Power Window

Study Power Windows

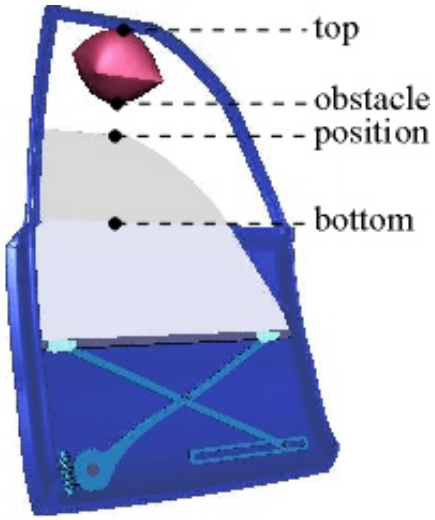
Automobiles use electronics for control operations such as:

Open This Example

- Opening and closing windows and sunroof
- Adjusting mirrors and headlights
- Locking and unlocking doors

These systems are subject to stringent operation constraints. Failures can cause dangerous and possibly life-threatening situations. As a result, careful design and analysis are needed before deployment.

This example focuses on the design of a power window system of an automobile, in particular, the passenger-side window. A critical aspect of this system is that it cannot exert a force of more than 100 N on an object when the window closes. When the system detects such an object, it must lower the window by about 10 cm.



As part of the design process, the example considers:

- Quantitative requirements for the window control system, such as timing and force requirements
- System requirements, captured in activity diagrams
- Data definitions for the signals used in activity diagrams

Other aspects of the design process that this example contains are:

- Managing the components of the system
- Building the model
- Validating the results of system simulation
- Generating code

MathWorks Software Used in This Example

In addition to Simulink, this example uses these additional MathWorks® products:

- DSP System Toolbox™
- Fixed-Point Designer™
- Simscape™ Multibody™
- Simscape Power Systems™
- Simscape

- Simulink® 3D Animation™
- Simulink Real-Time™
- Simulink Verification and Validation™
- Stateflow®

Quantitative Requirements

Quantitative requirements for the control are:

- The window must fully open and fully close within 4 s.
- If the up is issued for between 200 ms and 1 s, the window must fully open. If the down command is issued for between 200 ms and 1 s, the window must fully close.
- The window must start moving 200 ms after the command is issued.
- The force to detect when an object is present is less than 100 N.
- When closing the window, if an object is in the way, stop closing the window and lower the window by approximately 10 cm.

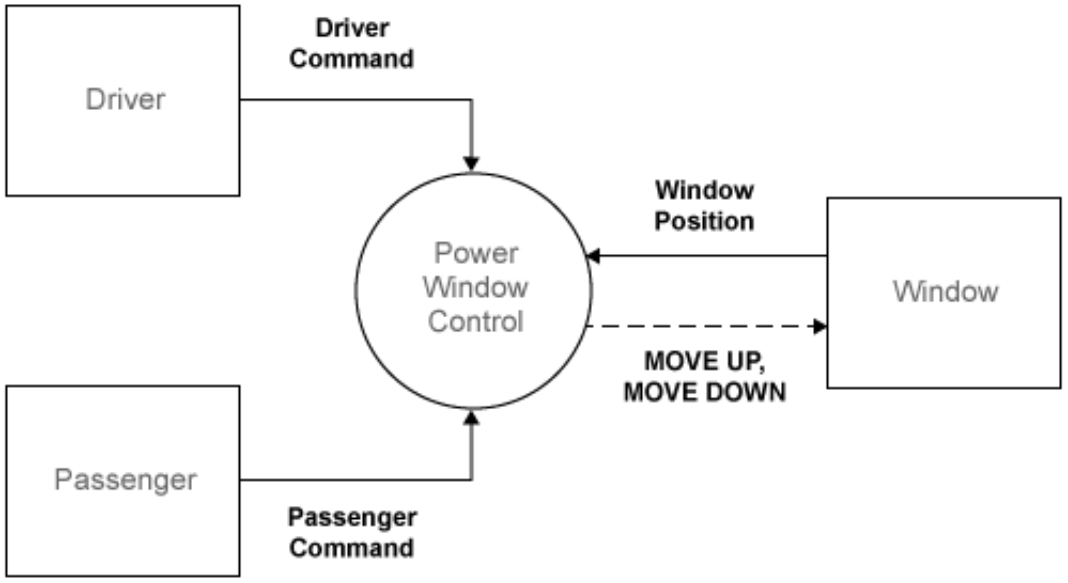
Capturing Requirements in Activity and Context Diagrams

Activity diagrams help you graphically capture the specification and understand how the system operates. A hierarchical structure helps with analyzing even large systems. At the top level, a context diagram describes the system environment and its interaction with the system under study in terms of data exchange and control operations. Then you can decompose the system into an activity diagram with processes and control specifications (CSPEC).

The processes guide the hierarchical decomposition. You specify each process using another activity diagram or a primitive specification (PSPEC). You can specify a PSPEC in a number of representations with a formal semantic, such as a Simulink block diagram. In addition, context diagrams graphically capture the context of system operation.

Context Diagram: Power Window System

The figure represents the context diagram of a power window system. The square boxes capture the environment, in this case, the driver, passenger, and window. Both the driver and passenger can send commands to the window to move it up and down. The controller infers the correct command to send to the window actuator (e.g., the driver command has priority over the passenger command). In addition, diagram monitors the state of the window system to establish when the window is fully opened and closed and to detect if there is an object between the window and frame.

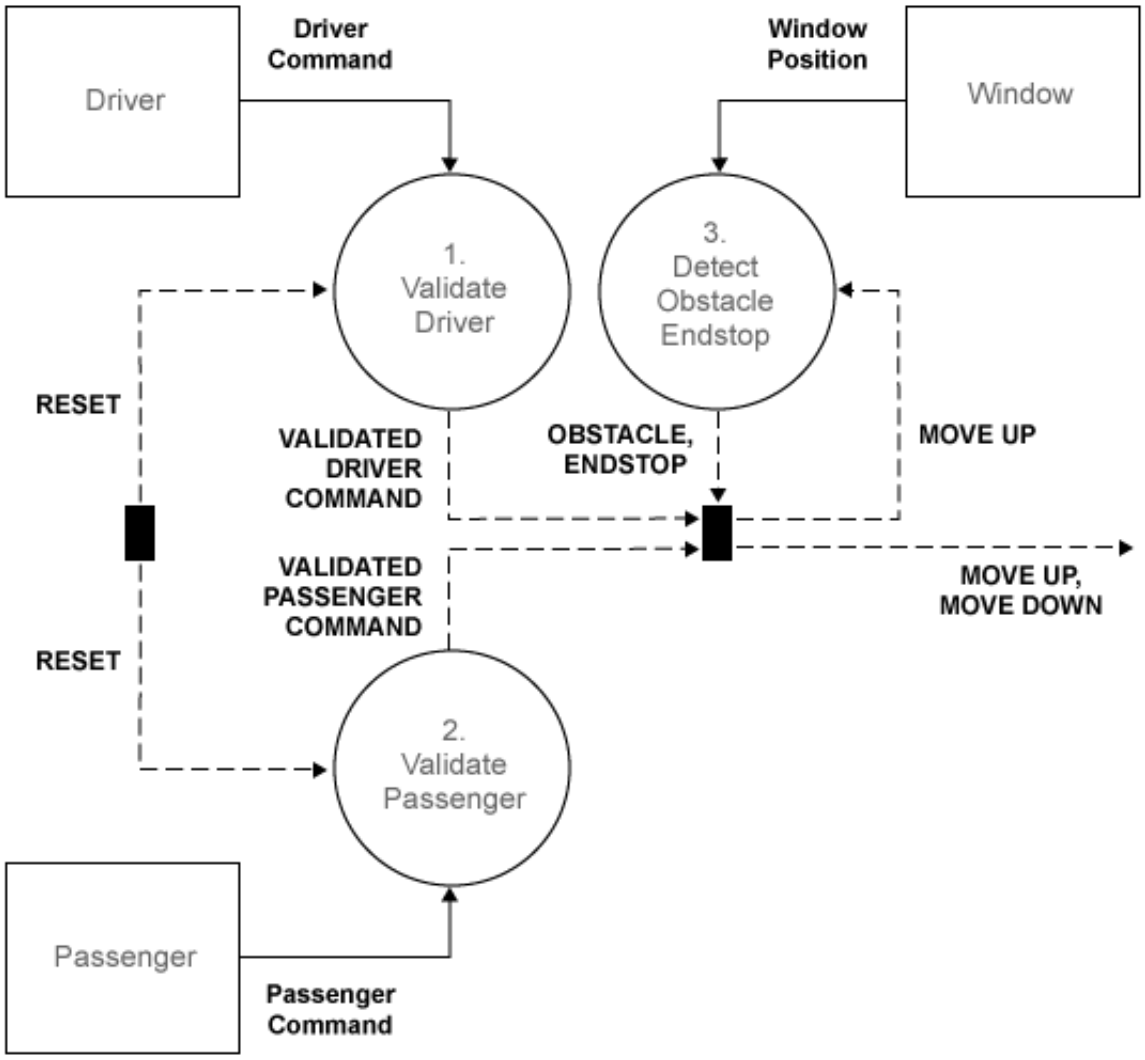


The circle (also known as a bubble) represents the power window controller. The circle is the graphical notation for a process. Processes capture the transformation of input data into output data. Primitive process might also generate. CSPECs typically consist of combinational or sequential logic to infer output control signals from input control.

