



# **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.

Author and Point of Contact:  
Micho Radovnikov

([mradovnikovich@dataspeedinc.com](mailto:mradovnikovich@dataspeedinc.com))

---

## Contents

<b>1. Introduction .....</b>	<b>4</b>
<b>2. Software Installation .....</b>	<b>6</b>
<b>3. Drive-by-Wire Interface Blocks .....</b>	<b>6</b>
<b>3.1. Throttle .....</b>	<b>7</b>
<b>3.2. Brake .....</b>	<b>10</b>
<b>3.3. Steering .....</b>	<b>13</b>
<b>3.4. Shifter .....</b>	<b>16</b>
<b>3.5. Turn Signal .....</b>	<b>18</b>
<b>3.6. Complete Interface .....</b>	<b>19</b>
<b>4. Vehicle Data Blocks .....</b>	<b>20</b>
<b>4.1. Wheel Speed .....</b>	<b>21</b>
<b>4.2. Gyro .....</b>	<b>22</b>
<b>4.3. Acceleration .....</b>	<b>23</b>
<b>4.4. Fuel Level .....</b>	<b>23</b>
<b>4.5. Tire Pressure .....</b>	<b>24</b>
<b>4.6. GPS .....</b>	<b>25</b>
<b>4.7. Miscellaneous Data .....</b>	<b>26</b>
<b>4.8. BrakeInfo Report Data .....</b>	<b>27</b>
<b>5. ADAS Kit Info Blocks .....</b>	<b>29</b>
<b>5.1. ADAS Kit Firmware Version .....</b>	<b>29</b>
<b>5.2. ADAS Kit License Info .....</b>	<b>30</b>
<b>6. Control Blocks .....</b>	<b>34</b>
<b>6.1. System Enable Logic Block .....</b>	<b>35</b>
<b>6.2. Acceleration Control .....</b>	<b>36</b>
<b>6.3. Speed and Steering Control .....</b>	<b>37</b>

---

<b>7. Demo Models .....</b>	<b>40</b>
-----------------------------	-----------

---

## 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-RevA09.pdf](#), [SteeringShifterDatasheet-RevA12.pdf](#)).

The main Simulink library **dataspeed\_drive\_by\_wire.slx** is shown in Figure 1. This library has the following components:

- **CAN Configuration** – A pre-configured block that properly sets up the dSPACE RTICAN interface to send messages to 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 is necessary for the correct path to the .dbc file to be set. The .dbc file can be found in the **lib** folder in the Dataspeed DBW release package.
- **Complete Interface Block** – This block contains all DBW subsystems and provides the required inputs and outputs to interface to the subsystems.
- **Links to Individual Subsystems** – These blocks simply open other Simulink libraries that contain individual blocks for each DBW subsystem, and each individual vehicle data subsystem.
- **Control Blocks** – These blocks provide basic control functionality.
- **Demo Models** – Models that demonstrate the usage of the DBW interface blocks, and provide baselines from which to expand.

Double-clicking the square boxes next to each 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 won't affect the original demo model.



Figure 1. Main Simulink library.

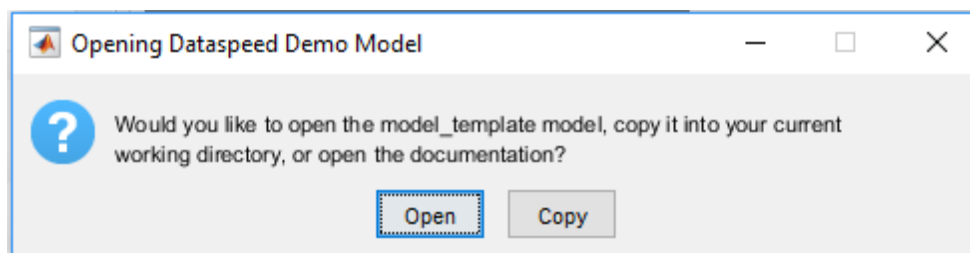


Figure 2. Demo Models dialog box.

---

## 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, simply hit "Enter" on your keyboard.
5. The installation script will add all of the 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 **drive\_by\_wire\_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. Throttle

The **Throttle Interface** block is shown in Figure 4 and 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 (ID = 0x62), and receive the throttle report CAN message (ID = 0x63).

The **Throttle\_CAN\_Transmit** and **Throttle\_CAN\_Receive** subsystem blocks are shown in Figure 6 and Figure 7, respectively.

The transmit block does the following:

- Passes the pedal command into the **PCMD** field of the throttle command CAN message.
- Passes the enable signal to the **EN** bit of the CAN message.
- Passes the clear signal to the **CLEAR** bit of the CAN message.
- Passes the ignore signal to the **IGNORE** bit of the CAN message.

The receive block does the following:

- Parses the throttle report message and combines the received data into a Simulink bus.
- The **Throttle\_RX\_Data\_Valid** flag is used to indicate if the report message is being received.

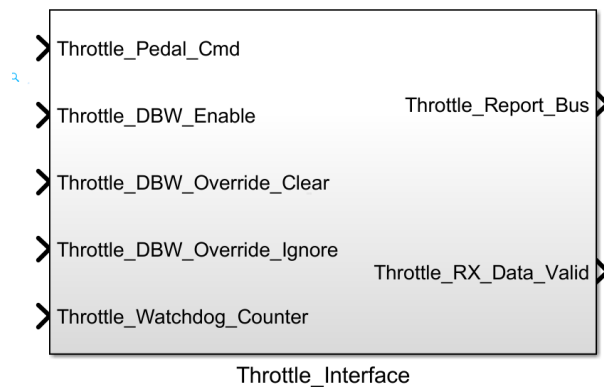


Figure 4. **Throttle\_Interface** block.

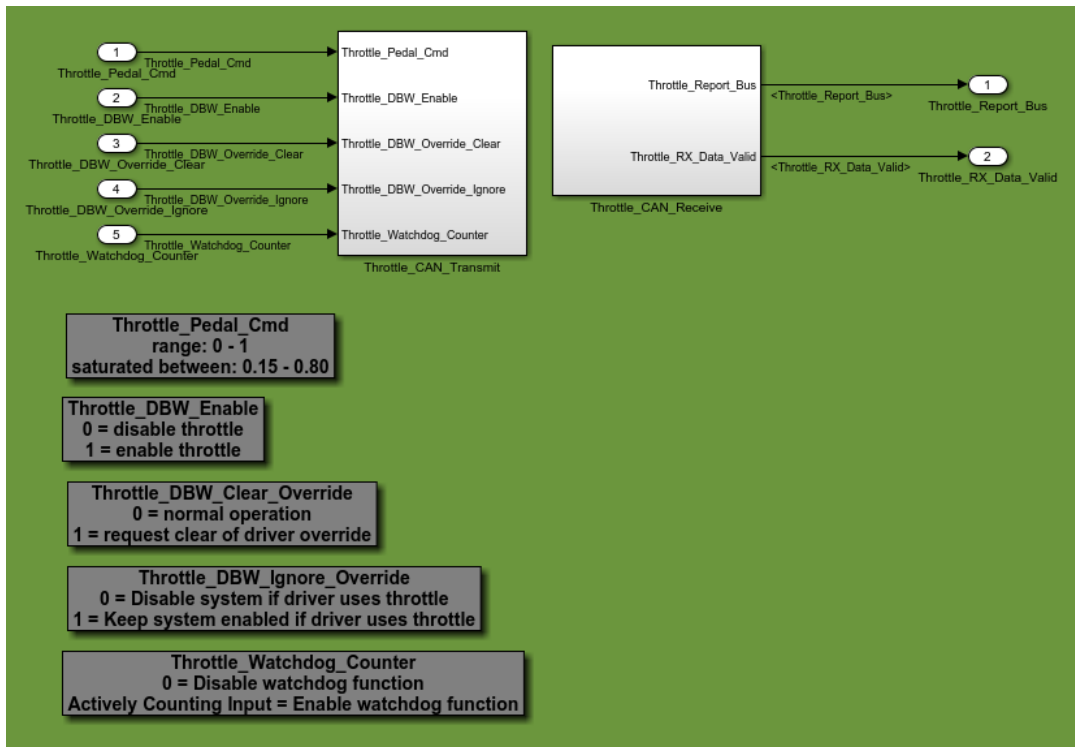


Figure 5. **Throttle\_Interface** block.

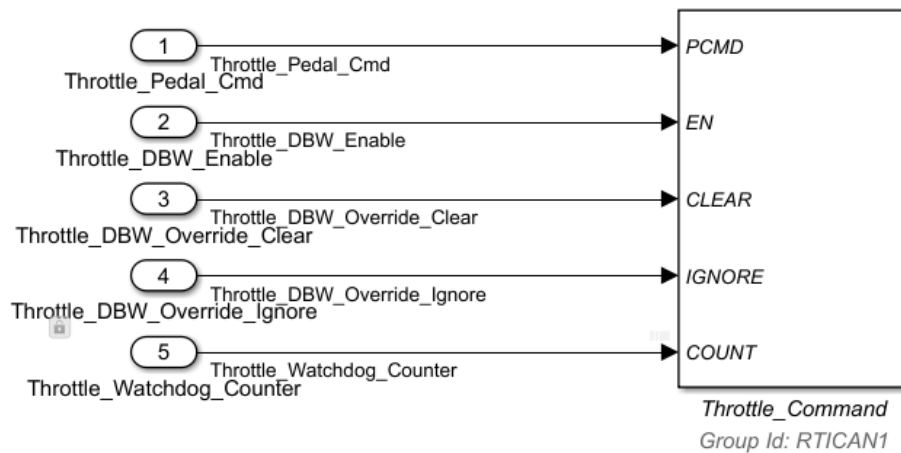


Figure 6. **Throttle\_CAN\_Transmit** block.



