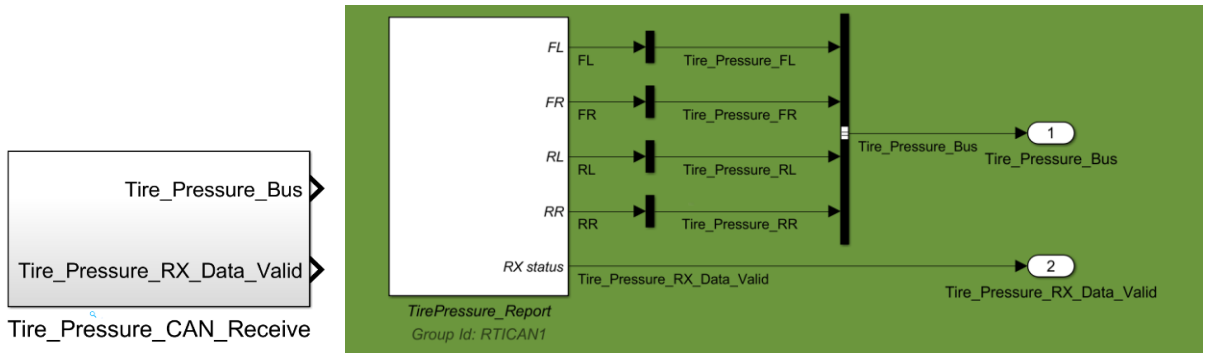


Figure 24. **Tire_Pressure_CAN_Receive** block.

Table 10. **Tire_Pressure_CAN_Receive** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Tire_Pressure_Bus	bus	0 — 65535	Simulink bus containing tire pressure for each tire in *kPa*.
	Tire_Pressure_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.6. GPS

The **GPS_CAN_Receive** block is shown in Figure 25, and its I/O is described in Table 11. CAN message ID's 0cx6D, 0cx6E and 0cx6F.

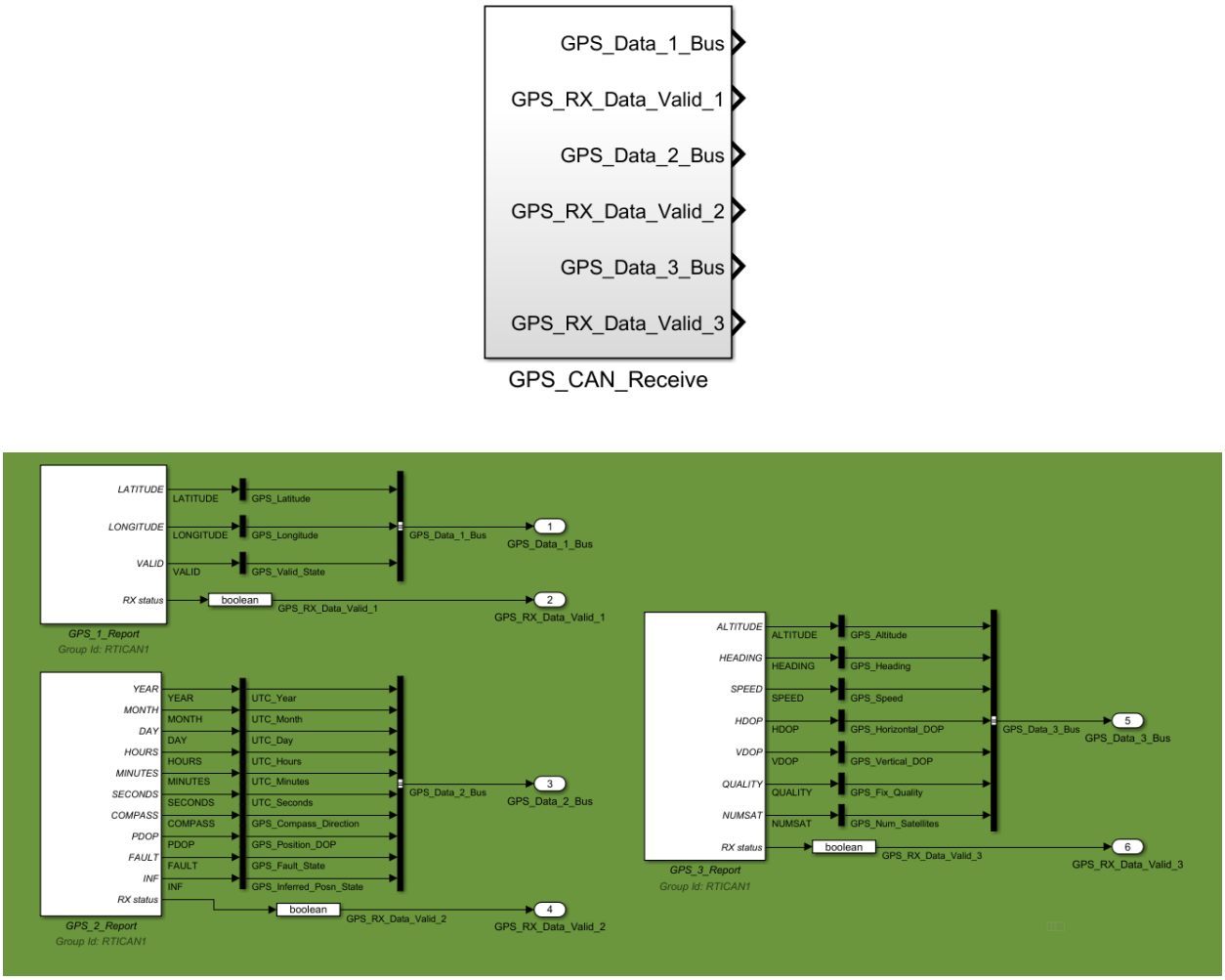


Figure 25. **GPS_CAN_Receive** block.

Table 11. **GPS_CAN_Receive** library block I/O.

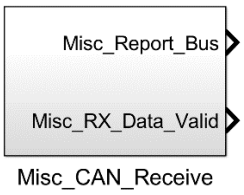
	Port Name	Data Type	Range	Description
Outputs	GPS_Data_1	bus	—	Simulink bus containing latitude and longitude data. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
	GPS_RX_Data_Valid_1	bool	—	Flag indicating if data is being received over the CAN network.
	GPS_Data_2	bus	—	Simulink bus containing GPS time stamp. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
	GPS_RX_Data_Valid_2	bool	—	Flag indicating if data is being received over the CAN network.
	GPS_Data_3	bus	—	Simulink bus containing altitude, heading, speed, and DOP values. See datasheet (SteeringShifterDatasheet-RevA14.pdf) for details.
	GPS_RX_Data_Valid_3	bool	—	Flag indicating if data is being received over the CAN network.