For implementation in the Simulink environment, see Implementation of Context Diagram: Power Window System.

Activity Diagram: Power Window Control

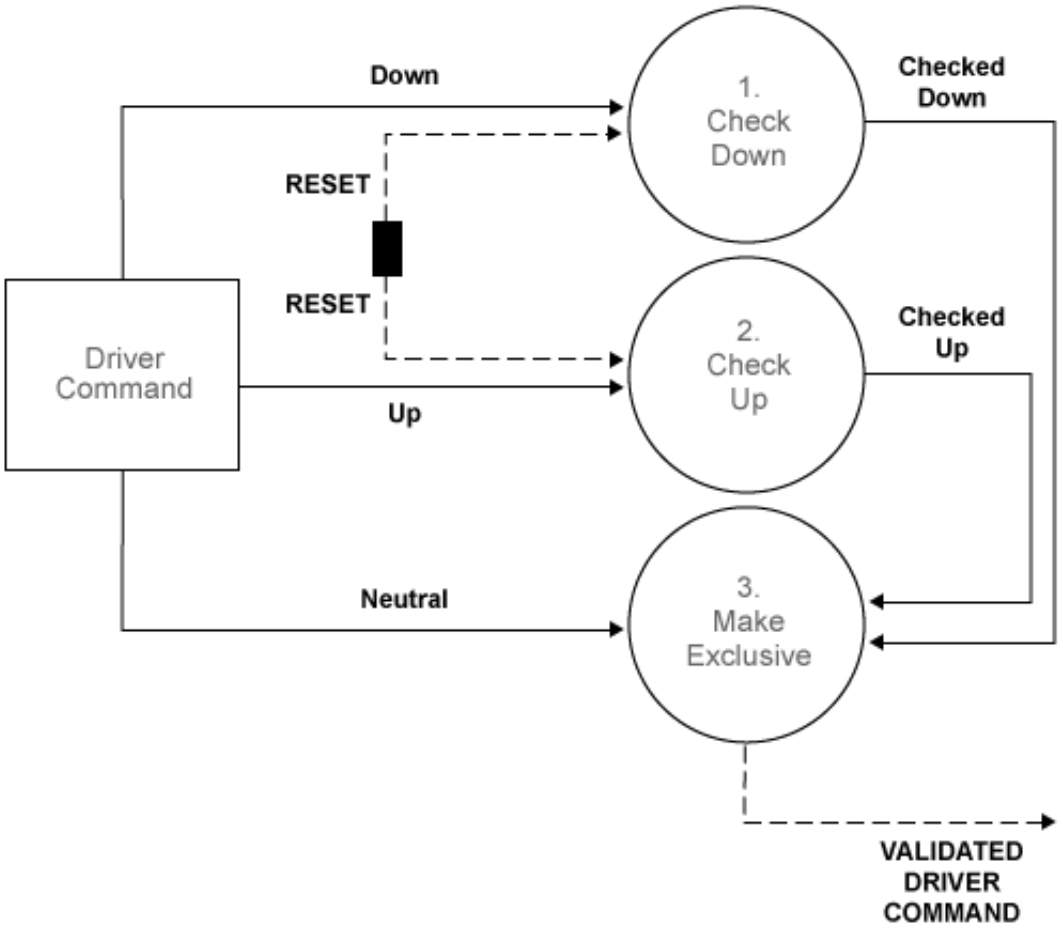
The power window control consists of three processes and a CSPEC. Two processes validate the driver and passenger input to ensure that their input is meaningful given the state of the system. For example, if the window is completely opened, the `MOVE DOWN` command does not make sense. The remaining process detects if the window is completely opened or completely closed and if an object is present. The CSPEC takes the control signals and infers whether to move the window up or down (e.g., if an object is present, the window moves down for about one second or until it reaches an endstop).



For implementation in the Simulink environment, see Implementation of Activity Diagram: Power Window Control.

Activity Diagram: Validate Driver

Each process in the VALIDATE DRIVER activity chart is primitive and specified by the following PSPEC. In the MAKE EXCLUSIVE PSPEC, for safety reasons the `DOWN` command takes precedence over the `UP` command.

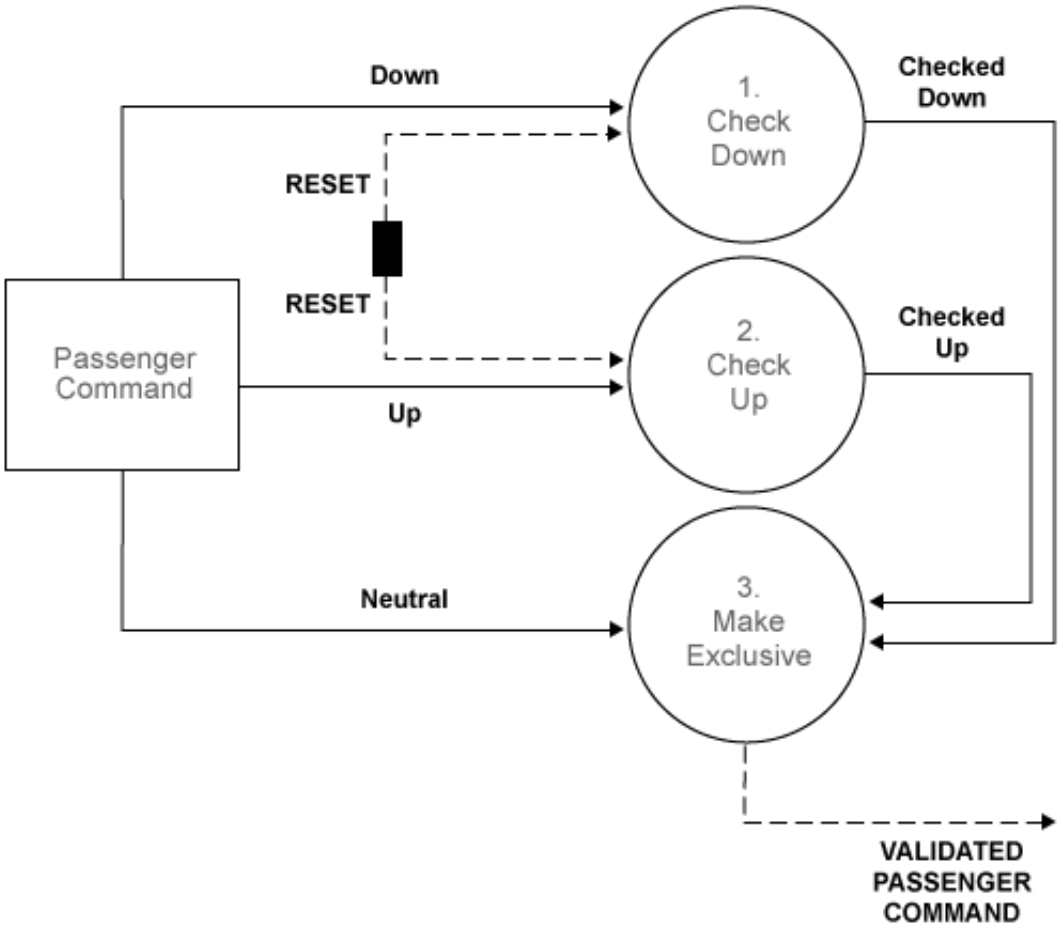



```
PSPEC 1.1.1: CHECK DOWN
  CHECKED_DOWN = DOWN and not RESET

PSPEC 1.1.2: CHECK UP
  CHECKED_UP = UP and not RESET

PSPEC 1.1.3: MAKE EXCLUSIVE
  VALIDATED_DOWN    = CHECKED_DOWN
  VALIDATED_UP      = CHECKED_UP and not CHECKED_DOWN
  VALIDATED_NEUTRAL = (NEUTRAL and not (CHECKED_UP and not CHECKED_DOWN))
                        or not (CHECKED_UP or CHECKED_DOWN)
```

For implementation in the Simulink environment, see Implementation of Activity Diagram: Validate.