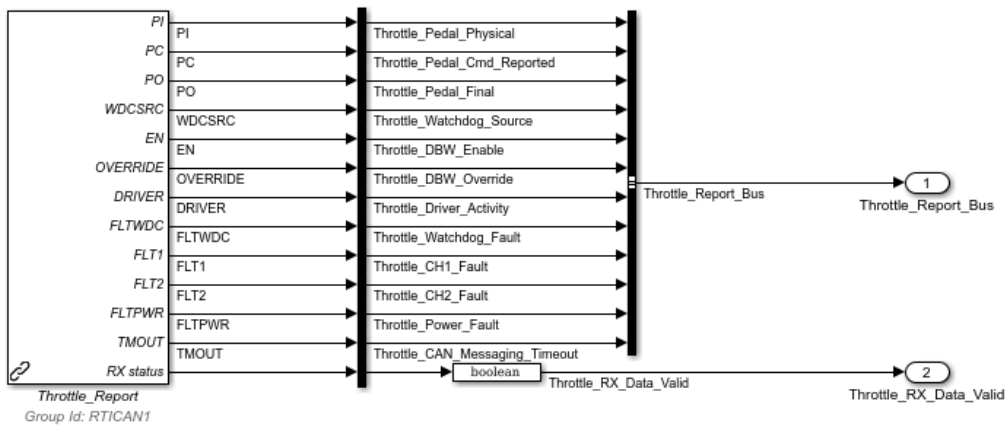


Figure 7. **Throttle\_CAN\_Receive** block.

Table 1. Port I/O of the Throttle Interface Library block.

	Port Name	Data Type	Range	Description
<b>Inputs</b>	Throttle_Pedal_Cmd	double	0.15 — 0.8	PWM duty cycle for the throttle pedal. Values outside the range will be saturated.
	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 ( <a href="#">ThrottleBrakeDatasheet-RevA09.pdf</a> ) for details
<b>Outputs</b>	Throttle_Report_Bus	bus	—	Simulink bus containing the data in the throttle report CAN message. See datasheet ( <a href="#">ThrottleBrakeDatasheet-RevA09.pdf</a> ) for details.
	Throttle_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

---

## 3.2. 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 (ID = 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.

The transmit block does the following:

- Passes the pedal command into the **PCMD** field of the CAN message.
- Passes the enable signal into the **EN** bit of the CAN message.
- Passes the clear signal to the **CLEAR** bit of the CAN message.
- If **BOO\_Command** is high, it overrides the pedal position logic and sets the **BCMD** bit to high in the CAN message.
- If **BOO\_Thres** is less than the minimum pedal position command of 0.15, then the **BCMD** bit is set to high when the pedal command becomes higher than the default value of 0.2.
- If **BOO\_Thres** is higher than 0.15, the **BCMD** bit is set to high when the pedal command becomes higher than the specified threshold.

The receive block does the following:

- Parses the brake report message and combines the received data into a Simulink bus.
- The **Brake\_RX\_Data\_Valid** flag is used to indicate if the report message is being received.

Table 2. Port I/O of the Brake Interface Library block.

	Port Name	Data Type	Range	Description
<b>Inputs</b>	Brake_Pedal_Cmd	double	0.15 — 0.5	PWM duty cycle for the throttle pedal. Values outside the range will be saturated.
	Brake_BOO_Cmd	bool	—	Flag to directly control the BOO signal. TRUE: force BOO high, FALSE: allow automatic BOO control based on pedal position.
	Brake_BOO_Thres	bool	0 — 0.5	Used to override the default BOO pedal position threshold of 0.2. ≤ 0.15: use default threshold of 0.2. > 0.15: replace threshold.
	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_Watchdog_Counter	uint8	0 — 255	Optional watchdog counter. 0: disables feature. See datasheet ( <a href="#">ThrottleBrakeDatasheet-RevA09.pdf</a> ) for details
<b>Outputs</b>	Brake_Report_Bus	bus	—	Simulink bus containing the data in the brake report CAN message. See datasheet ( <a href="#">ThrottleBrakeDatasheet-RevA09.pdf</a> ) for details.
	Brake_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.



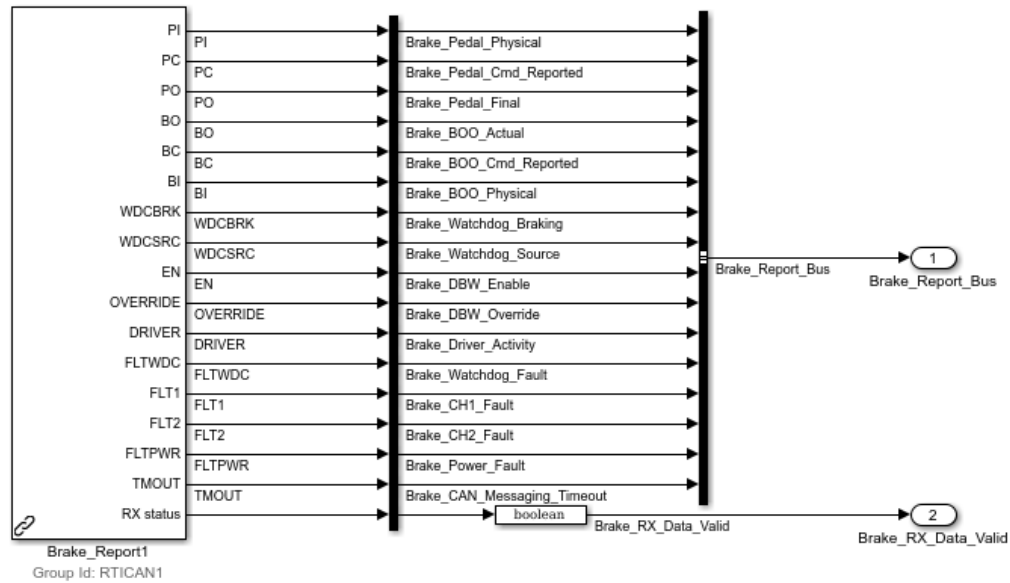


Figure 10. **Brake\_CAN\_Receive** block.

### 3.3. 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 (ID = 0x64), and receive the steering report CAN message (ID = 0x65).

The **Steering\_CAN\_Transmit** and **Steering\_CAN\_Receive** subsystem blocks are shown in Figure 12 and Figure 13, respectively.

The transmit block does the following:

- Passes the steering command into the **SCMD** field of the steering command CAN message.
- Passes the enable signal into **EN** bit of the CAN message.
- Passes the clear signal to the **CLEAR** bit of the CAN message.
- Passes the steering velocity command into the **SVEL** field of the CAN message.

The receive block does the following:

- Parses the steering report message and combines the received data into a Simulink bus.
- The **Steering\_RX\_Data\_Valid** flag is used to indicate if the report message is being received.

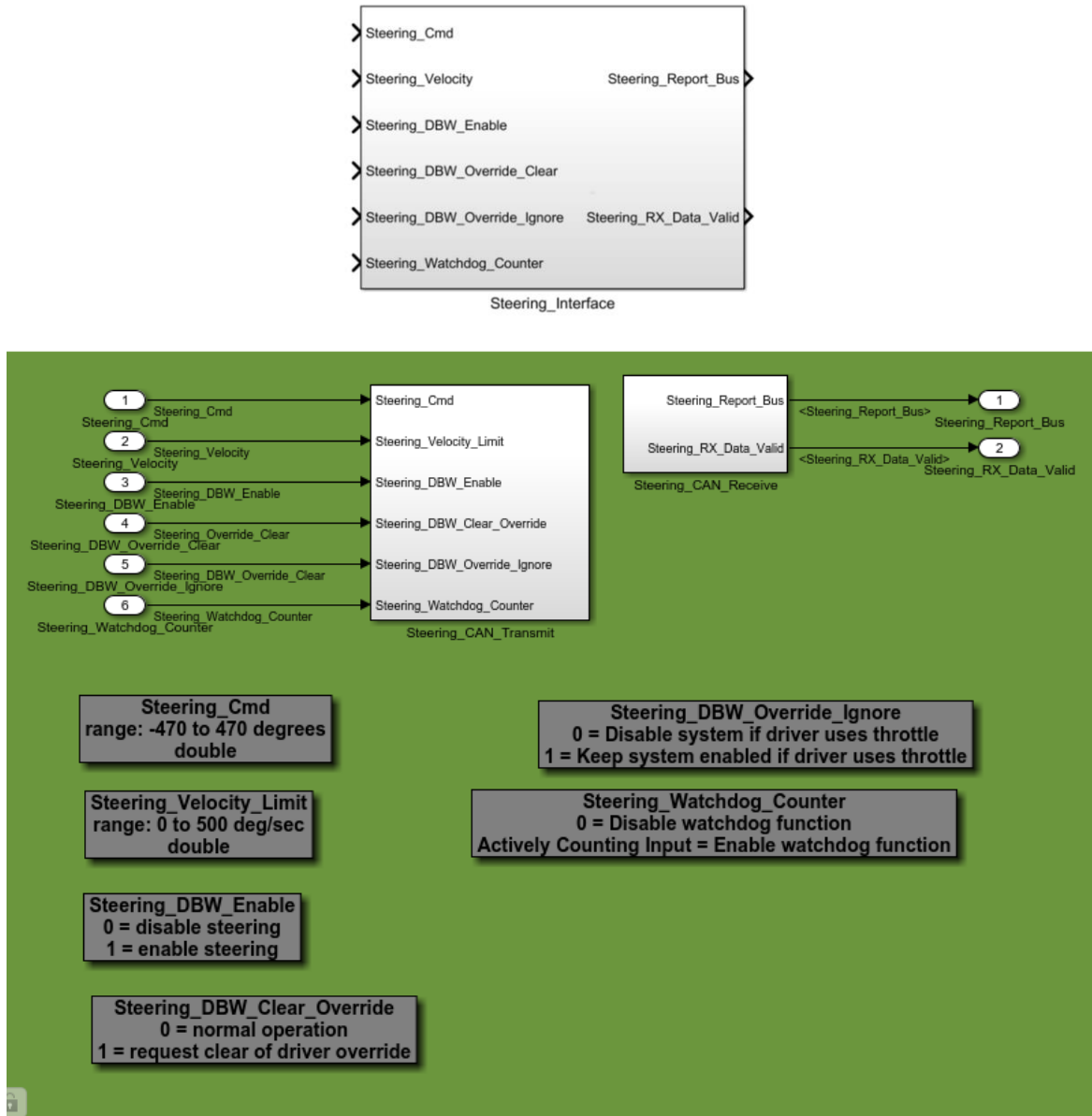


Figure 11. **Steering\_Interface** block.