4.7. Miscellaneous Data

The **Misc_CAN_Receive** block is shown in Figure 26, and its I/O is described in Table 12. CAN message ID 0x69.

Table 12: **Misc_CAN_Receive** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Misc_Report_Bus	bus	—	Simulink bus containing the information in the Miscellaneous Report CAN message (ID = 0x69). See the Steering-Shifter module datasheet for details (SteeringShifterDatasheet-RevA14.pdf).
	Misc_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.



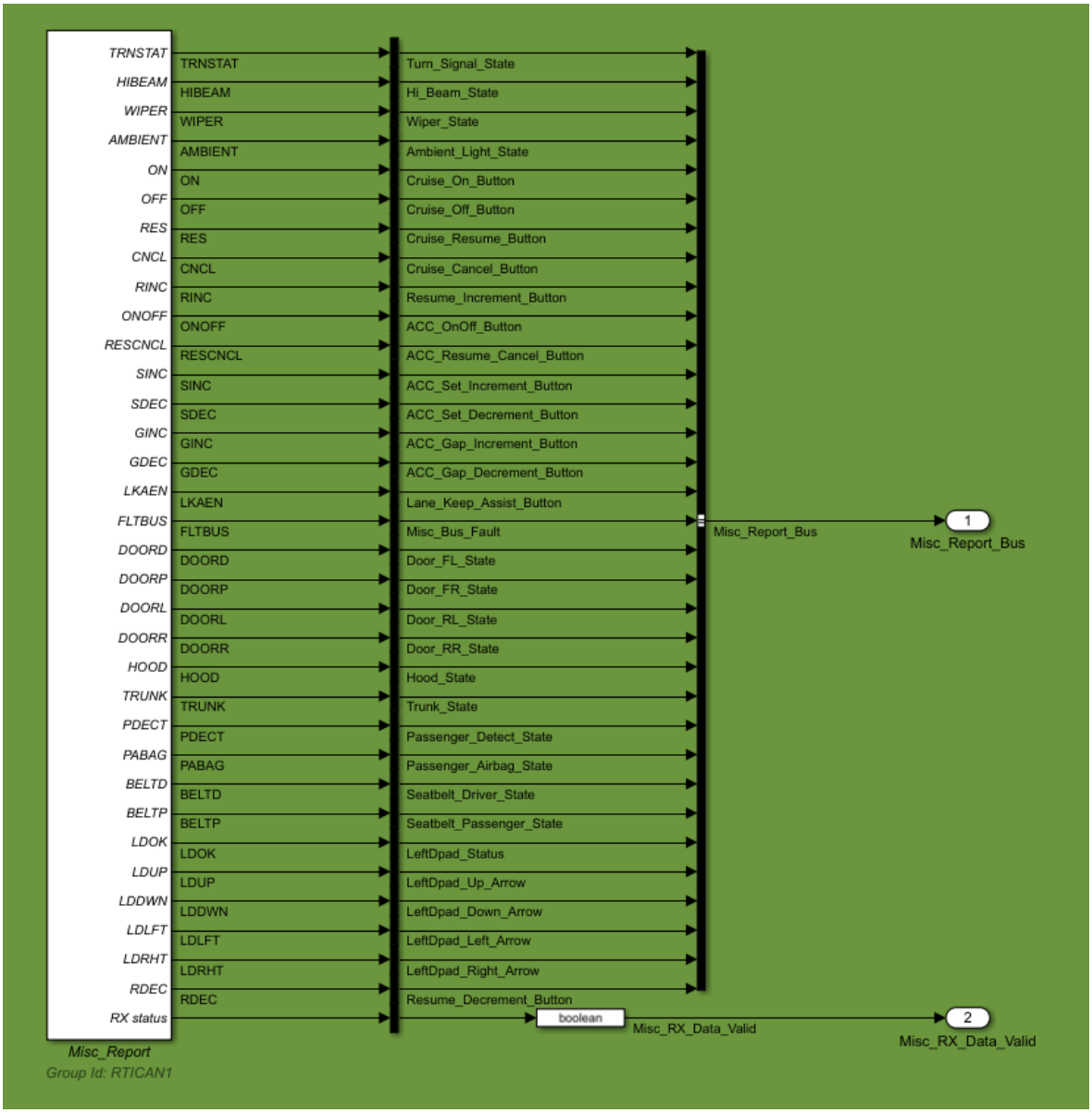


Figure 26. **Misc_CAN_Receive** block.

4.8. BrakeInfo Report Data

The **BrakeInfo_Report_CAN_Receive** block is shown in *Figure 27*, a*nd* its I/O is described in Table 13. CAN message ID 0x74.