Activity Diagram: Validate Passenger

The internals of the VALIDATE PASSENGER process are the same as the VALIDATE DRIVER process. The only difference is the different input and output.

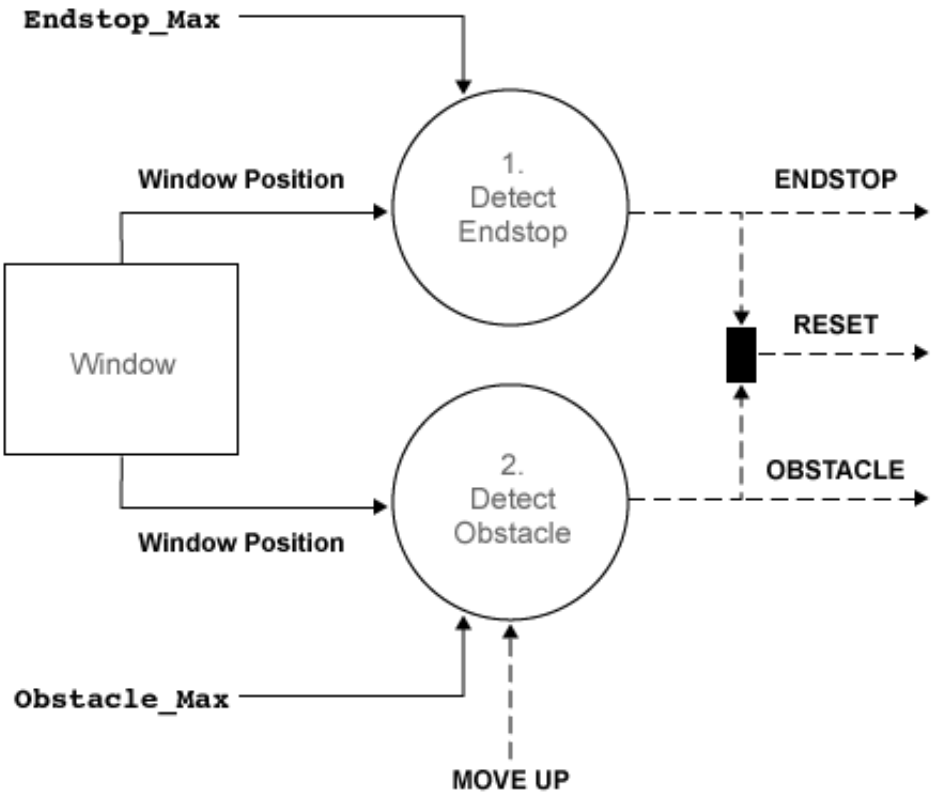



```
PSPEC 1.2.1: CHECK DOWN
  CHECKED_DOWN = DOWN and not RESET

PSPEC 1.2.2: CHECK UP
  CHECKED_UP = UP and not RESET

PSPEC 1.2.3: MAKE EXCLUSIVE
  VALIDATED_DOWN    =  CHECKED_DOWN
  VALIDATED_UP      =  CHECKED_UP and not CHECKED_DOWN
  VALIDATED_NEUTRAL = (NEUTRAL and not (CHECKED_UP and not CHECKED_DOWN))
                        or not (CHECKED_UP or CHECKED_DOWN)
```

For implementation in the Simulink environment, see Activity Diagram: Validate Passenger.

Activity Diagram: Detect Obstacle Endstop

The third process in the POWER WINDOW CONTROL activity diagram detects the presence of an obstacle or when the window reaches its top or bottom (ENDSTOP). The detection mechanism is based on the armature current of the window actuator. During normal operation, this current is within certain bounds. When the window reaches its top or bottom, the electromotor draws a large current (more than 15 A or less than –15 A) to try and sustain its angular velocity. Similarly, during normal operation the current is about 2 A or –2 A (depending on whether the window is opening or closing). When there is an object, there is a slight deviation from this value. To keep the window force on the object less than 100 N, the control switches to its emergency operation when it detects a current that is less than –2.5 A. This operations is necessary only when the window is rolling up, which corresponds to a negative current in the particular wiring of this model. The DETECT OBSTACLE ENDSTOP activity diagram embodies this functionality.




```
CSPEC 1.3: DETECT OBSTACLE ENDSTOP
  RESET = OBSTACLE or ENDSTOP

PSPEC 1.3.1: DETECT ENDSTOP
  ENDSTOP = WINDOW_POSITION > ENDSTOP_MAX

PSPEC 1.3.2: DETECT OBSTACLE
  OBSTACLE = (WINDOW_POSITION > OBSTACLE_MAX) and MOVE_UP for 500 ms
```

For implementation in the Simulink environment, see Activity Diagram: Detect Obstacle Endstop.

Data Definitions

The functional decomposition unambiguously specifies each process by its decomposition or primitive specification (PSPEC). In addition, it must also formally specify the signals in the activity diagrams. Use data definitions for these specifications.

The following tables are data definitions for the signals used in the activity diagrams.

For the associated activity diagram, see Context Diagram: Power Window System.

Context Diagram: Power Window System Data Definitions

Signal	Information Type	Continuous/ Discrete	Data Type	Values
DRIVER_COMMAND	Data	Discrete	Aggregate	`Neutral`, `up`, `down`
PASSENGER_COMMAND	Data	Discrete	Aggregate	`Neutral`, `up`, `down`
WINDOW_POSITION	Data	Continuous	Real	0 to 0.4 m
MOVE_UP	Control	Discrete	Boolean	`'True'`, `'False'`
MOVE_DOWN	Control	Discrete	Boolean	`'True'`, `'False'`

For the associated activity diagram, see Activity Diagram: Power Window Control.

Activity Diagram: Power Window Control Data Definitions

Signal	Information Type	Continuous/ Discrete	Data Type	Values
DRIVER_COMMAND	Data	Discrete	Aggregate	`Neutral, up, down`
PASSENGER_COMMAND	Data	Discrete	Aggregate	`Neutral, up, down`
WINDOW_POSITION	Data	Continuous	Real	0 to 0.4 m
MOVE_UP	Control	Discrete	Boolean	`'True', 'False'`
MOVE_DOWN	Control	Discrete	Boolean	`'True', 'False'`

For the associated activity diagram, see Activity Diagram: Validate Driver.

Activity Diagram: Validate Driver Data Definitions

Signal	Information Type	Continuous/ Discrete	Data Type	Values
DRIVER_COMMAND	Data	Discrete	Aggregate	`Neutral, up, down`
PASSENGER_COMMAND	Data	Discrete	Aggregate	`Neutral, up, down`
WINDOW_POSITION	Data	Continuous	Real	0 to 0.4 m
MOVE_UP	Control	Discrete	Boolean	`'True', 'False'`
MOVE_DOWN	Control	Discrete	Boolean	`'True', 'False'`

For the associated activity diagram, see Activity Diagram: Validate Passenger.

Activity Diagram: Validate Passenger Data Definitions

Signal	Information Type	Continuous/ Discrete	Data Type	Values
NEUTRAL	Data	Discrete	Boolean	`'True', 'False'`
UP	Data	Discrete	Boolean	`'True', 'False'`
DOWN	Data	Discrete	Boolean	`'True', 'False'`
CHECKED_UP	Data	Discrete	Boolean	`'True', 'False'`
CHECKED_DOWN	Data	Discrete	Boolean	`'True', 'False'`

For the associated activity diagram, see Activity Diagram: Detect Obstacle Endstop.

Activity Diagram: Detect Obstacle Endstop Data Definitions

Signal	Information Type	Continuous/ Discrete	Data Type	Values
ENDSTOP_MIN	Data	Constant	Real	0.0 m
ENDSTOP_MAX	Data	Constant	Real	0.4 m
OBSTACLE_MAX	Data	Constant	Real	0.3 m

The model design iterates as we examine more detailed implementations. For information about how the model design iterates as you introduce more detail, see Iterate on the Design.

Simulink Power Window Controller in Simulink Project

- Explore the Power Window Controller Project

MATLAB® and Simulink support Model-Based Design for embedded control design, from initial specification to code generation. To organize large projects and share your work with others, use Simulink Projects.