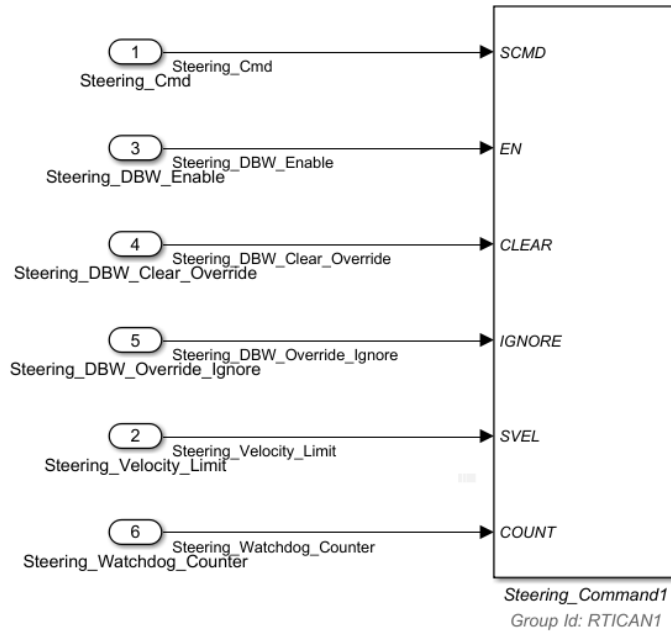


Figure 12. **Steering\_CAN\_Transmit** block.

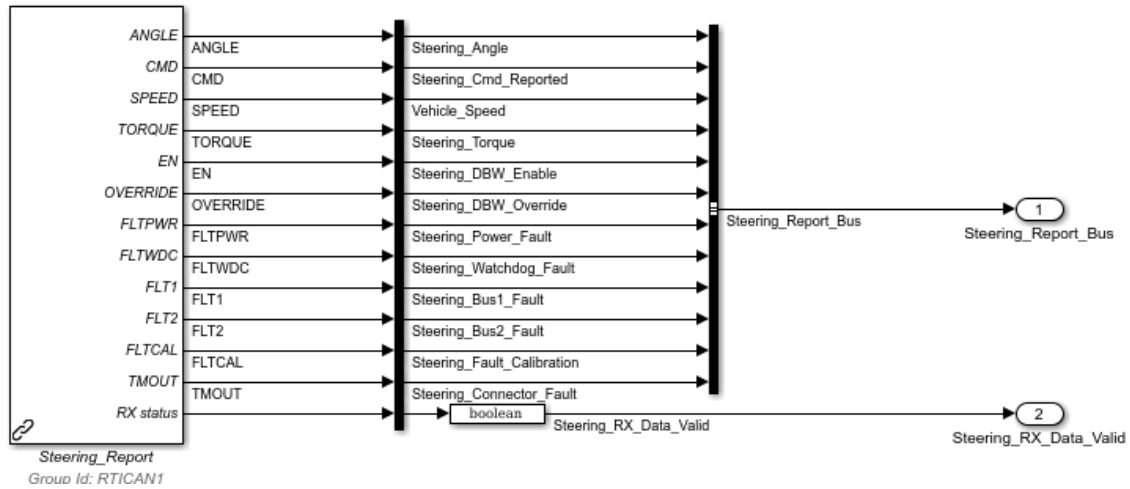


Figure 13. **Steering\_CAN\_Receive** block.

Table 3. Port I/O of the **Steering\_Interface** library block.

	Port Name	Data Type	Range	Description
<b>Inputs</b>	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 <i>deg/sec</i> to use while moving steering wheel to desired angle. Setting to zero will default to maximum of 500 <i>deg/sec</i> .
	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 ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) for details.
<b>Outputs</b>	Steering_Report_Bus	bus	—	Simulink bus containing the data in the steering report CAN message. See datasheet ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) for details.
	Steering_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

### 3.4. 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 (ID = 0x66), and receive the gear report CAN message (ID = 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** field 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.

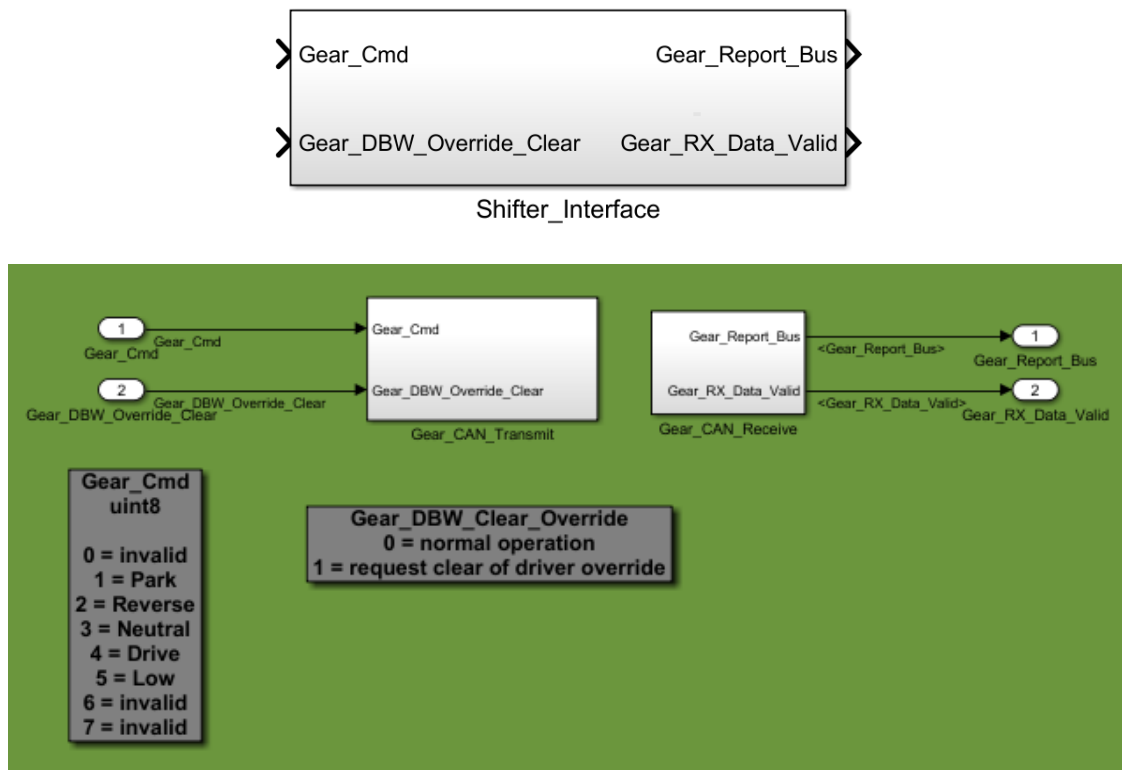


Figure 14. **Shifter\_Interface** block.

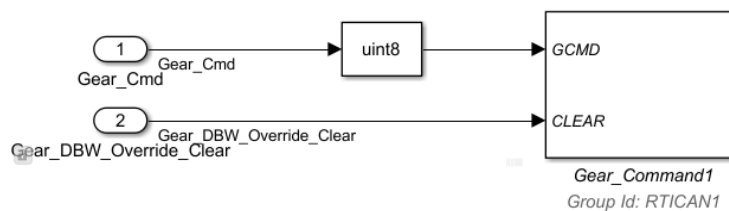


Figure 15. **Gear\_CAN\_Transmit** block.

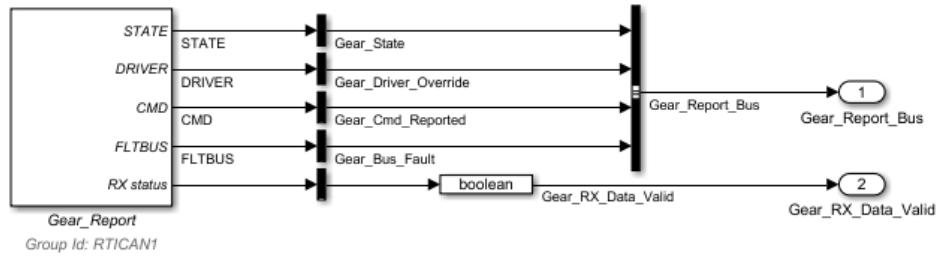


Figure 16. **Gear\_CAN\_Receive** block.

Table 4. Port I/O of the **Shifter\_Interface** library block.

	Port Name	Data Type	Range	Description
<b>Inputs</b>	Gear_Cmd	uint8	0 — 5	Enumeration of desired shifter position. 0: None 1 — 5: P, R, N, D, L
	Clear	bool	—	Clear driver override. TRUE: request clear of driver override. FALSE: normal operation.
<b>Outputs</b>	Report	bus	—	Simulink bus containing the data in the gear report CAN message. See datasheet ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) for details.
	Gear_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

### 3.5. 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 (ID = 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.

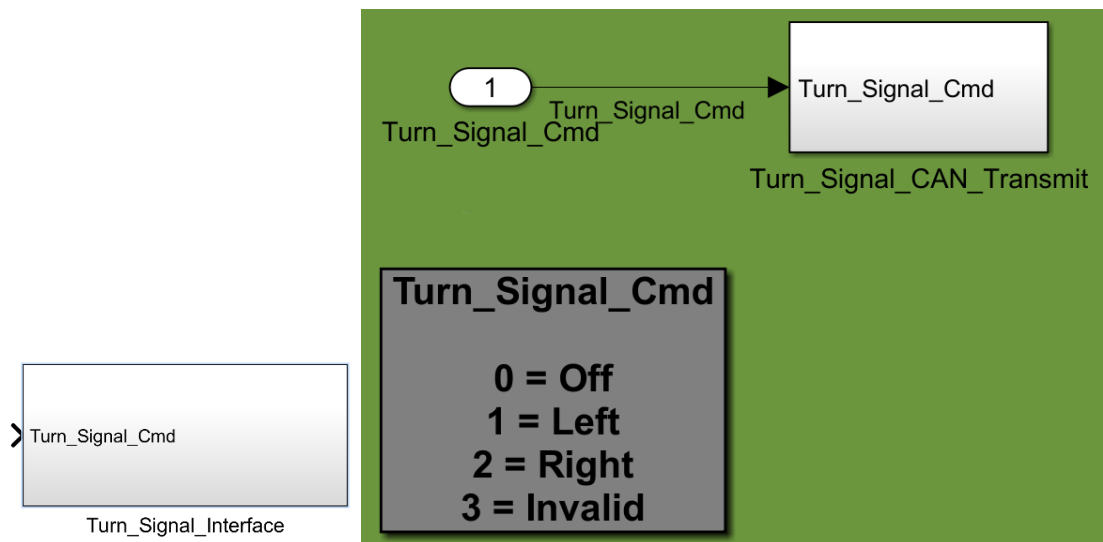


Figure 17. **Turn\_Signal\_Interface** block

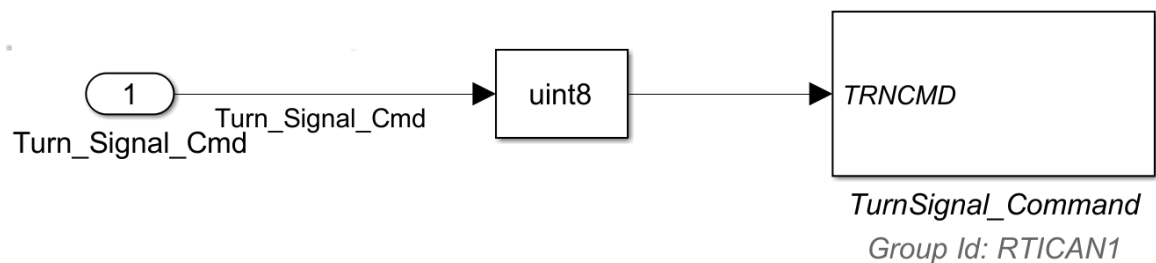


Figure 18. **Signal\_CAN\_Transmit** block

Table 5. Port I/O of the **Turn\_Signal\_Interface** library block.