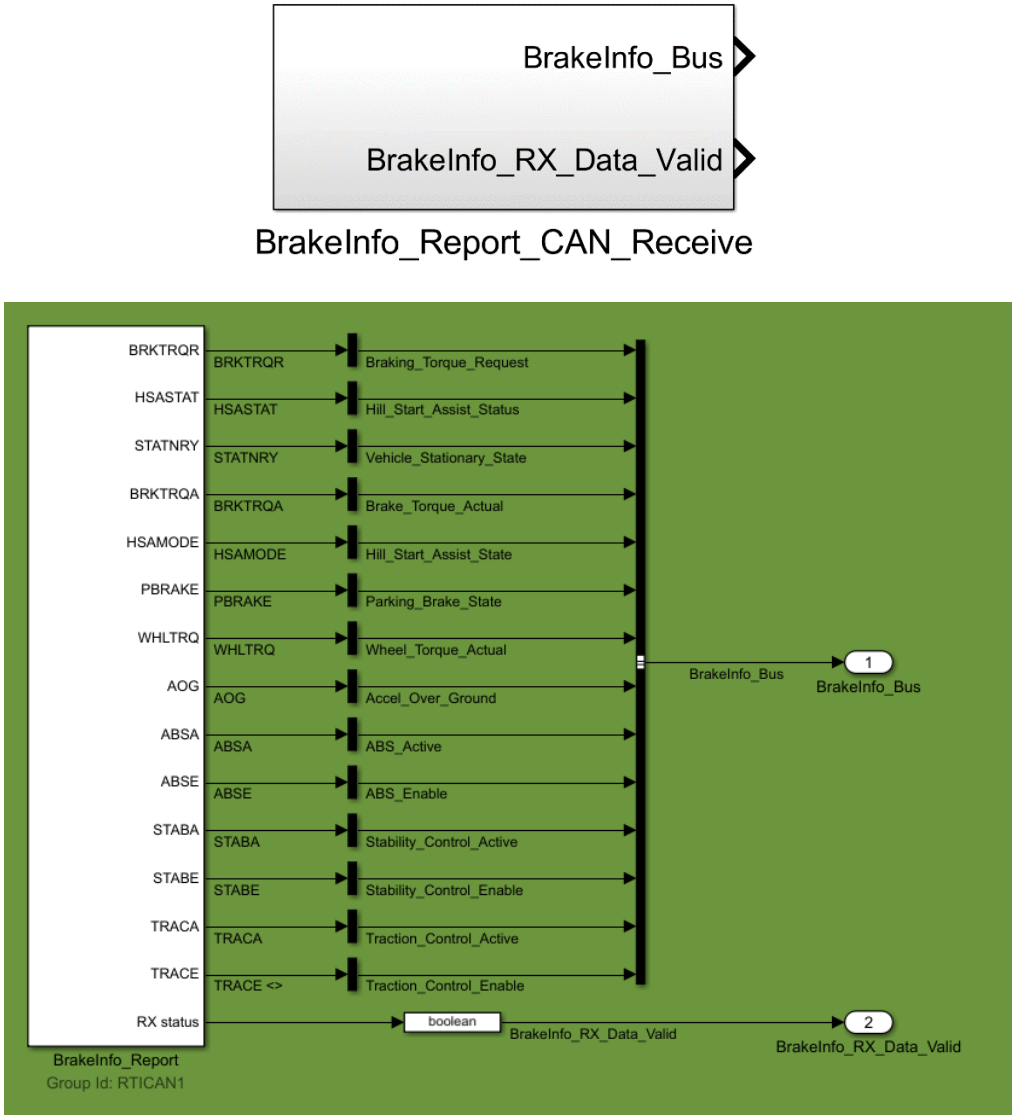


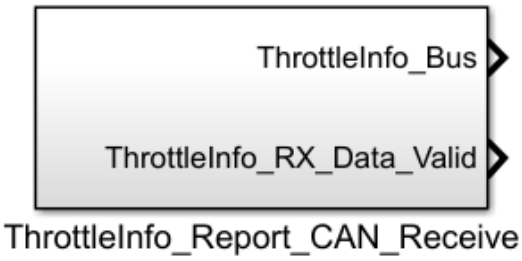
Figure 27. **BrakeInfo_Report_CAN_Receive** block.

Table 13: Port I/O of the **BrakeInfo_CAN_Receive** library block.

	Port Name	Data Type	Range	Description
Outputs	BrakeInfo_Report_Bus	bus	—	Simulink bus containing the information in the BrakeInfo_Report CAN message (ID = 0x74). See the Steering-Shifter module datasheet for details (SteeringShifterDatasheet-RevA14.pdf).
	BrakeInfo_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

4.9. ThrottleInfo Report Data

The **ThrottleInfo_Report_CAN_Receive** block is shown in Figure *28*, a*nd* its I/O is described in Table *14*. CAN message ID 0x63.



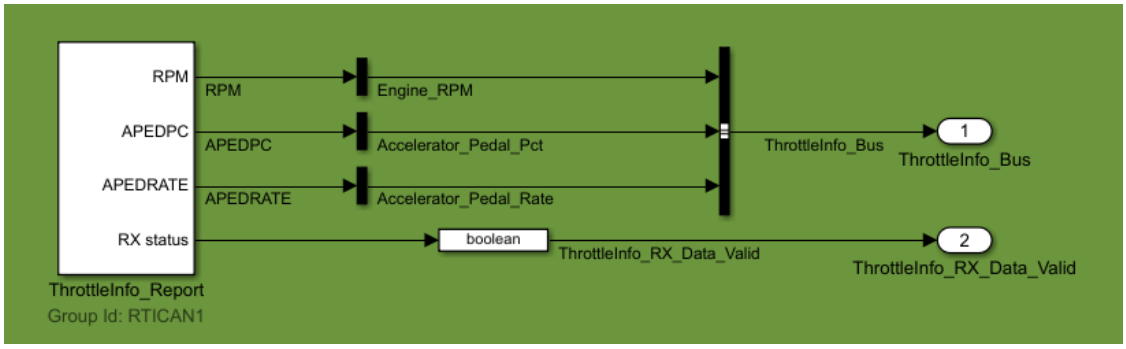


Figure 28. **ThrottleInfo_Report_CAN_Receive** block.

Table 14: Port I/O of the **ThrottleInfo_CAN_Receive** library block.

	Port Name	Data Type	Range	Description
Outputs	Engine_RPM	double	—	Engine speed (RPM).
	Accelerator_Pedal_Pct	double	0—99.9	Reported accelerator pedal position, in percent.
	Accelerator_Pedal_Rate	double	-5.12—5.08	Reported accelerator pedal rate, in %/ms.

5. ADAS Kit Info Blocks

These library blocks de-multiplex and bundle like data pertaining to the ADAS Kit itself, into the following Simulink buses:

- Firmware Version (one bus each for Brake, Throttle and Steering/Shift)
- License General Info
- MAC Address
- Build Date
- VIN (Vehicle Identification Number)
- Software Features (one bus for each feature)

They also provide the scalars:

- Platform
- Firmware Version Data Valid
- License Manager Data Valid

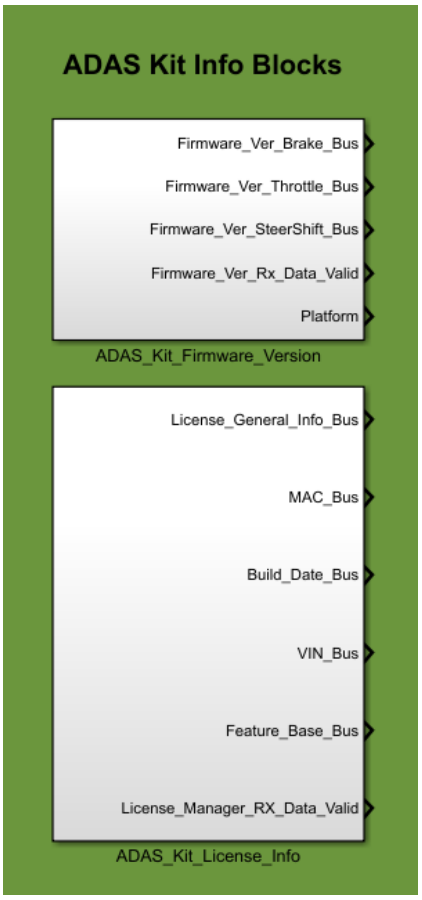


Figure 29. ADAS Kit License Info Library.