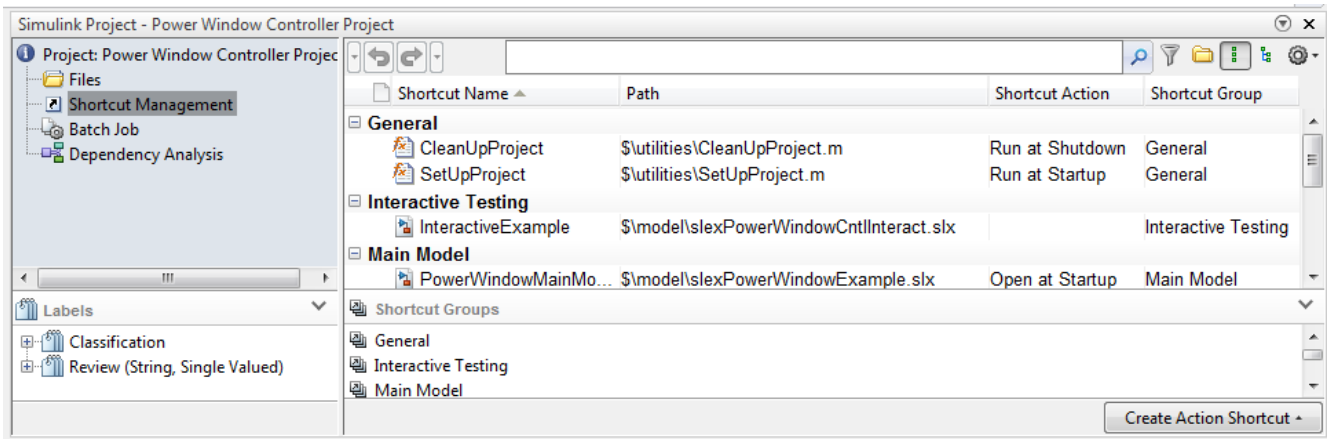
The Power Window Control Project example shows how you can use MathWorks tools and the Model-Based Design process to go from concept through implementation for an automobile power window system. It uses Simulink Projects to organize the files and other model components.

In addition, this example shows how to link models to system documentation.

Explore the Power Window Controller Project

1. To open the Power Window Controller project, in the MATLAB Command Window, type:

   ```
   slexPowerWindowStart
   ```



2. Explore the project folders. In particular, note the **task** folders. This folder contains scripts that run frequent tasks for a model. For the Power Window Controller Project, these scripts:

   - Set up the model to control window movement on a controller area network (CAN).
   - Set up the model to use the Stateflow and Simulink software to model discrete-event reactive behavior and continuous time behavior, with a low-order plant model.
   - Set up the model with a more detailed plant model that includes power effects in the electrical and mechanical domains. The plant model validates the force exerted by the window on a trapped object.
   - Set up the model with a model that includes other effects that may change the model, such as quantization of the measurements.

   > **Note:** These scripts also simulate the model. To only configure the model, see the scripts in the **configureModel** folder.

   - Use the increase coverage model to generate the model coverage report.

3. The **Shortcut Management** section contains quick-access commands that you can double-click to perform common tasks such as:

   - Set up and clean up projects.
   - Add projects to MATLAB paths.
   - Perform interactive testing.
   - Validate model testing with model coverage.
   - Open the main model.
   - Simulate the model with various configurations.
   - Generate a model coverage report for increased coverage of the model.
   - Open the model used for increasing model coverage.

Simulink Power Window Controller

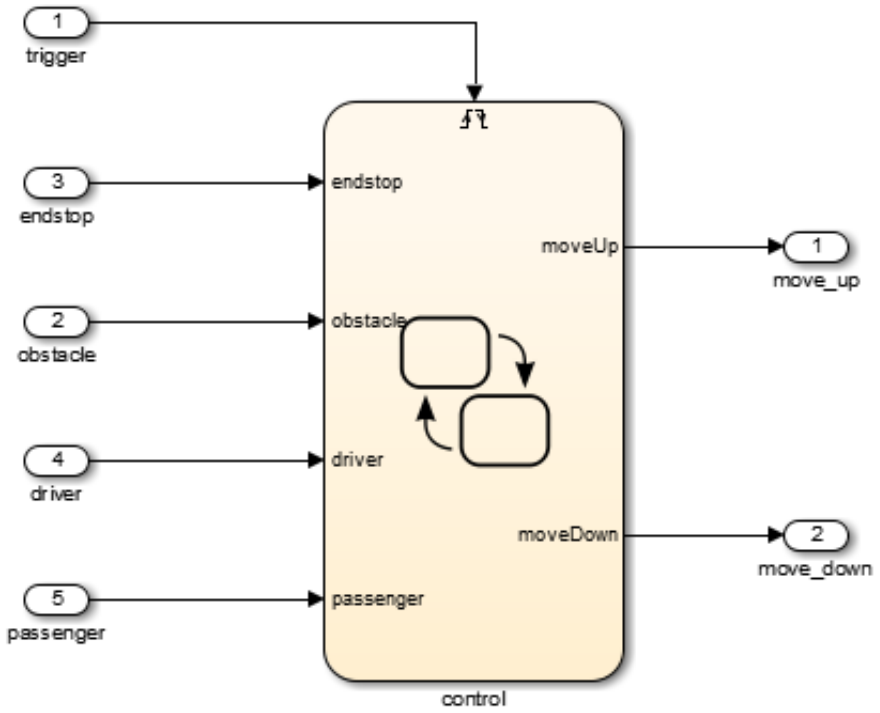
Implementation of Activity Diagram: Power Window Control

This topic describes the high-level discrete-event control specification for a power window control.

You can model the discrete-event control of the window with a Stateflow chart. A Stateflow chart is a finite state machine with hierarchy and parallelism. This state machine contains the basic states of the power window system: up, auto-up, down, auto-down, rest, and emergency. It models the state transitions and accounts for the precedence of driver commands over the passenger commands. It also includes emergency behavior that activates when the software detects an object between the window and the frame while moving up.

The initial Simulink model for the power window control, `slexPowerWindowControl`, is a discrete-event controller that runs at a given sample rate.

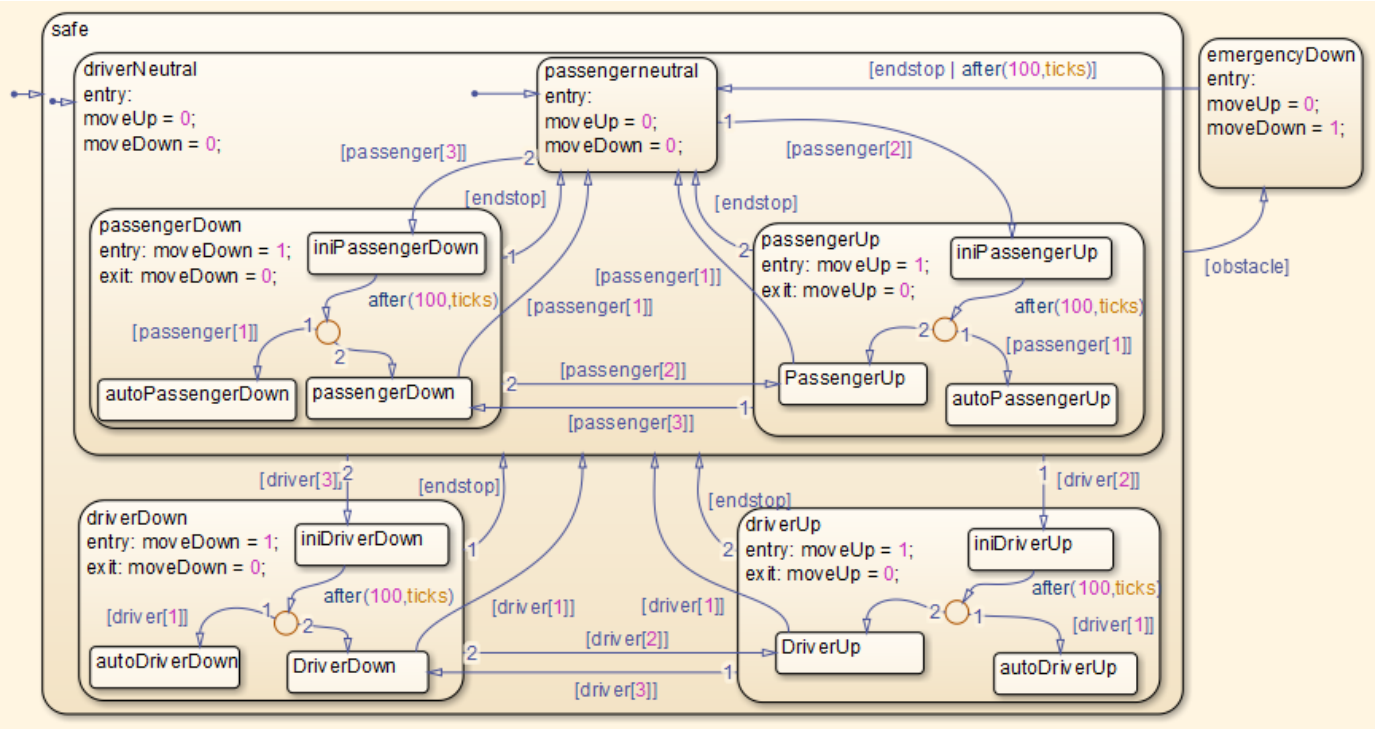
In this implementation, open the power window control subsystem and observe that the Stateflow chart with the discrete-event control forms the CSPEC, represented by the tilted thick bar in the bottom right corner. The detect_obstacle_endstop subsystem encapsulate the threshold detection mechanisms.