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

### 3.6. Complete Interface

In the main library, a block is provided that contains all of the individual subsystem interfaces, as shown in Figure 19 and Figure 20.

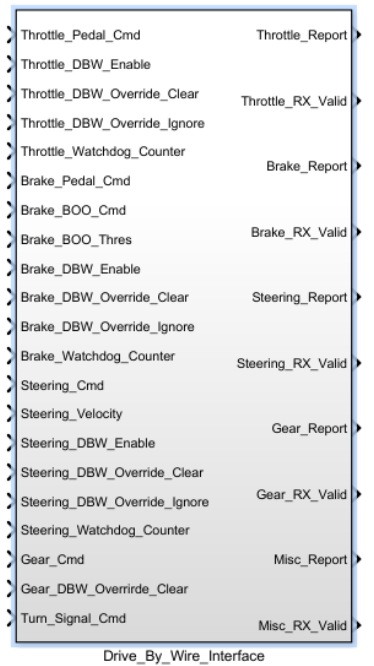


Figure 19. Complete **Drive\_By\_Wire\_Interface** block.



Figure 20. Complete **Drive\_By\_Wire\_Interface** block (internals).

## 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 21. This library can also be opened from the main **dataspeed\_drive\_by\_wire.slx** library.

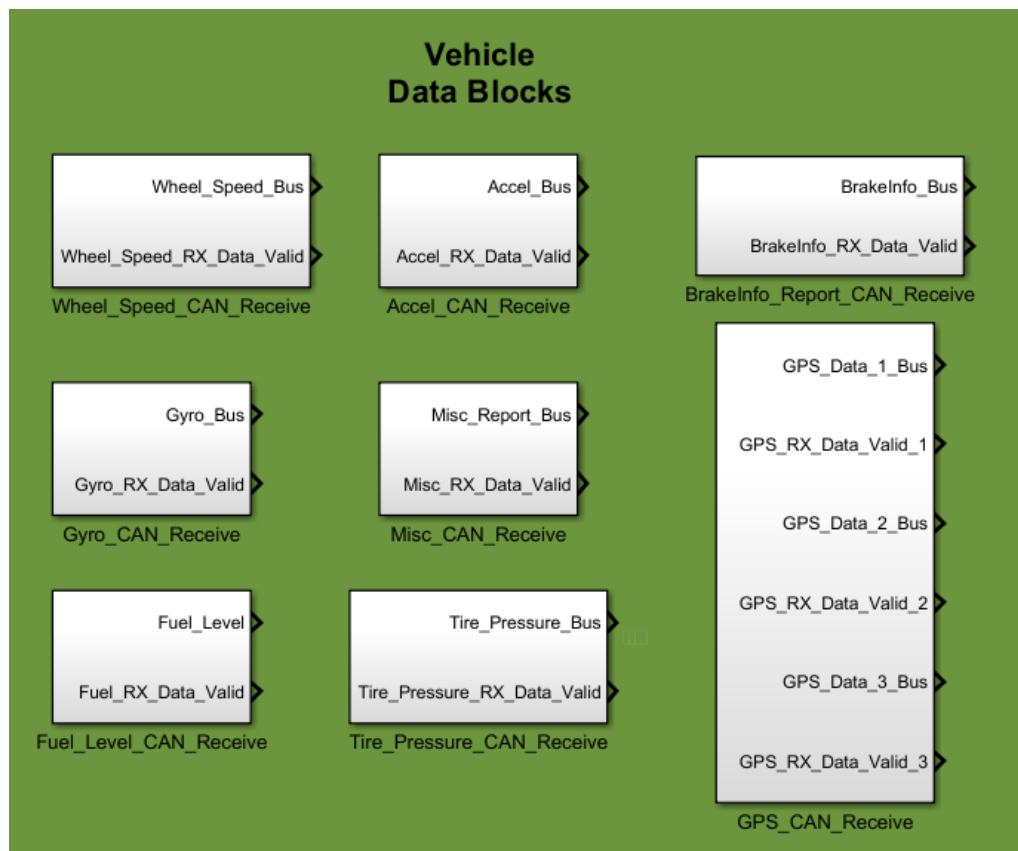


Figure 21. Vehicle Data Library.

## 4.1. Wheel Speed

The **Wheel\_Speed\_CAN\_Receive** block is shown in Figure 22, and its I/O is described in Table 6.

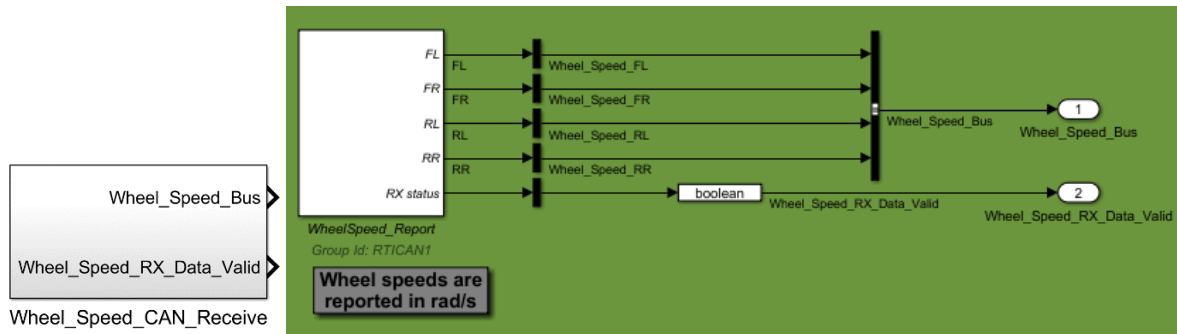


Figure 22. **Wheel\_Speed\_CAN\_Receive** block.

Table 6. Port I/O of the **Wheel\_Speed\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
Outputs	Wheel_Speed_Bus	bus	—	Simulink bus containing the four individual wheel speeds in <i>rad/s</i> .
	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 23, and its I/O is described in Table 7.

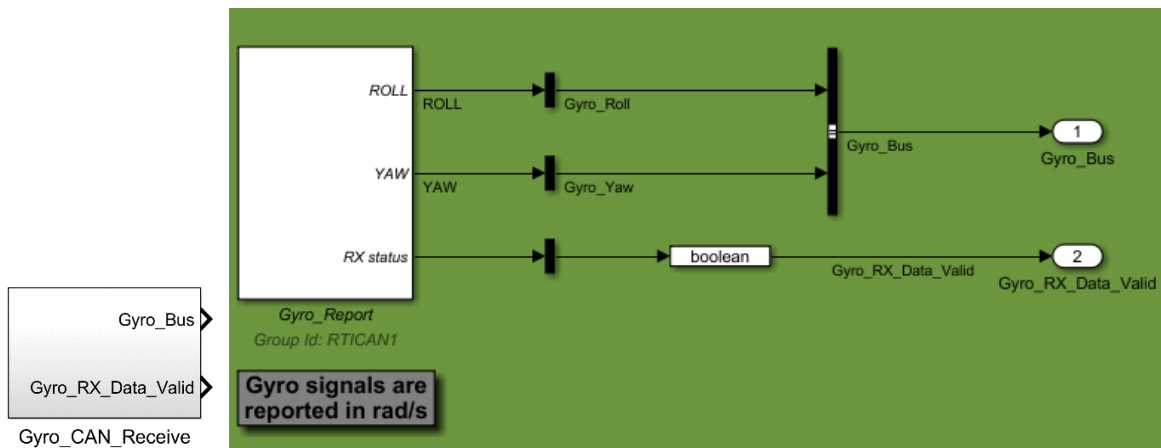


Figure 23. **Gyro\_CAN\_Receive** block.

Table 7. Port I/O of the **Gyro\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	Gyro_Bus	bus	—	Simulink bus containing the vehicle roll and yaw rates in <i>rad/s</i> .
	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 24, and its I/O is described in Table 8.

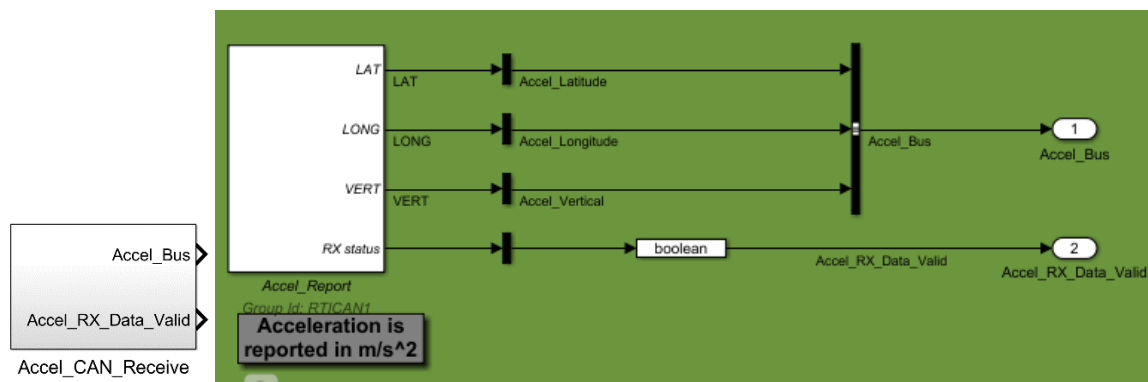


Figure 24. **Accel\_CAN\_Receive** block.

Table 8: Port I/O of the **Accel\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	Accel_Bus	bus	—	Simulink bus containing the longitudinal, lateral and vertical acceleration in <i>m/s<sup>2</sup></i> .
	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 25, and its I/O is described in Table 9.