5.1. ADAS Kit Firmware Version

The **ADAS_Kit_Firmware_Version** block is shown in Figure 30, and its output is described in Table 6. CAN message ID 0x7F.

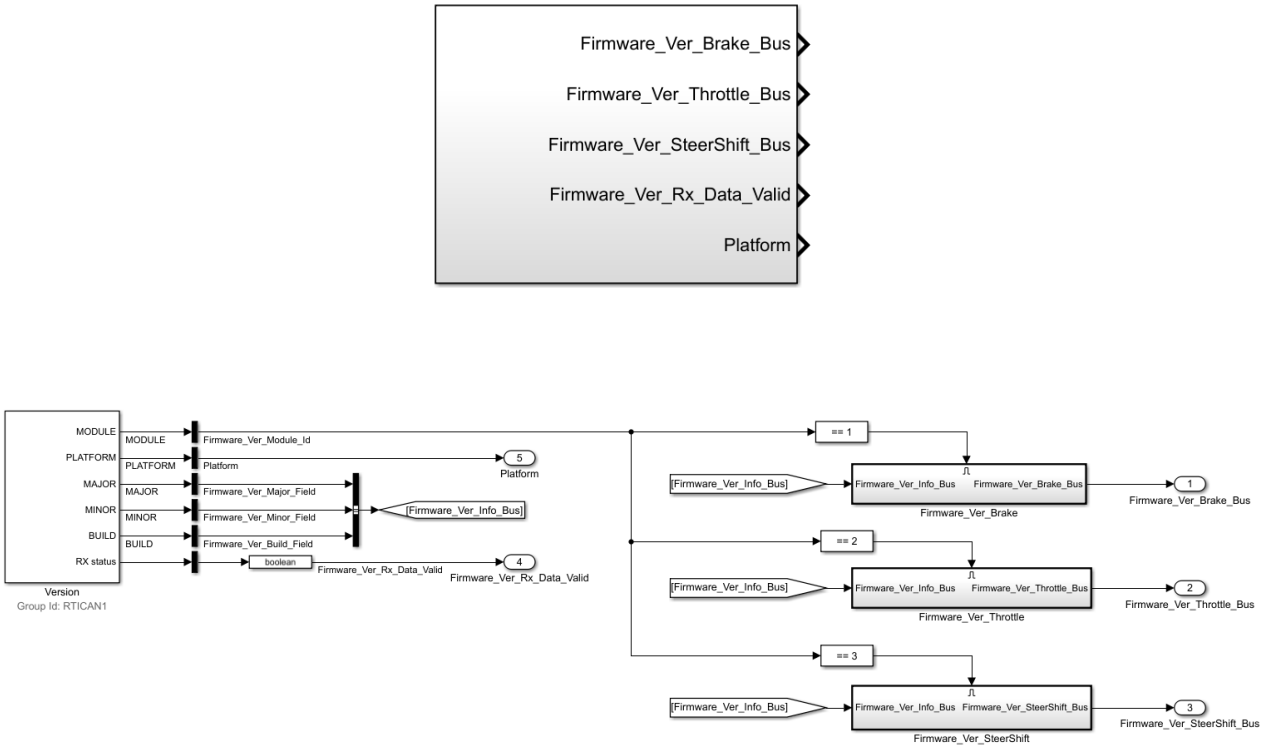


Figure 30. **ADAS_Kit_Firmware_Version** block.

Table 15. **ADAS_Kit_Firmware_Version** library block I/O.

	Port Name	Data Type	Range	Description
Outputs	Firmware_Ver_Brake_Bus	—	—	Simulink bus containing firmware version information for the brake module.
	Firmware_Ver_Throttle_Bus	—	—	Simulink bus containing firmware version information for the throttle module.
	Firmware_Ver_SteerShift_Bus	—	—	Simulink bus containing firmware version information for the steer/shift module.
	Platform	uint8	—	Vehicle platform. 0 = Ford CD4 1 – 255 = not yet assigned
	Firmware_Version_Rx_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

5.2. ADAS Kit License Info

The **ADAS_Kit_License_Info** block is shown in Figure 31, and its I/O is described in Table 16. Details of each bus are shown in Figure 32 through Figure 36. CAN message ID 0x7E.

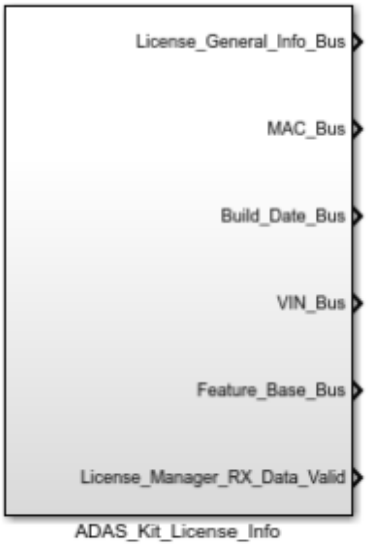


Figure 31. **ADAS_Kit_License_Info** block.

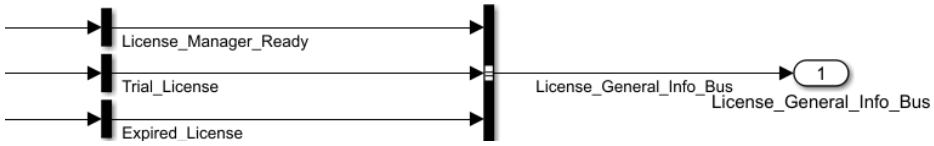


Figure 32. **License_General_Info** bus.