The discrete-event control is a Stateflow model that extends the state transition diagram notion with hierarchy and parallelism. State changes because of passenger commands are encapsulated in a super state that does not correspond to an active driver command.

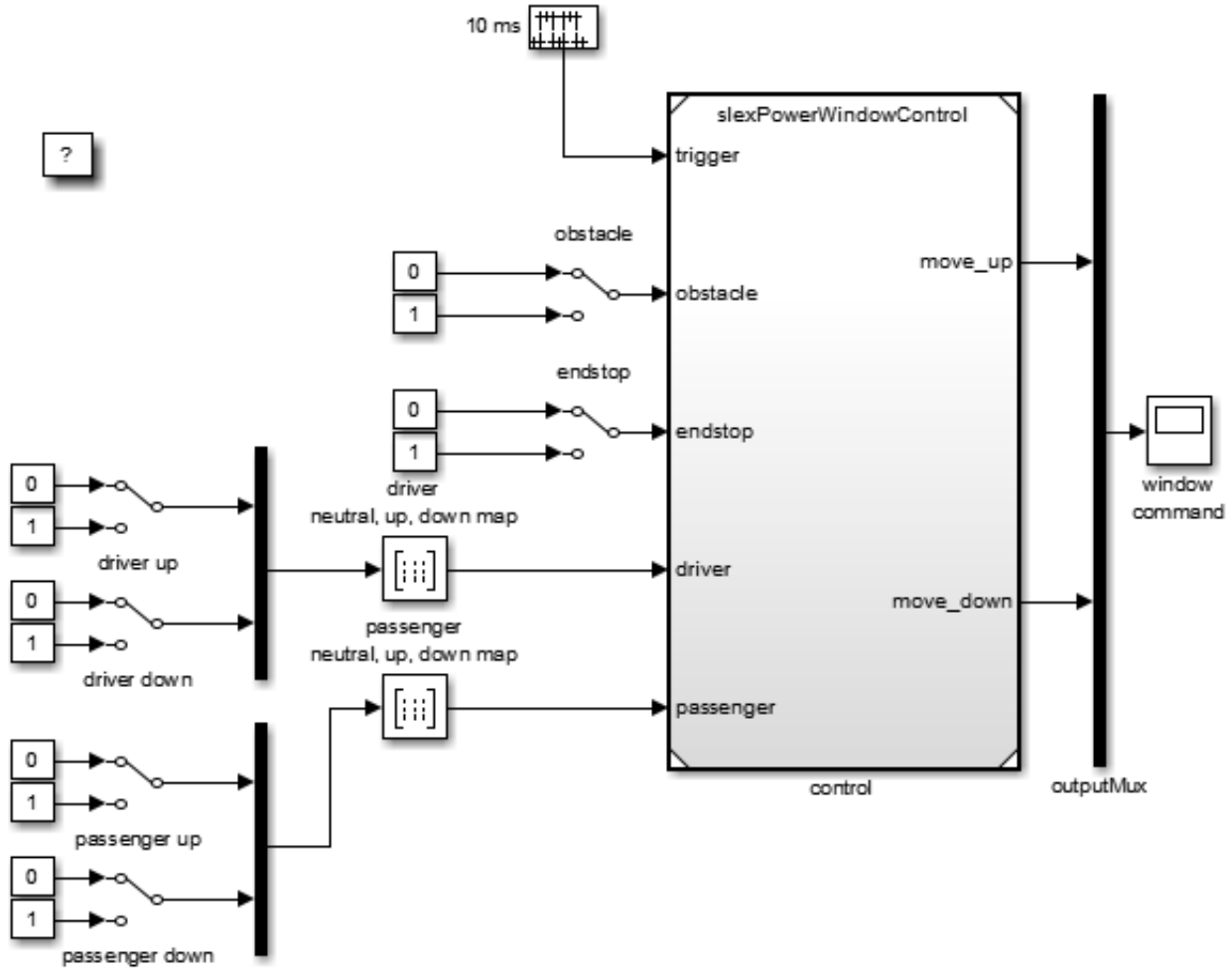
Consider the control of the passenger window. The passenger or driver can move this window up and down.

This state machine contains the basic states of the power window system: up, auto-up, down, auto-down, rest, and emergency.



Interactive Testing

Control Input. The `slexPowerWindowCntlInteract` model includes this control input as switches. Double-click these switches to manually operate them.



Test the state machine that controls a power window by running the input test vectors and checking that it reaches the desired internal state and generates output. The power window has the following external inputs:

- Passenger input

- Driver input
- Window up or down
- Obstacle in window

Each input consists of a vector with these inputs.

Passenger Input

Element	Description
`neutral`	Passenger control switch is not depressed.
`up`	Passenger control switch generates the up signal.
`down`	Passenger control switch generates the down signal.

Driver Input

Element	Description
`neutral`	Driver control switch is not depressed.
`up`	Driver control switch generates the up signal.
`down`	Driver control switch generates the down signal.

Window Up or Down

Element	Description
`0`	Window moves freely between top or bottom.
`1`	Window is stuck at the top or bottom because of physical limitations.

Obstacle in Window

Element	Description
`0`	Window moves freely between top or bottom.
`1`	Window has obstacle in the frame.

Generate the passenger and driver input signals by mapping the up and down signals according to this table:

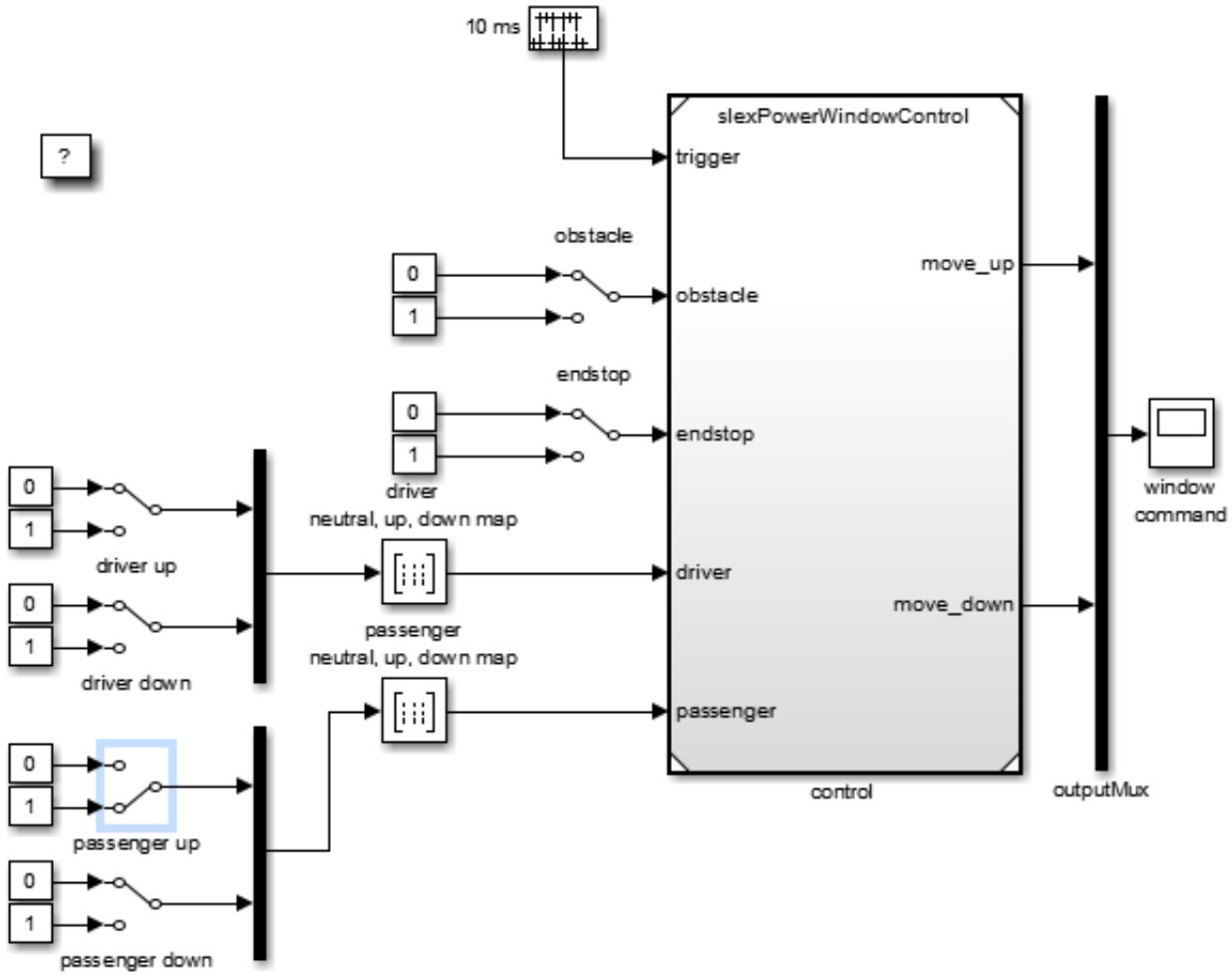
Inputs		Outputs		
up	**down**	**up**	**down**	**neutral**
`0`	`0`	`0`	`0`	`1`
`0`	`1`	`0`	`1`	`0`
`1`	`0`	`1`	`0`	`0`
`1`	`1`	`0`	`0`	`1`