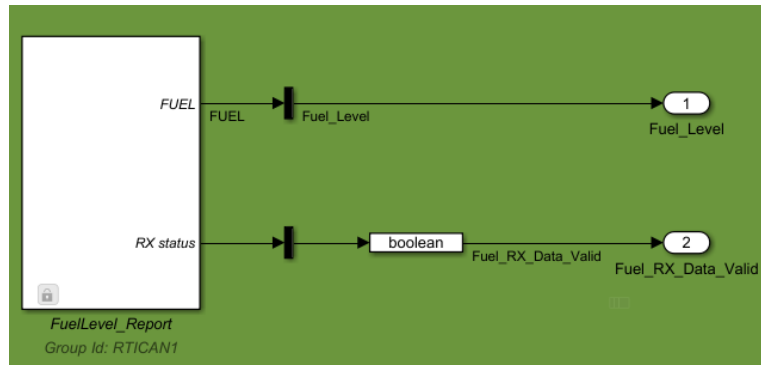
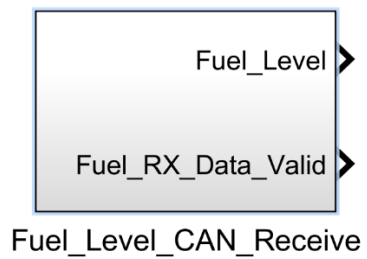


Figure 25. **Fuel\_Level\_CAN\_Receive** block.

Table 9. Port I/O of the fuel level data library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	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 26, and its I/O is described in Table 10.

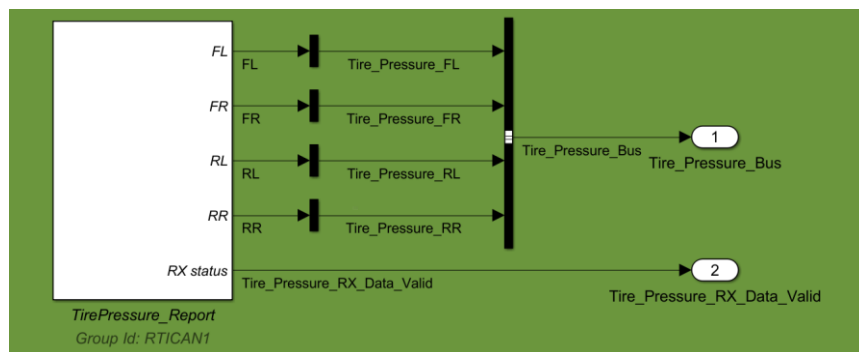
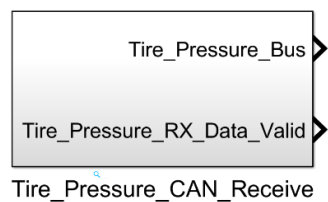


Figure 26. **Tire\_Pressure\_CAN\_Receive** block.



Table 10. Port I/O of the **Tire\_Pressure\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	Tire_Pressure_Bus	bus	0 — 65535	Simulink bus containing tire pressure for each tire in <i>kPa</i> .
	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 27, and its I/O is described in Table 11

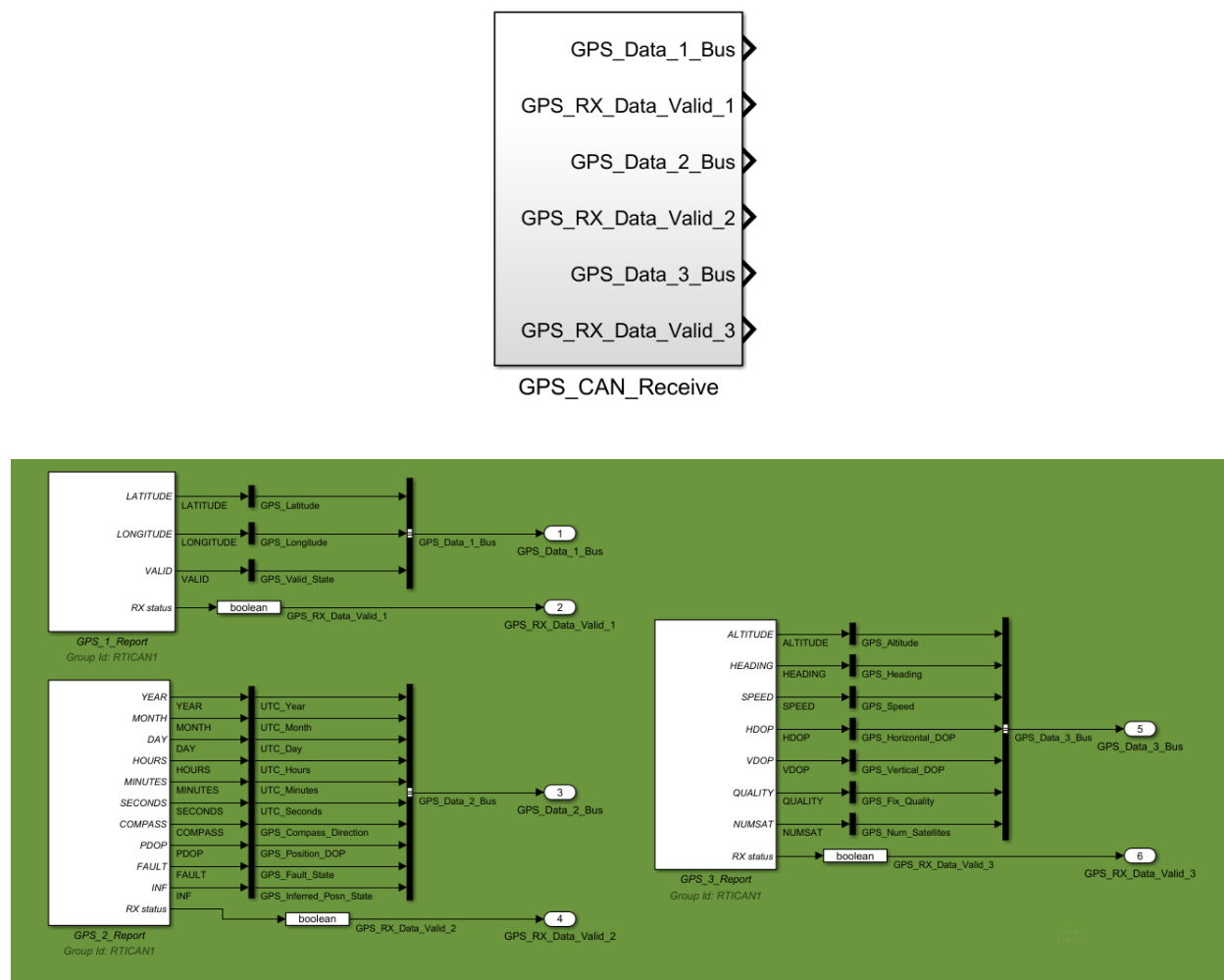


Figure 27. **GPS\_CAN\_Receive** block.

Table 11. Port I/O of the **GPS\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	GPS_Data_1	bus	—	Simulink bus containing latitude and longitude data. See datasheet ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) 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 ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) 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 ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ) 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 28, and its I/O is described in Table 12.

Table 12: Port I/O of the **Misc\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	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 ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ).
	Misc_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

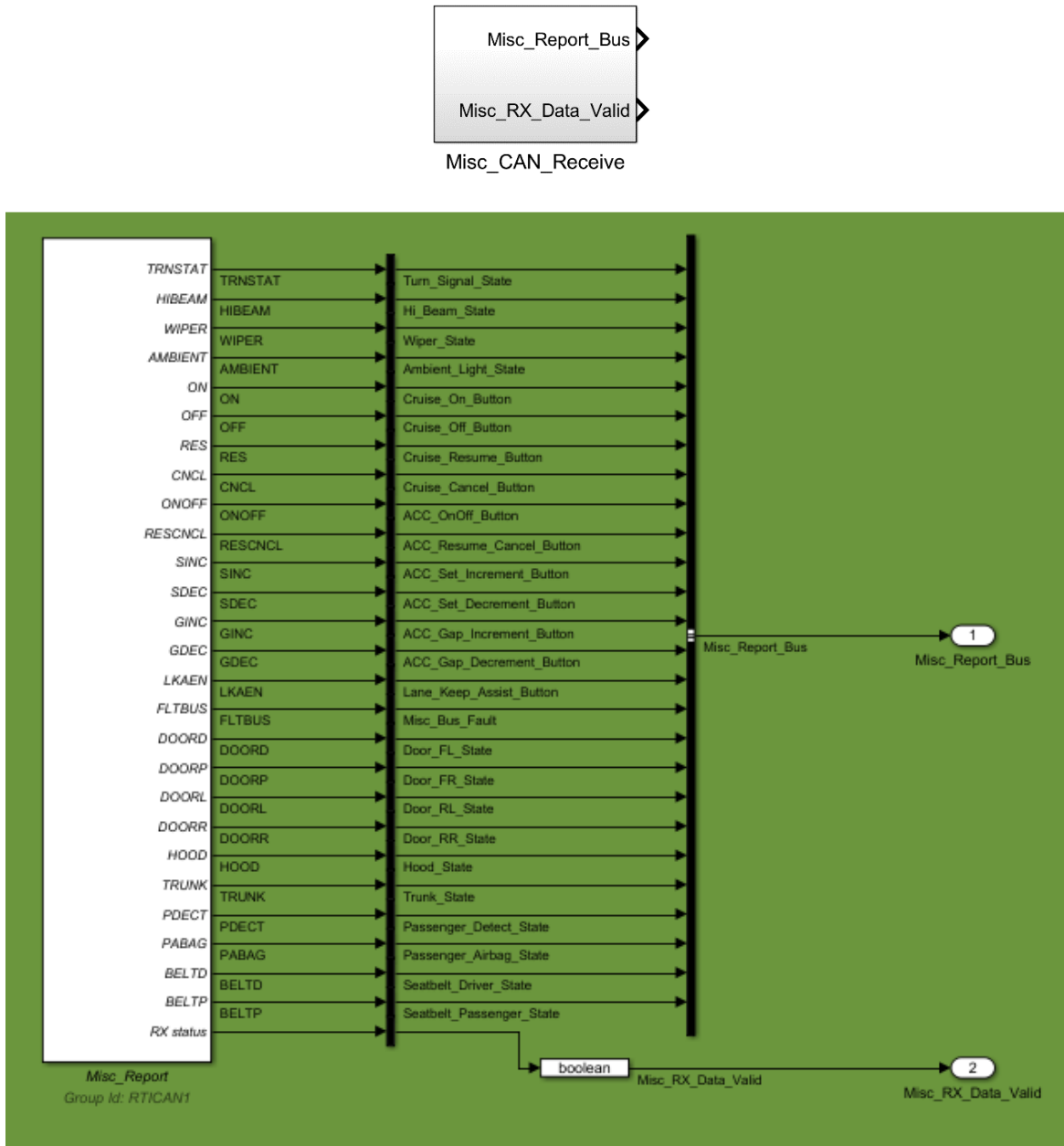


Figure 28. **Misc\_CAN\_Receive** block.