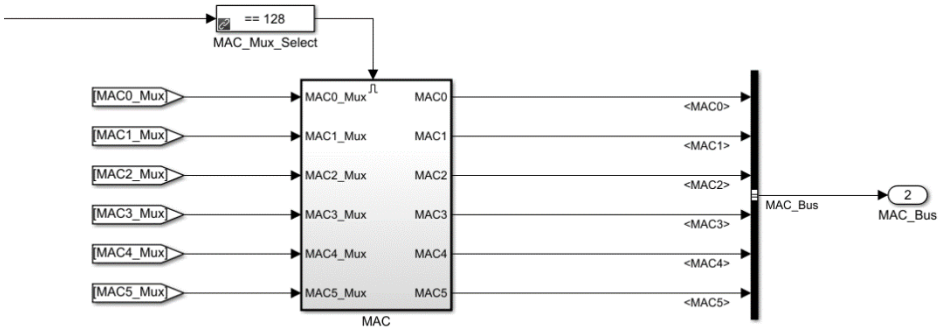


Figure 33. **MAC** bus.

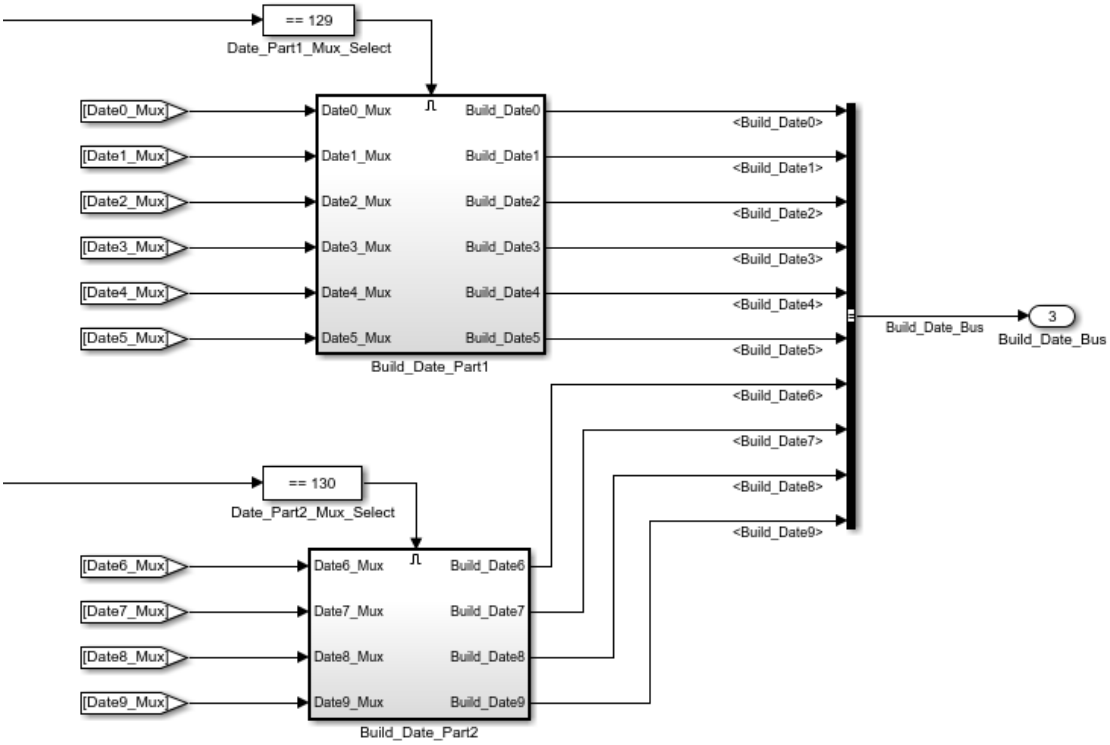


Figure 34. **Build_Date** bus.

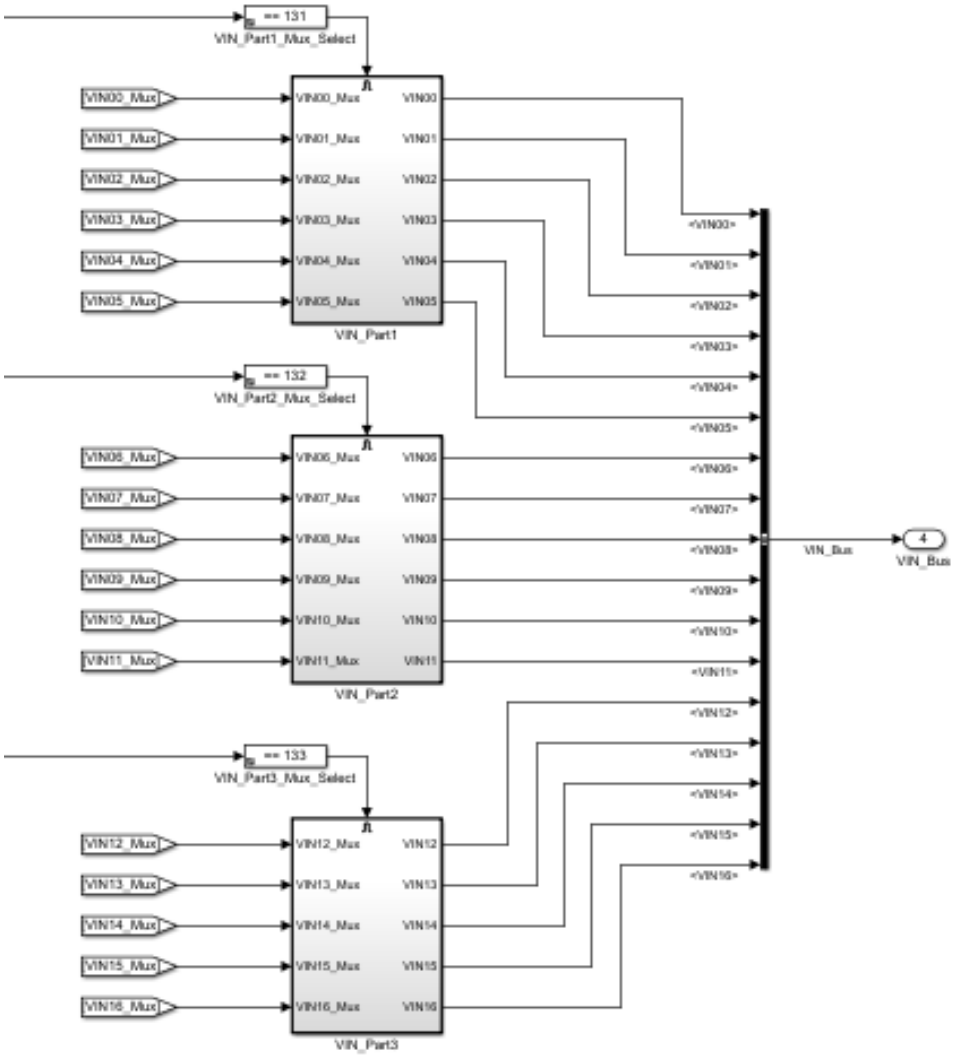


Figure 35. **VIN** bus.