The inputs explicitly generate the `neutral` event from the `up` and `down` events, generated by pressing a power window control switch. The inputs are entered as a truth table in the passenger neutral, up, down map and the driver neutral, up, down map.

Experimental Results from Interactive Testing

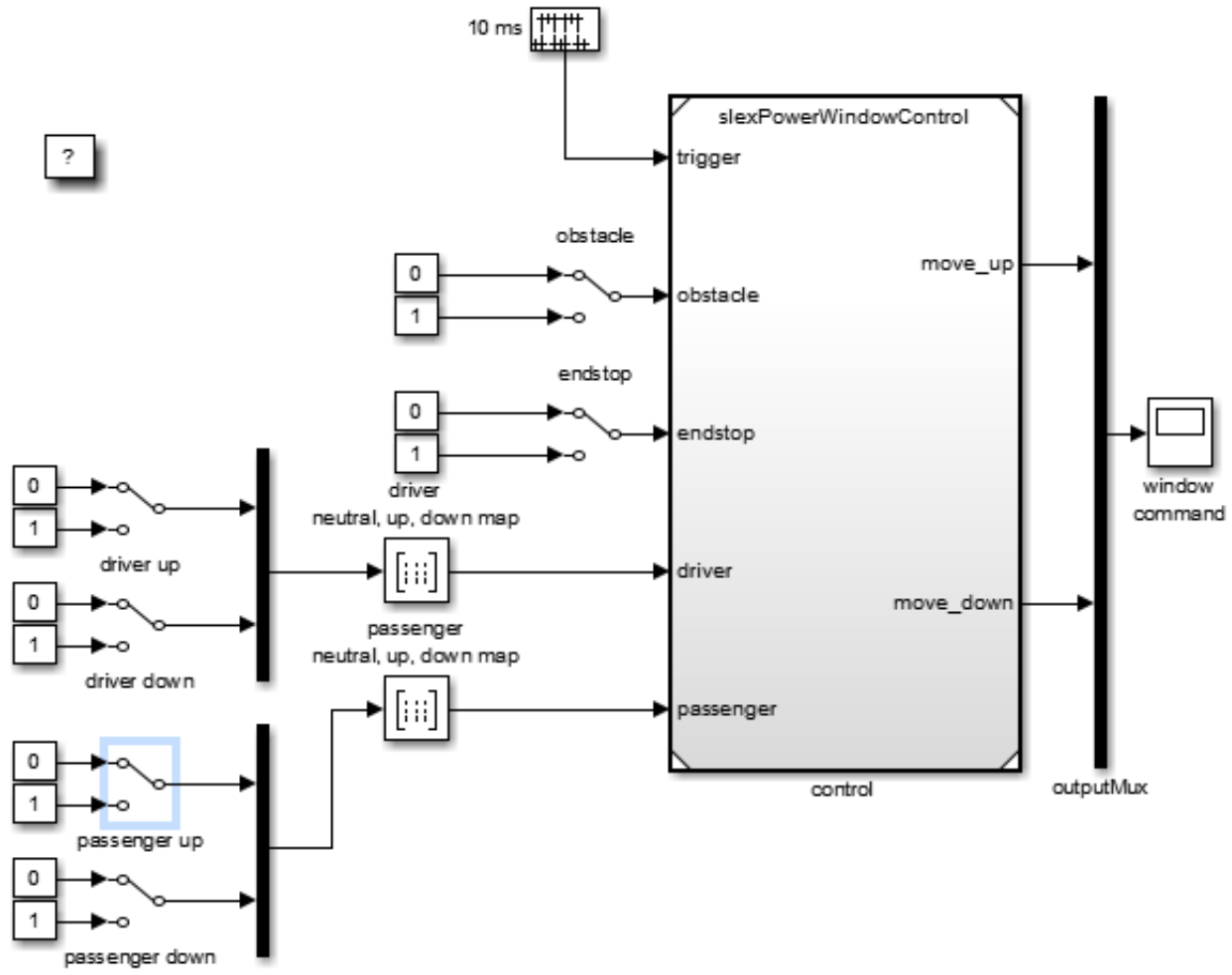
Case 1: Window Up. To observe the state machine behavior:

1. Open the `slexPowerWindowCntlInteract` model.
2. Run the simulation and then double-click the passenger up switch.



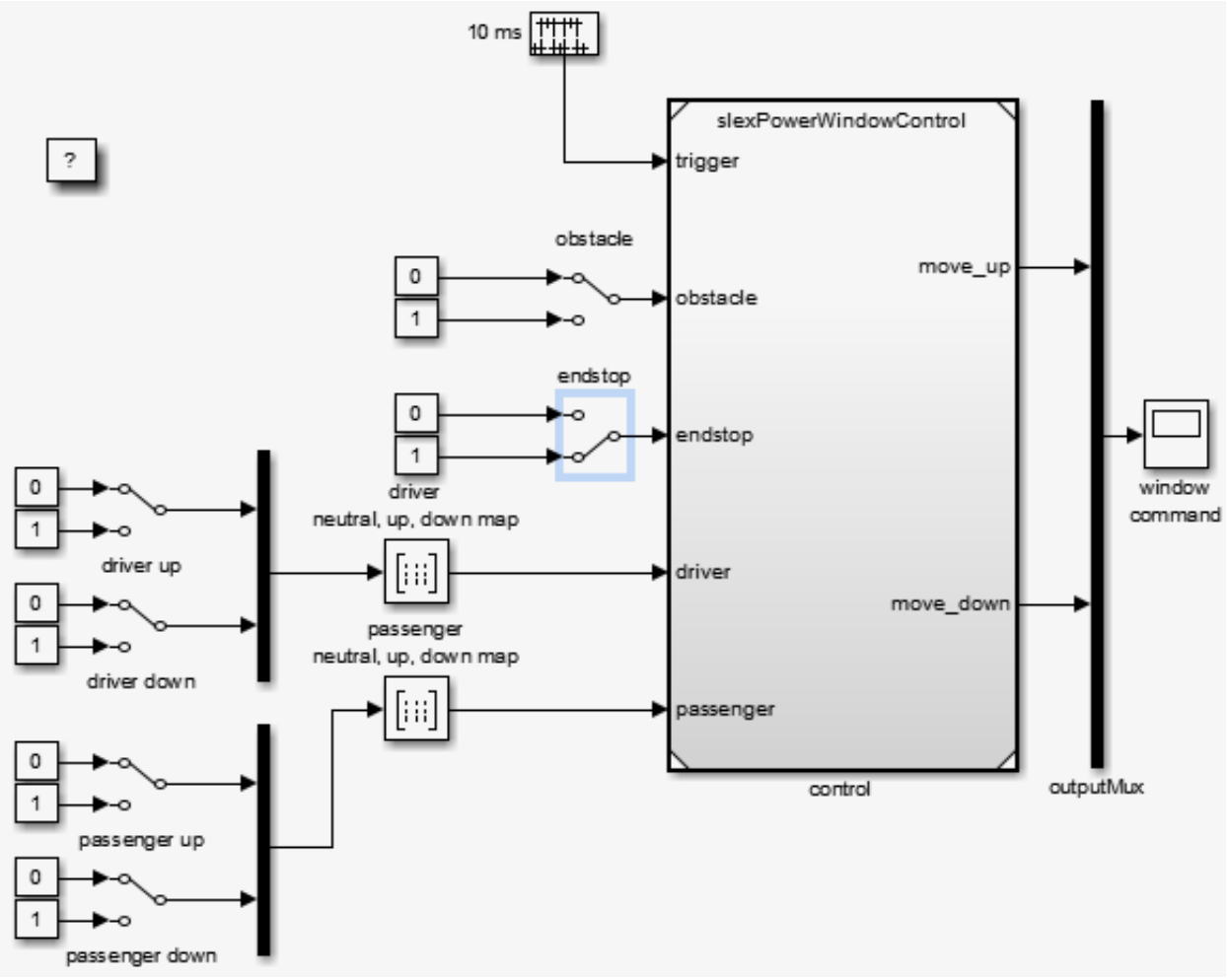
If you press the physical window switch for more than one second, the window moves up until the up switch is released (or the top of the window frame is reached and the `endstop` event is generated).