## 4.8. BrakeInfo Report Data

The **BrakeInfo\_Report\_CAN\_Receive** block is shown in *Figure 29*, and its I/O is described in Table 13.

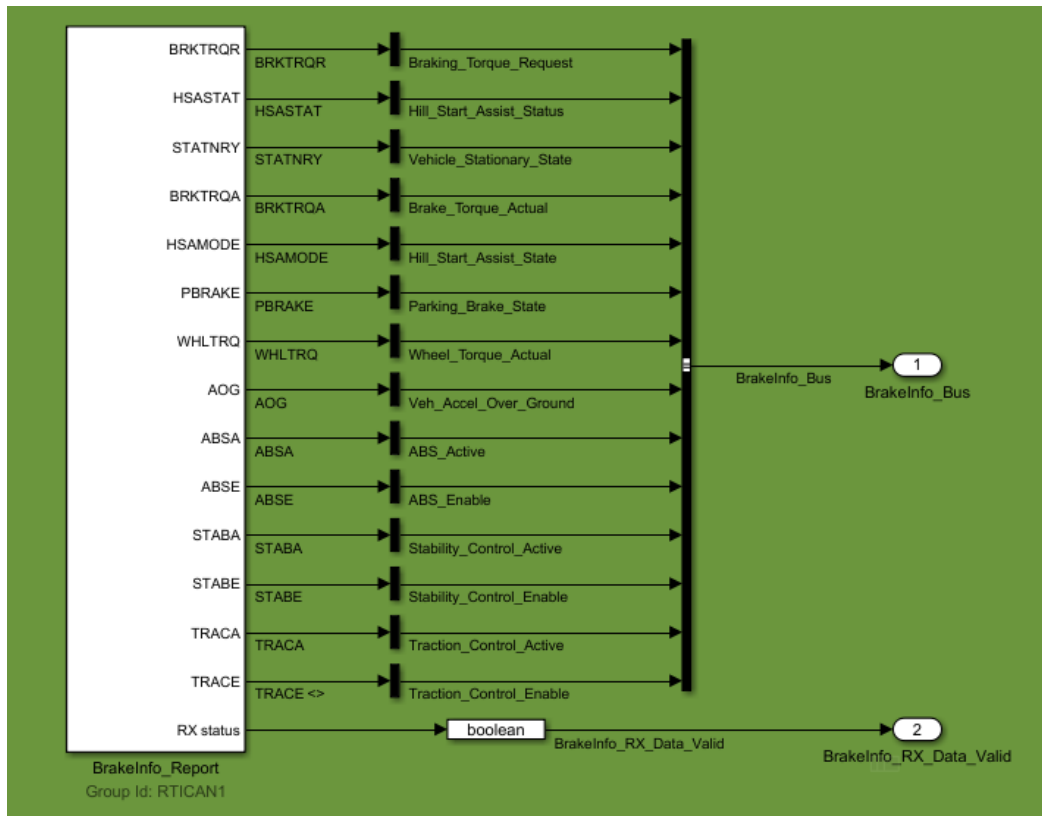
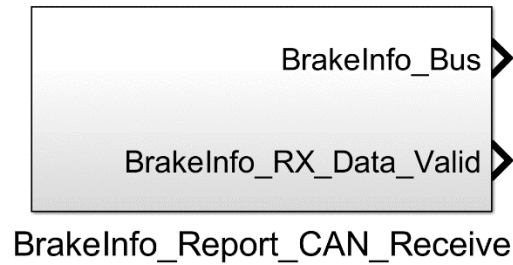


Figure 29. **BrakelInfo\_Report\_CAN\_Receive** block.

Table 13: Port I/O of the **BrakelInfo\_CAN\_Receive** library block.

	Port Name	Data Type	Range	Description
Outputs	BrakelInfo_Report_Bus	bus	—	Simulink bus containing the information in the Miscellaneous Report CAN message (ID = 0x69). See the Steering-Shifter module datasheet for details ( <a href="#">SteeringShifterDatasheet-RevA12.pdf</a> ).
	BrakelInfo_RX_Data_Valid	bool	—	Flag indicating if data is being received over the CAN network.

---

## 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)

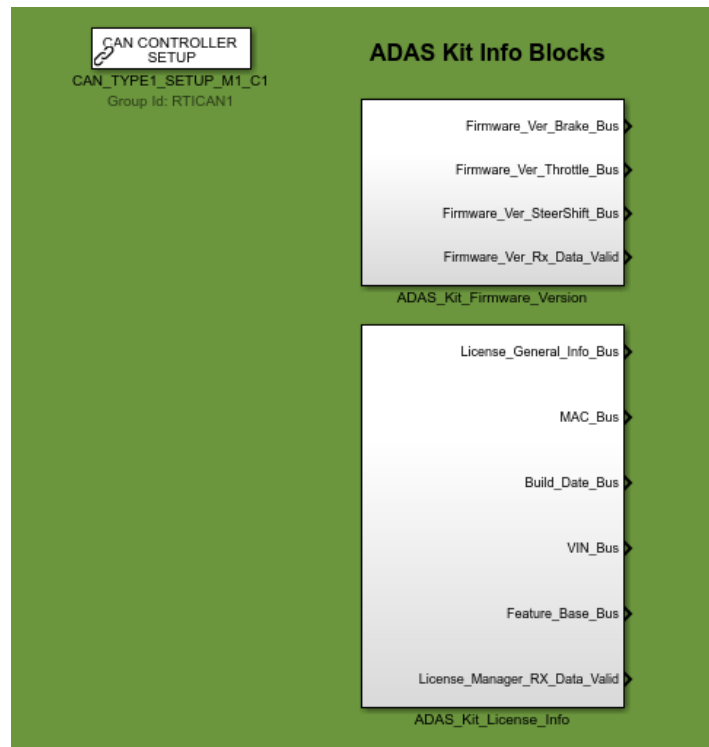


Figure 30. ADAS Kit License Info Library.

### 5.1. ADAS Kit Firmware Version

The **ADAS\_Kit\_Firmware\_Version** block is shown in Figure 31, and its output is described in Table 6.

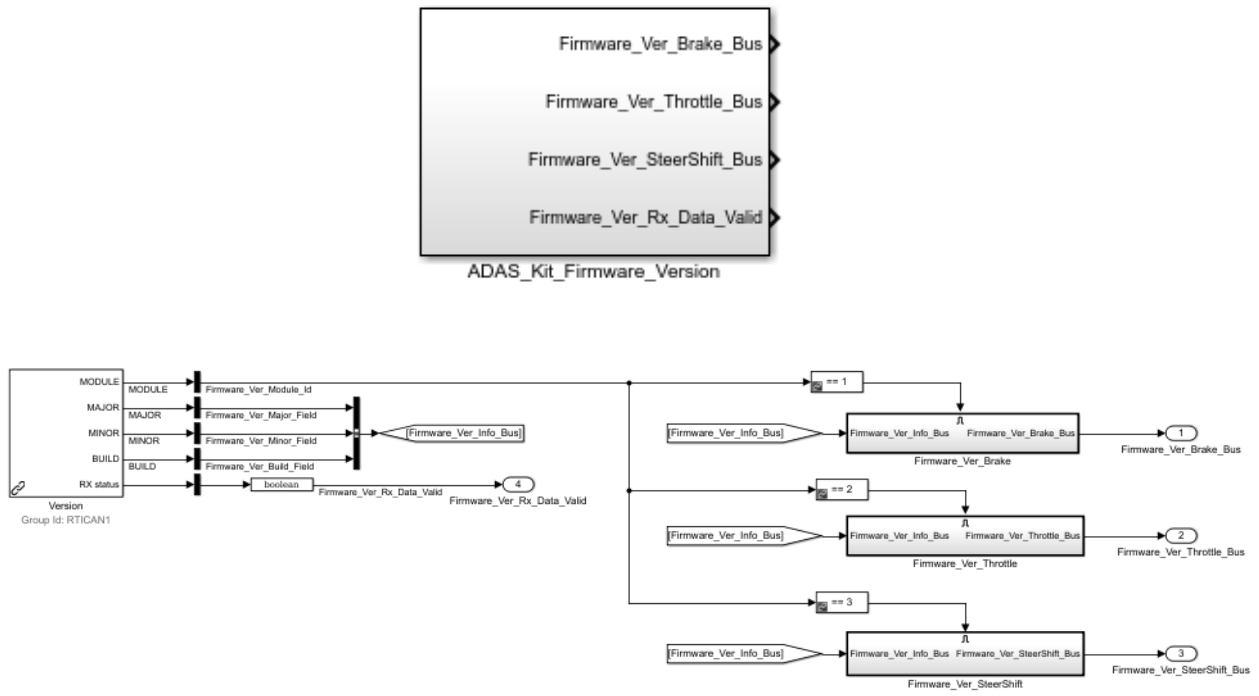


Figure 31. **ADAS\_Kit\_Firmware\_Version** block.

Table 14. Port I/O of the **ADAS\_Kit\_Firmware\_Version** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	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.
	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 32, and its I/O is described in Table 15. Details of each bus are shown in Figure 33 through Figure 37.

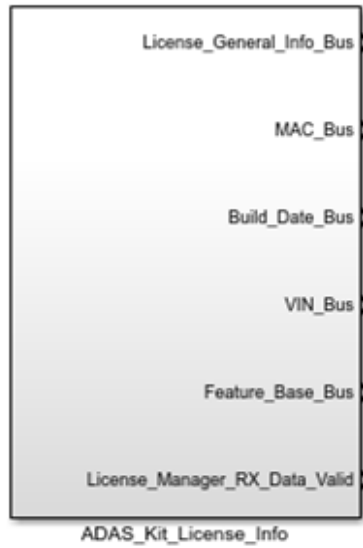


Figure 32. **ADAS\_Kit\_License\_Info** block.