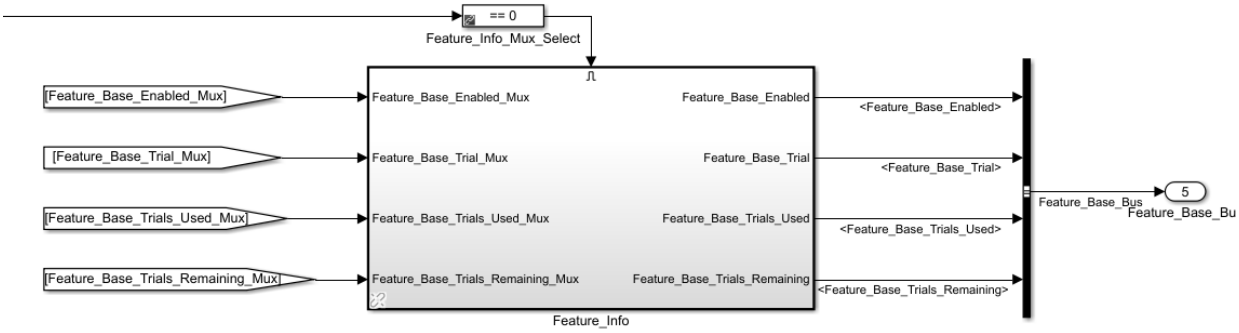


Figure 36. **Feature_Base** bus.

Table 17. Port I/O of the **ADAS_Kit_License_Info** library block.

	Port Name	Data Type	Range	Description
Outputs	License_General_Info_Bus	bus	—	Simulink bus containing bit flags indicating if the overall data for this block has yet been updated, if this is a trial license, and if the license has expired.
	MAC_Bus	bus	—	Simulink bus containing the MAC address of the Kit. MAC0 is the first (leftmost) field in the address. MAC5 is the last.
	Build_Date_Bus	bus	—	Simulink bus containing the build date of the kit's firmware. (Format: YYYY / MM / DD).
	VIN_Bus	bus	—	Simulink bus containing the VIN. VIN0 is the first (leftmost) character in the address. VIN16 is the last.
	Feature_Base_Bus	bus	—	Simulink bus providing licensing status information for the software feature called "Base". It contains bit flags reporting if this feature is enabled, and if it is a trial. For trials, it reports the number of trials used and the number remaining.
	License_Manager_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

6. Vehicle Control Blocks

These blocks can be found in the **vehicle_control_blocks.slx** library, which can be opened from the main library.

The library is shown in Figure 37. The control blocks assist the user in getting a functioning system up and running quickly.

These blocks are:

- The **DBW_System_Enable_Logic** block allows the use of steering wheel buttons to enable and disable the DBW and application level systems.
- The **Accel_Control** block provides a closed-loop acceleration control system that actuates the throttle and brake to achieve a target acceleration.

- The **Speed_And_Steering_Control** block provides a closed-loop speed controller that outputs a desired acceleration control.

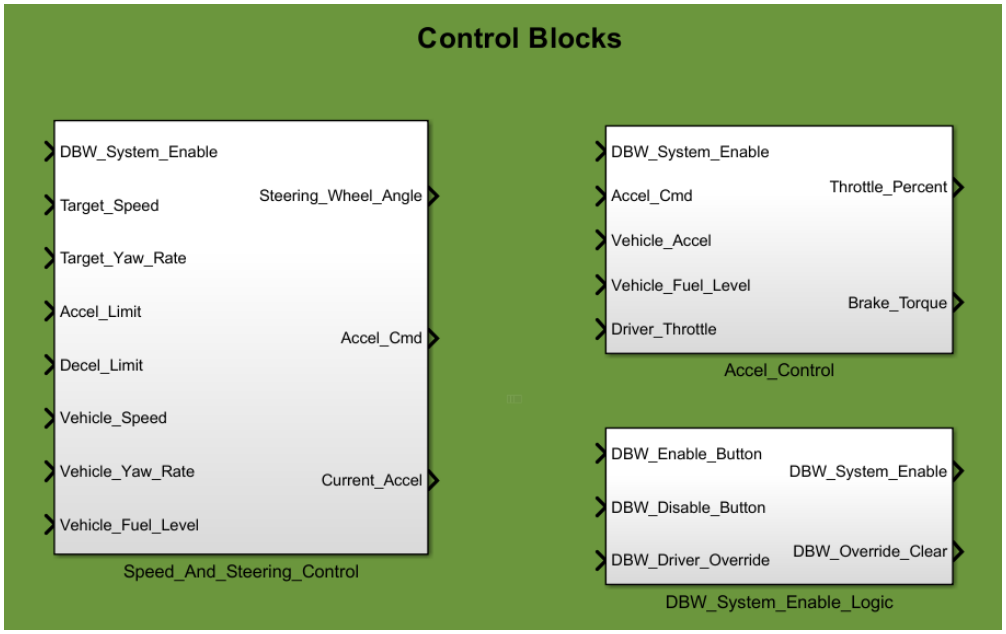


Figure 37. Individual Control Blocks library.

6.1. System Enable Logic Block

This block provides a method of using the steering wheel buttons that are available in the **Misc_Report_Bus** message (Section 4.7) to enable and disable the DBW control. This block is shown in Figure 38, and its I/O is summarized in

Table *18*.

The system enable logic block does the following:

- Waits for the **DBW_Enable_Button** input to go high, which then triggers the **DBW_Override_Clear** output high to request clearing of the driver override states of each DBW module.

- Waits until the **DBW_Driver_Override** input signal goes low, indicating that all driver override signals have been cleared. Once this happens, the **DBW_System_Enable** signal is set high to indicate that DBW control is ready.