3. Double-click the selected passenger up switch to release it.



4. Simulate the model.

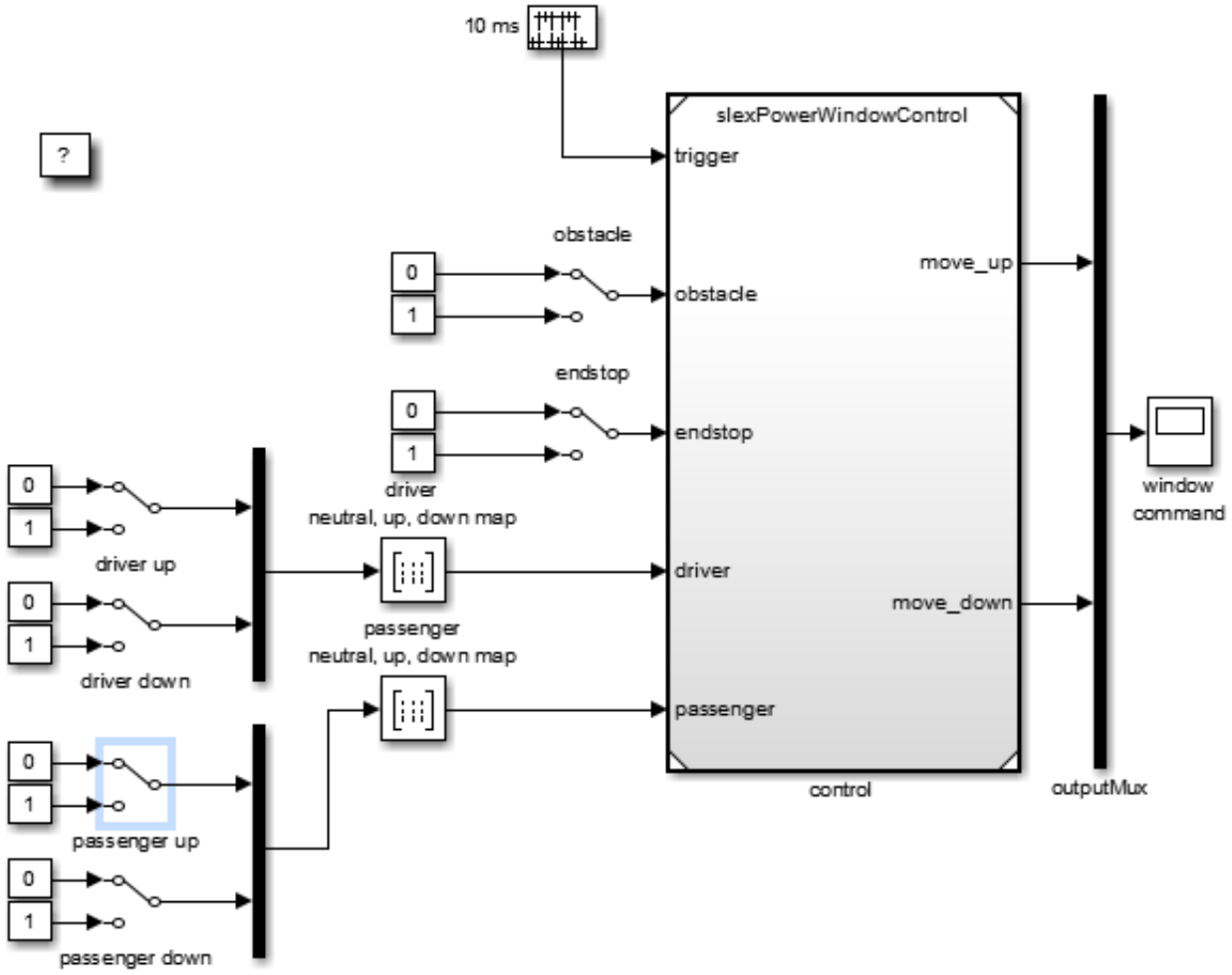
   Setting the endstop switch generates the endstop event.



Case 2: Window Auto-Up. If you press the physical passenger window up switch for a short period of time (less than a second), the software activates auto-up behavior and the window continues to move up.

1. Press the physical passenger window up switch for a short period of time (less than a second).

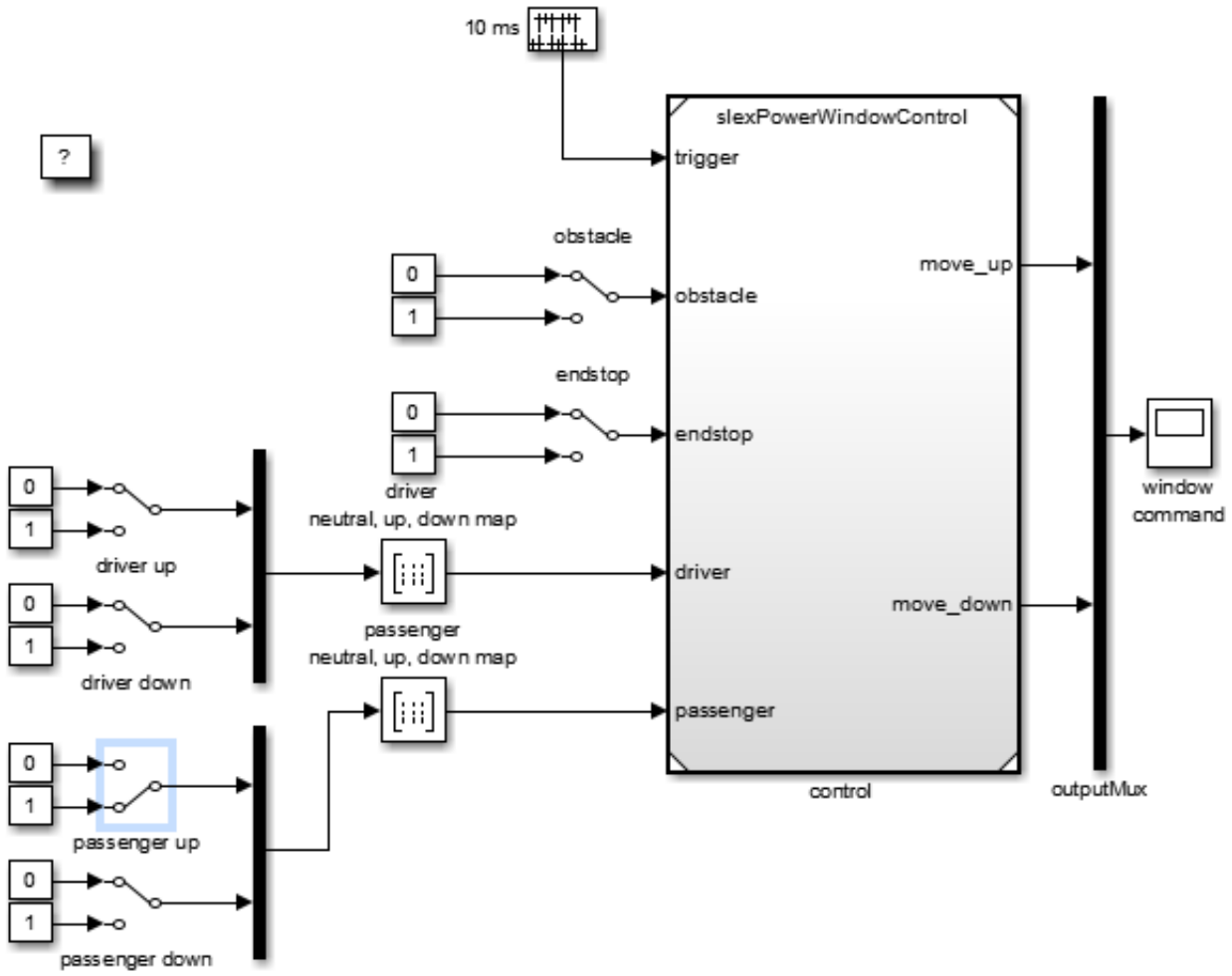
   Ultimately, the window reaches the top of the frame and the software generates the `endstop` event. This event moves the state machine back to its neutral state.