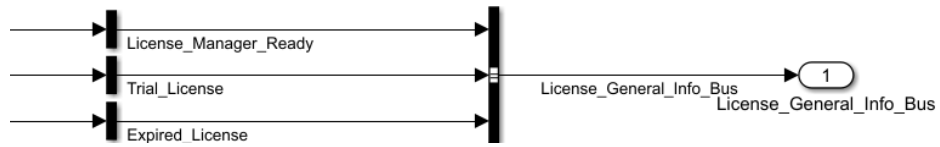


Figure 33. **License\_General\_Info** bus.

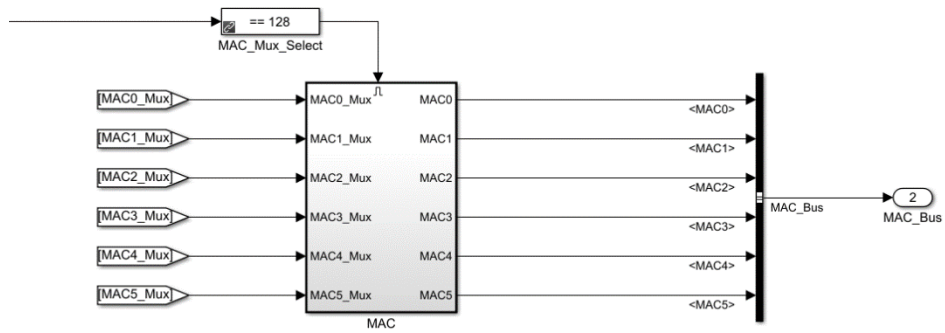


Figure 34. **MAC** bus.

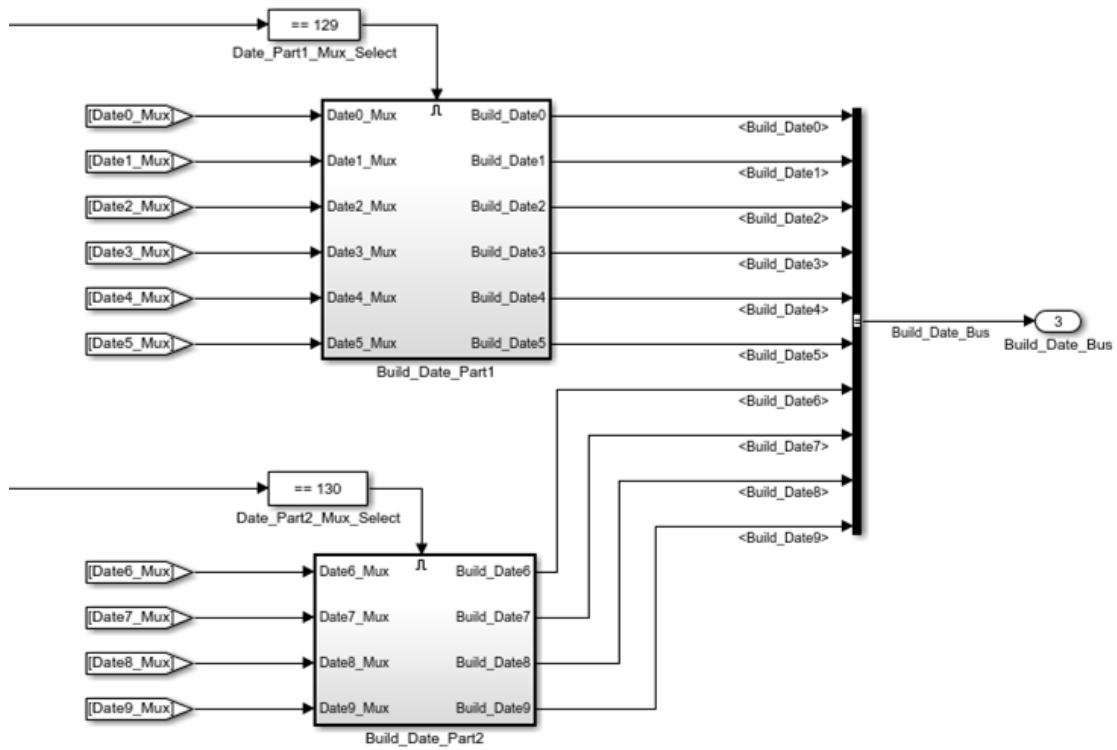


Figure 35. **Build\_Date** bus.



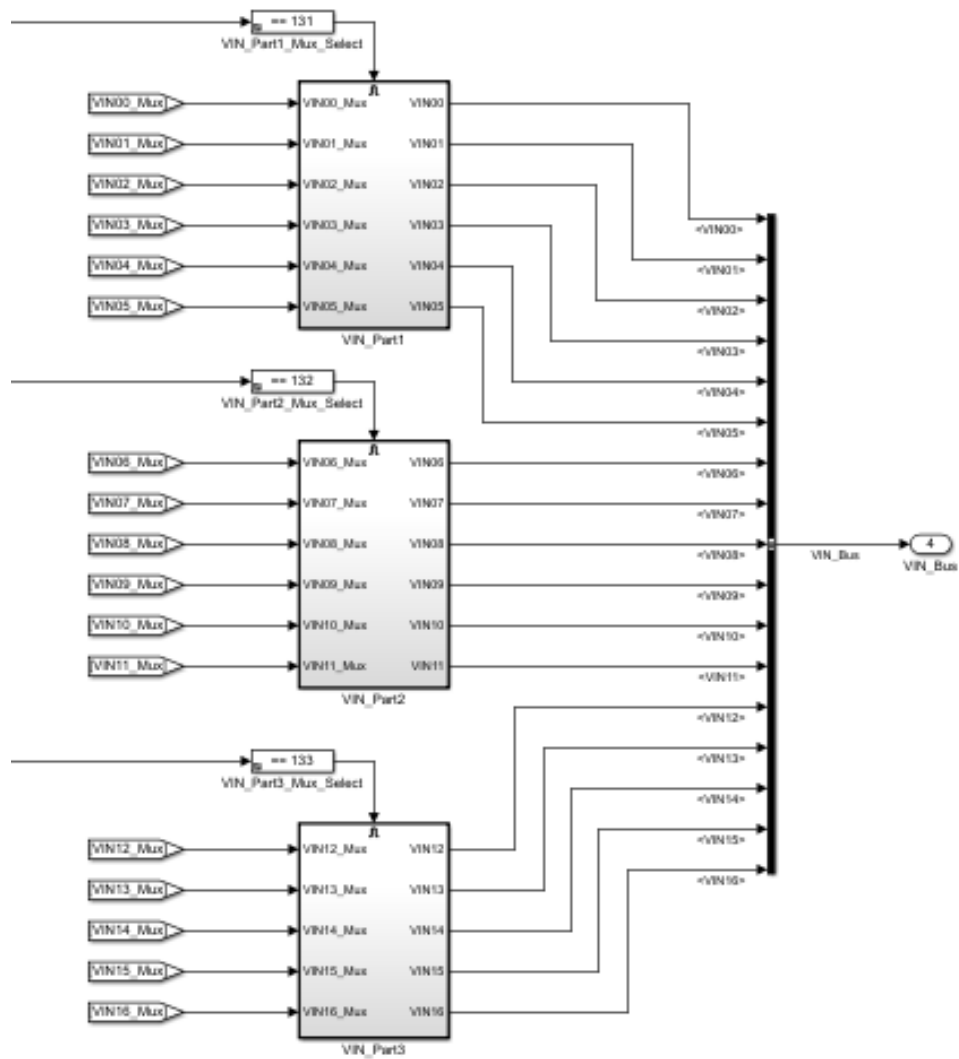


Figure 36. VIN bus.

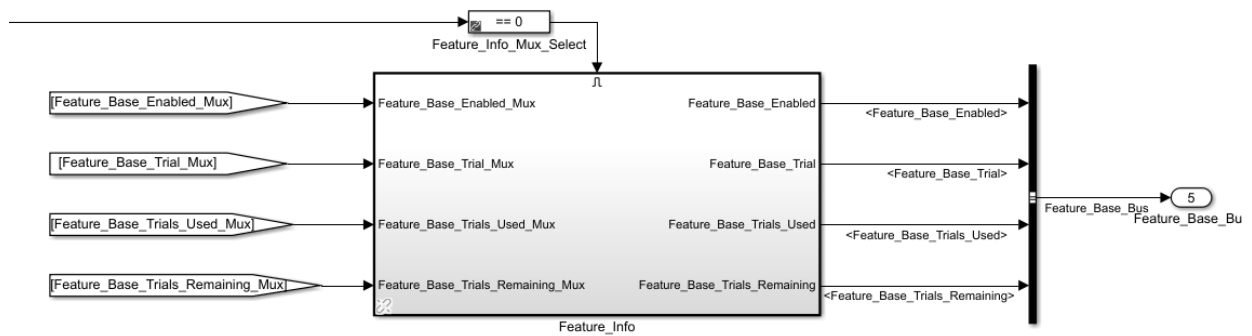


Figure 37. Feature\_Base bus.

Table 16. Port I/O of the **ADAS\_Kit\_License\_Info** library block.

	Port Name	Data Type	Range	Description
<b>Outputs</b>	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. Control Blocks

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

The library is shown in Figure 38. 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.

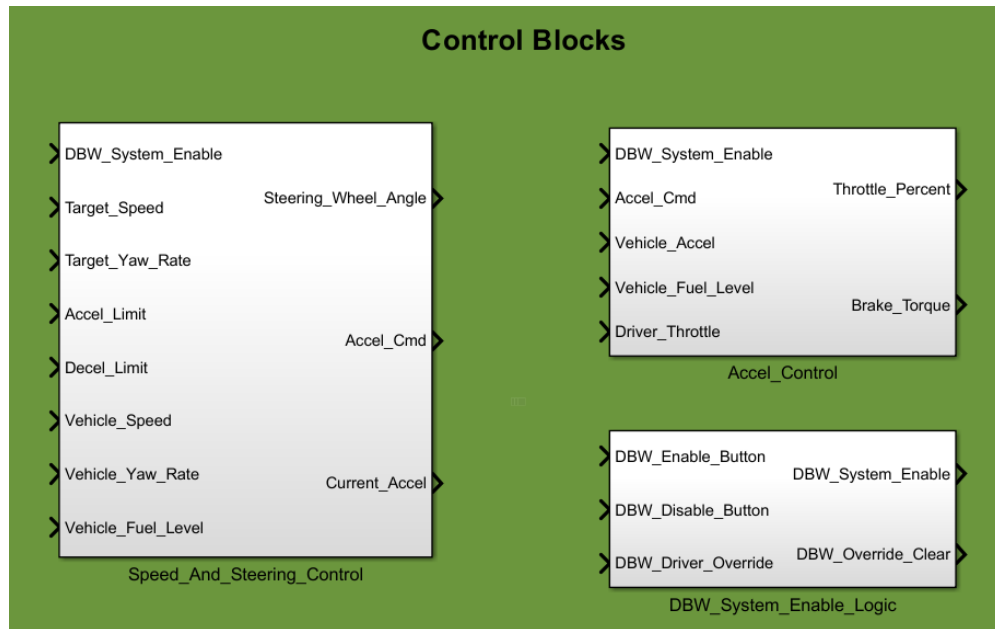


Figure 38. Individual Control Blocks library.