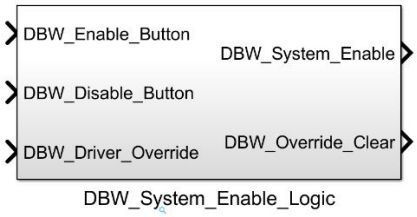
- Listens for either the **DBW_Disable_Button** input or the **DBW_Driver_Override** input to go high, indicating that the driver pressed the disable button or intervened with control of the steering wheel, pedals, or shifter. In this case, the **DBW_System_Enable** output is set low to indicate that the driver has disabled the system.

- The **DBW_Driver_Override** input should be the logical OR of all the individual override bits received from the CAN report messages from each DBW module.

- The **DBW_Override_Clear** output should be connected to all the **CLEAR** inputs on each DBW interface block to appropriately clear any driver overrides when the enable button is pressed.

- The **DBW_System_Enable** output is intended to be used within the application model as an indicator of whether DBW system control is enabled.

See the **joystick_teleop.slx** and **twist_controller.slx** demo models to see how this block can be used in an application-level system (Section 7).



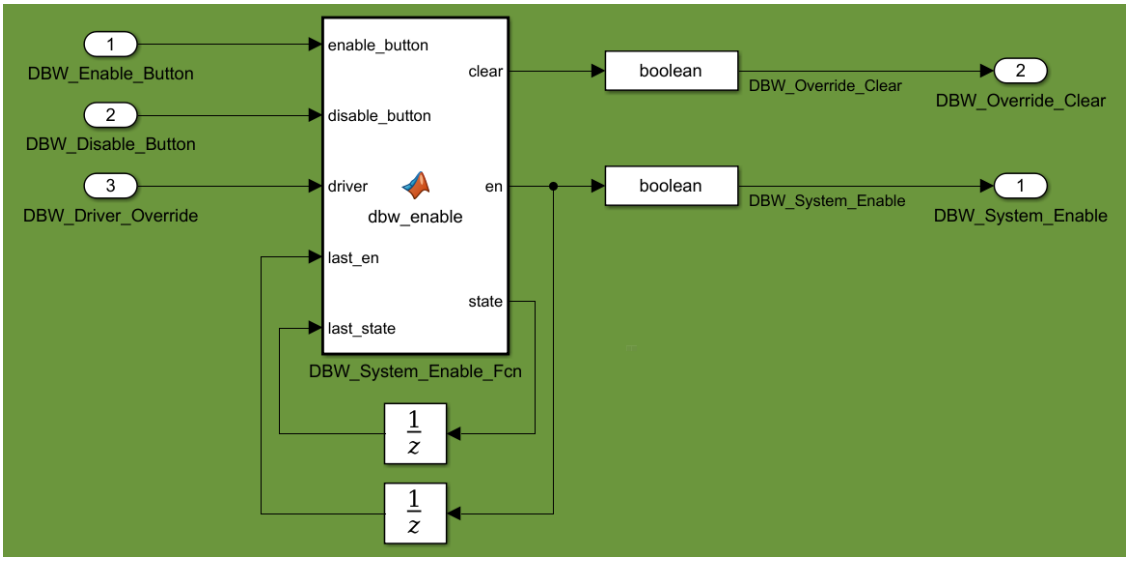


Figure 38. **DBW_System_Enable_Logic** block.

6.2. Acceleration Control

This block implements a PI controller that actuates the throttle and brake to achieve a specified longitudinal acceleration. The block is shown in Figure 39, and its I/O is summarized in Table 19. The acceleration controller has a set of configuration parameters, whose default values are specified in the **accel_controller_config_init.m** script. The script can be found in the **lib/init scripts** folder of the Dataspeed release package. This initialization script is run in the **Accel_Control** block's initialization function. It is recommended to only change the default values if absolutely necessary.

Table 18. **DBW_System_Enable_Logic** library block I/O.

	Port Name	Data Type	Range	Description
Inputs	DBW_Enable_Buttton	bool	—	Boolean signal that upon going high will clear driver overrides and enable the DBW system.
	DBW_Disable_Button	bool	—	Boolean signal that will disable the DBW system upon going high.
	DBW_Driver_Override	bool	—	Boolean status of the driver override conditions. This should be the logical OR of all override signals from each individual DBW interface.
Outputs	DBW_System_Enable	bool	—	Signal to indicate that DBW control is ready. Intended for use within the model to control program operation.
	DBW_Override_Clear	bool	—	Should be routed to each CLEAR input of the separate DBW interfaces.

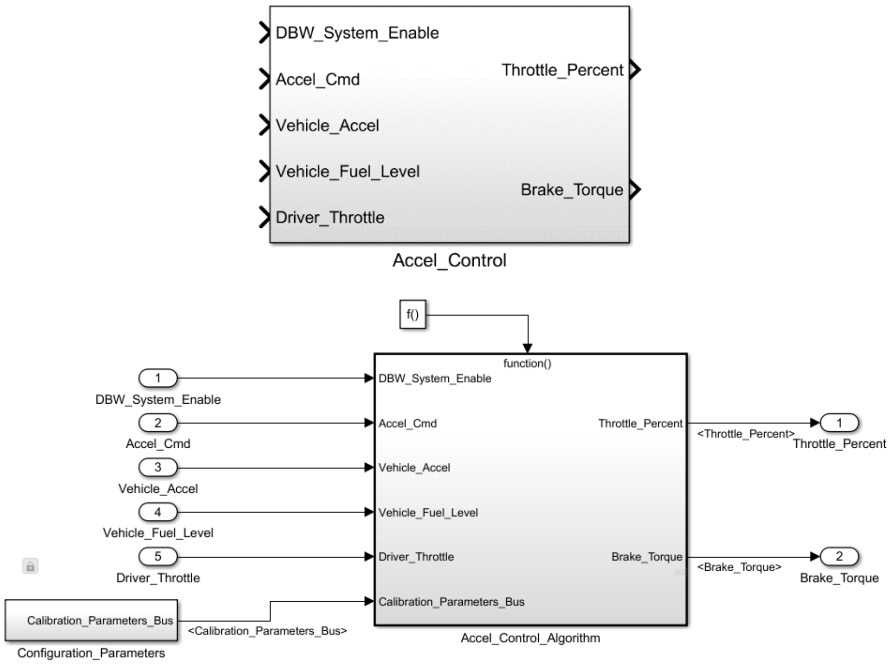


Figure 39. **Accel_Control** block.