## 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 39, and its I/O is summarized in Table 17.

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).

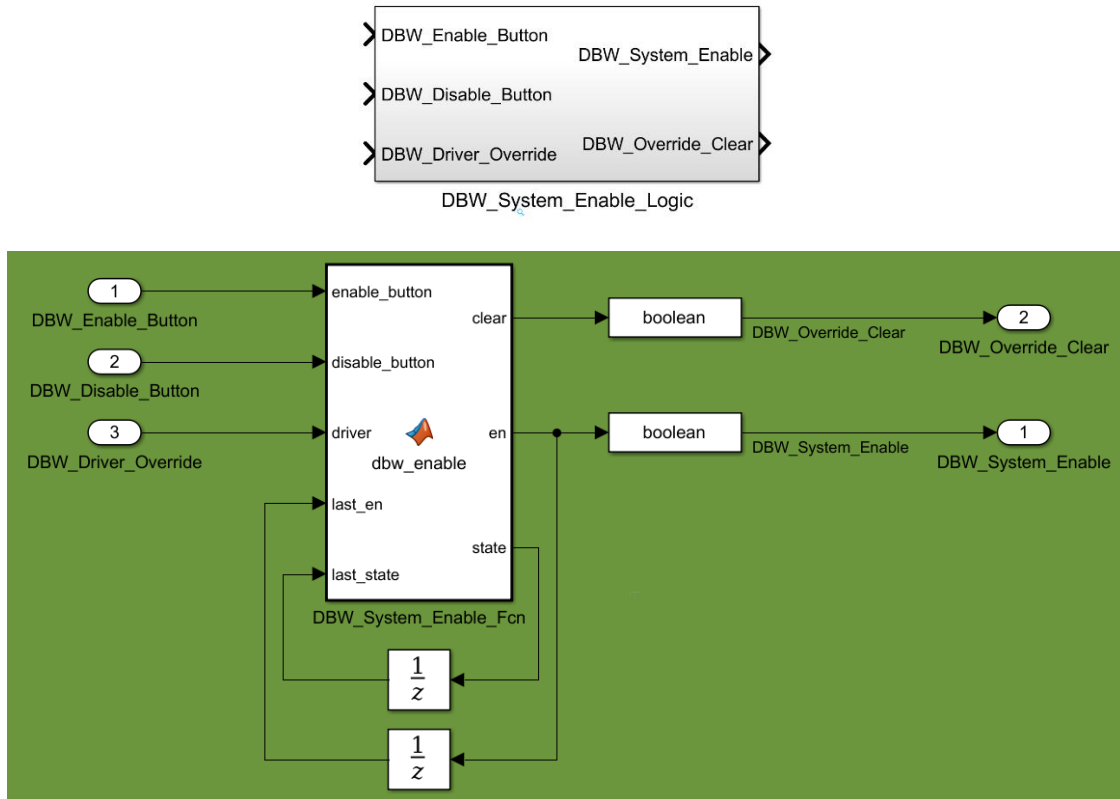


Figure 39. **DBW\_System\_Enable\_Logic** block.

## 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 40, and its I/O is summarized in Table 18. 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 17. Port I/O of the **DBW\_System\_Enable\_Logic** library block.

Port Name	Data Type	Range	Description
-----------	-----------	-------	-------------

<b>Inputs</b>	DBW_Enable_Button	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.
<b>Outputs</b>	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.

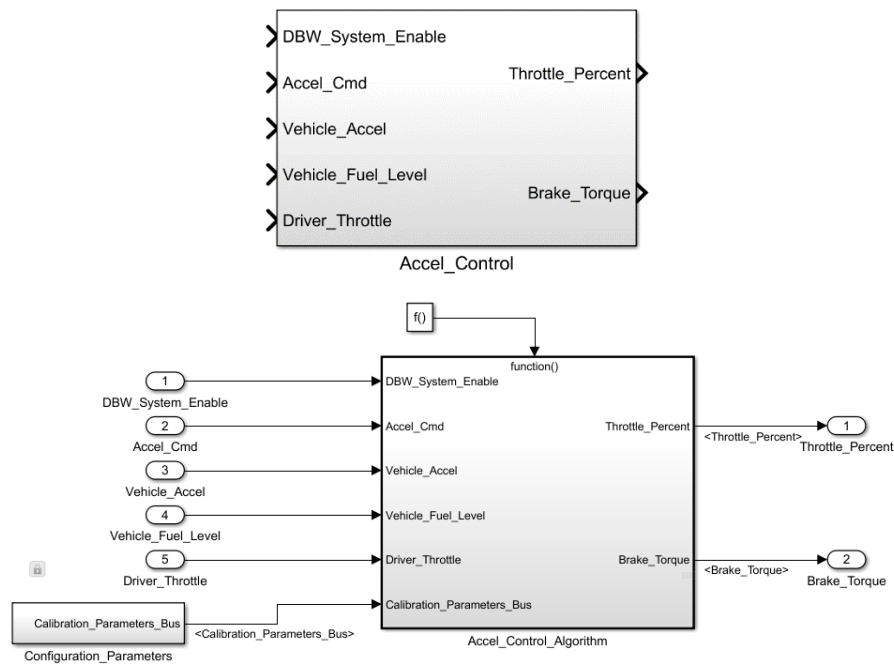


Figure 40. **Accel\_Control** block.

### 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 41, and its I/O is summarized in Table 19. 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 18: Port I/O of the **Accel\_Control** library block.

	Port Name	Data Type	Range	Description
<b>Inputs</b>	DBW_System_Enable	bool	—	Boolean signal indicating if DBW system is enabled. Expected to come from the <b>DBW_System_Enable_Logic</b> 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 <b>Fuel_Level_Receive</b> 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 <b>Throttle_Pedal_Physical</b> signal on the <b>Throttle_Report_Bus</b> (Section 3.1).
<b>Outputs</b>	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 <b>twist_controller.slx</b> example model (Section 7). Look for the <b>Throttle_Percent_To_Command</b> embedded MATLAB function in the <b>Vehicle_Interface</b> block.
	Brake_Torque_Cmd	double	0 — 3412	Brake torque output in $N\cdot m$ . A lookup-table to map brake torque to brake pedal command is implemented in the <b>twist_controller.slx</b> example model (Section 7). (Look for the <b>Brake_Torque_To_Command</b> embedded MATLAB function in the <b>Vehicle_Interface</b> block.)

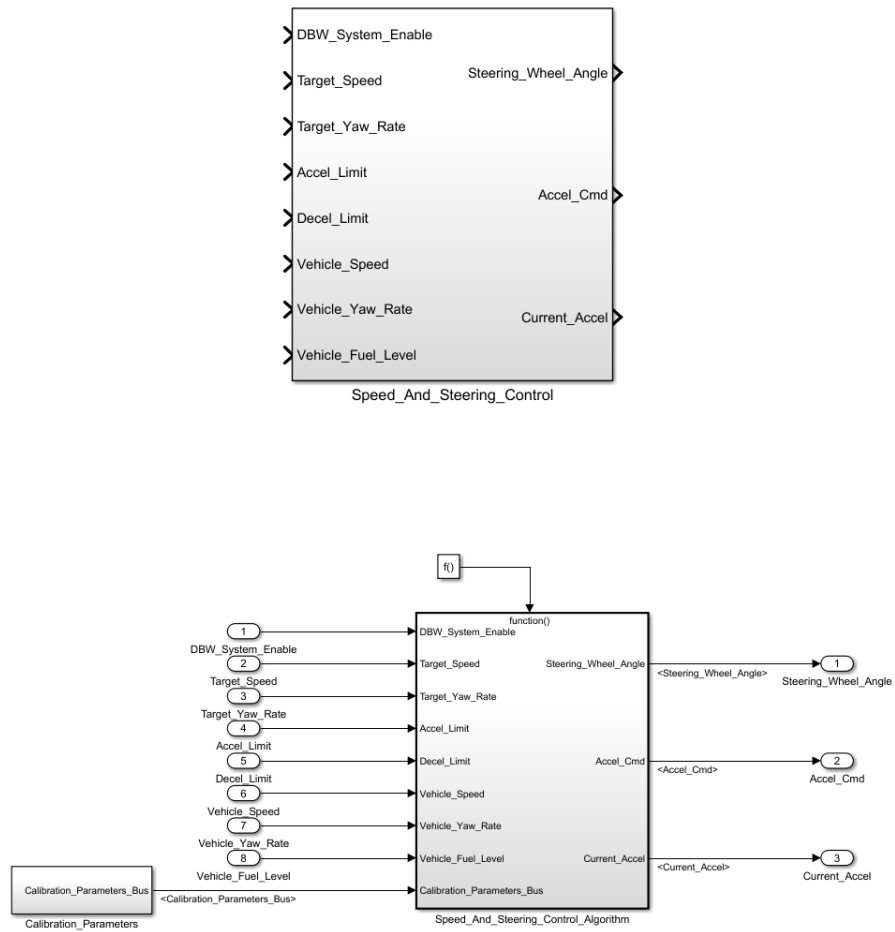


Figure 41. **Speed\_And\_Steering\_Control** block.

Table 19. Port I/O of the **Speed\_And\_Steering\_Control** library block.

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

## 7. Demo Models

The provided demo models include:

- **Model Template** – Provides a starting point from which users can begin to implement their application around a model that is pre-configured with Dataspeed DBW block libraries.
- **Joystick Teleop** – Uses the signals from a USB video game joystick to control the steering, brakes, throttle and shifter.
- **Twist Controller** – Implements closed-loop control of throttle, brakes, and steering to track user-specified forward speed and yaw-rate commands. It is recommended to reference this example model when integrating the **Accel\_Control** and **Speed\_And\_Steering\_Control** blocks (Sections 6.2 and 6.3) into your own application-level model.