6.3. Speed and Steering Control

This block implements a closed-loop speed controller that outputs an acceleration command to track a forward speed command. Also, it implements a kinematic steering controller to track a specified yaw rate command. The block is shown in Figure 40, and its I/O is summarized in Table 20. The speed and steering controllers have a set of configuration parameters, whose default values are specified in the **speed_controller_config_init.m** script. The script can be found in the **lib/init scripts** folder of the Dataspeed release package. This initialization script is run in

the **Speed_And_Steering_Control** block's initialization function. It is recommended to only change the default values if absolutely necessary.

Table 19: **Accel_Control** library block I/O.

	Port Name	Data Type	Range	Description
Inputs	DBW_System_Enable	bool	—	Boolean signal indicating if DBW system is enabled. Expected to come from the **DBW_System_Enable_Logic** block (Section 6.1).
	Accel_Cmd	double	-9.8 — 9.8	Desired longitudinal acceleration in m/s^2.
	Vehicle_Accel	double	Application Specific	Measurement of the current longitudinal acceleration in m/s^2.
	Vehicle_Fuel_Level	double	0 — 100	Fuel level percentage. Expected to come from the **Fuel_Level_Receive** block (Section 4.4).
	Driver_Throttle	double	0.15 — 0.8	Current throttle position applied by driver. This is used to detect when the driver takes over the throttle. When the driver takes over, the controller integrator is reset and brake output is disabled until the driver releases the throttle, at which point control resumes automatically. Expected to come from the **Throttle_Pedal_Physical** signal on the **Throttle_Report_Bus** (Section 3.1).
Outputs	Throttle_Percent_Cmd	double	0 — 1	Relative throttle output, where 0 is no throttle, and 1 is full throttle. Remember to scale this value into the valid output range 0.15 – 0.8. An embedded MATLAB code block to do this can be found in the **twist_controller.slx** example model (Section 7). Look for the **Throttle_Percent_To_Command** embedded MATLAB function in the **Vehicle_Interface** block.
	Brake_Torque_Cmd	double	0 — 3412	Brake torque output in *N-m*. A lookup-table to map brake torque to brake pedal command is implemented in the **twist_controller.slx** example model (Section 7). (Look for the **Brake_Torque_To_Command** embedded MATLAB function in the **Vehicle_Interface** block.)

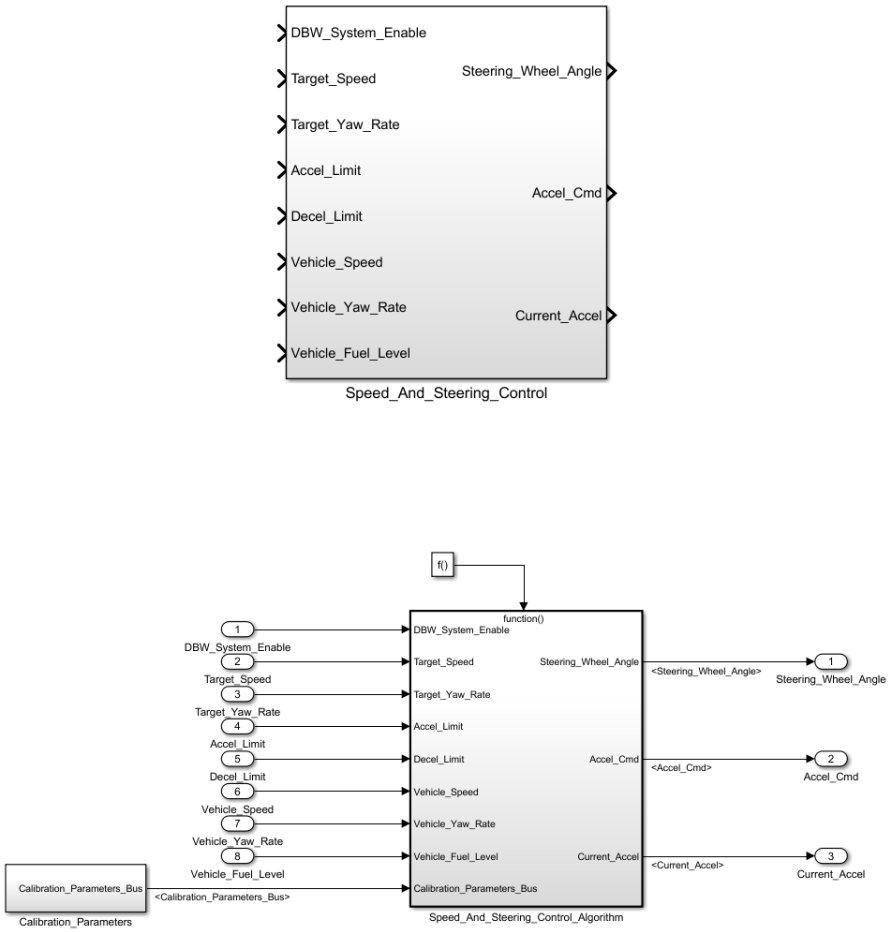


Figure 40. **Speed_And_Steering_Control** block.

Table 20. Port I/O of the **Speed_And_Steering_Control** library block.

	Port Name	Data Type	Range	Description
Inputs	DBW_System_Enable	bool	—	Boolean signal indicating if DBW system is enabled. Expected to come from the **DBW_System_Enable_Logic** block (Section 6.1).
	Target_Speed	double	0 — 50	Set point speed for the controller to track in *m/s*.
	Target_Yaw_Rate	double	-2.0 — 2.0	Set point yaw rate in *rad/s*.
	Accel_Limit	double	0 — 9.8	External acceleration limit in m/s^2. If set to zero, then the default value **ACCEL_MAX** from the initialization script is used.
	Decel_Limit	double	0 — 9.8	External deceleration limit in m/s^2. If set to zero, then the default value **DECEL_MAX** from the initialization script is used.
	Vehicle_Speed	double	Application Specific	Measurement of the current vehicle speed in *m/s*.
	Vehicle_Yaw_Rate	double	Application Specific	Measurement of the current vehicle yaw rate in *rad/s*.
	Vehicle_Fuel_Level	double	0 — 100	Fuel level in *percent*. Expected to come from the **Fuel_Level_CAN_Receive** block (Section 4.4).

7. Demo Model

- **Joystick Teleop** – Uses the signals from a USB video game joystick to control the steering, brakes, throttle, shifter and turn signals.