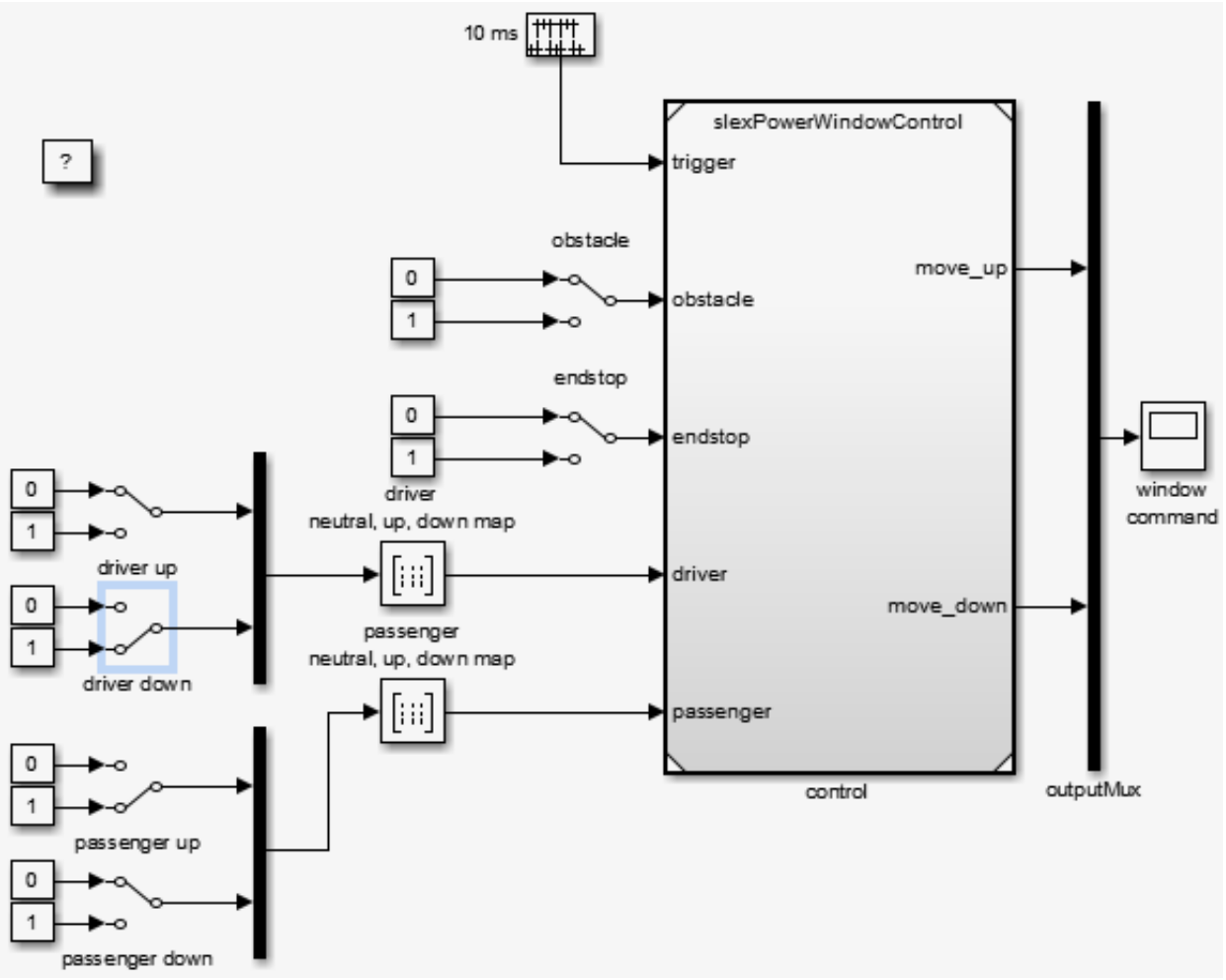
2. Simulate the model.

Case 3: Driver-Side Precedence. The driver switch for the passenger window takes precedence over the driver commands. To observe the state machine behavior in this case:

1. Run the simulation, and then move the system to the `passenger up` state by double-clicking the passenger window up switch.

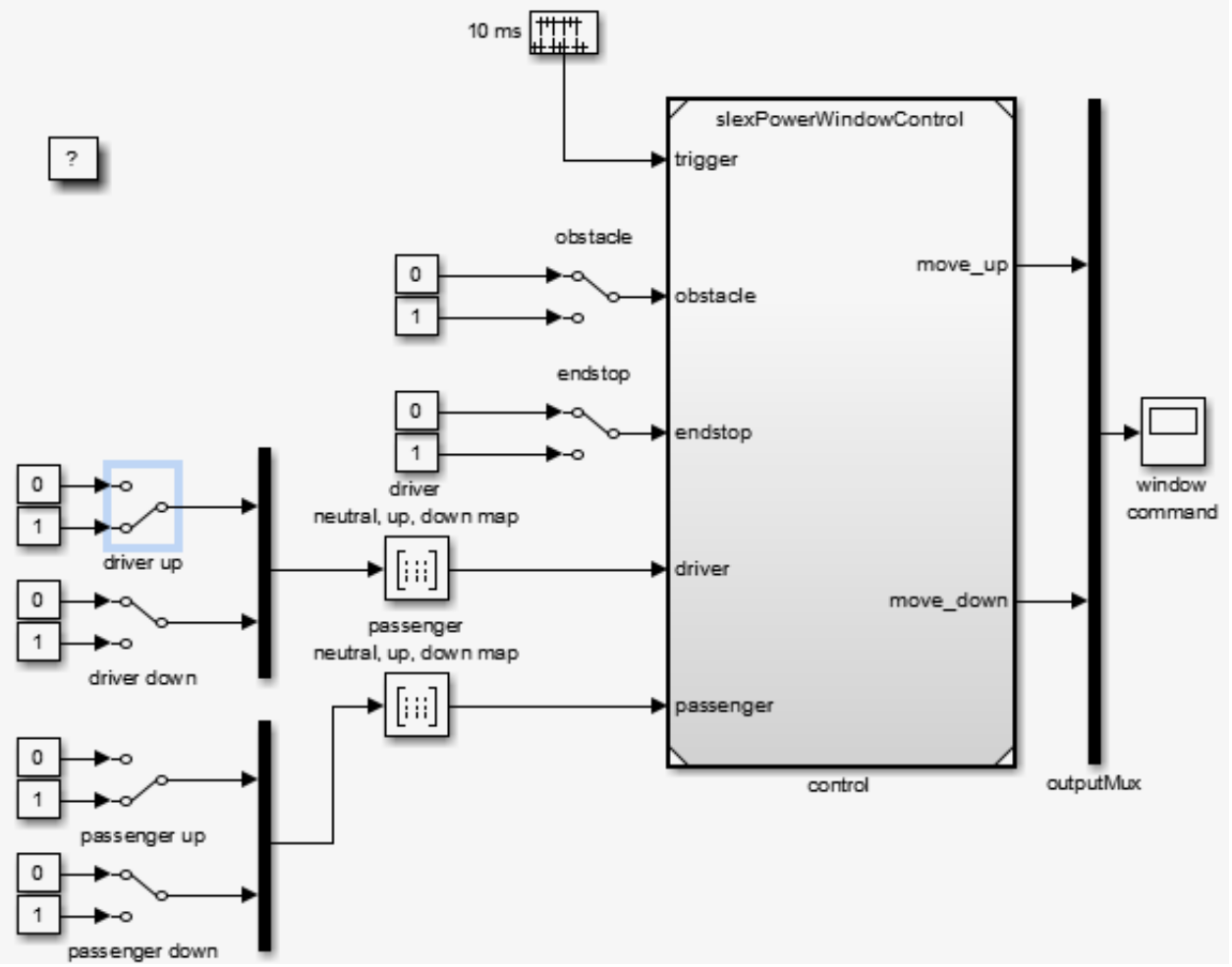


2. Double-click the driver down switch.

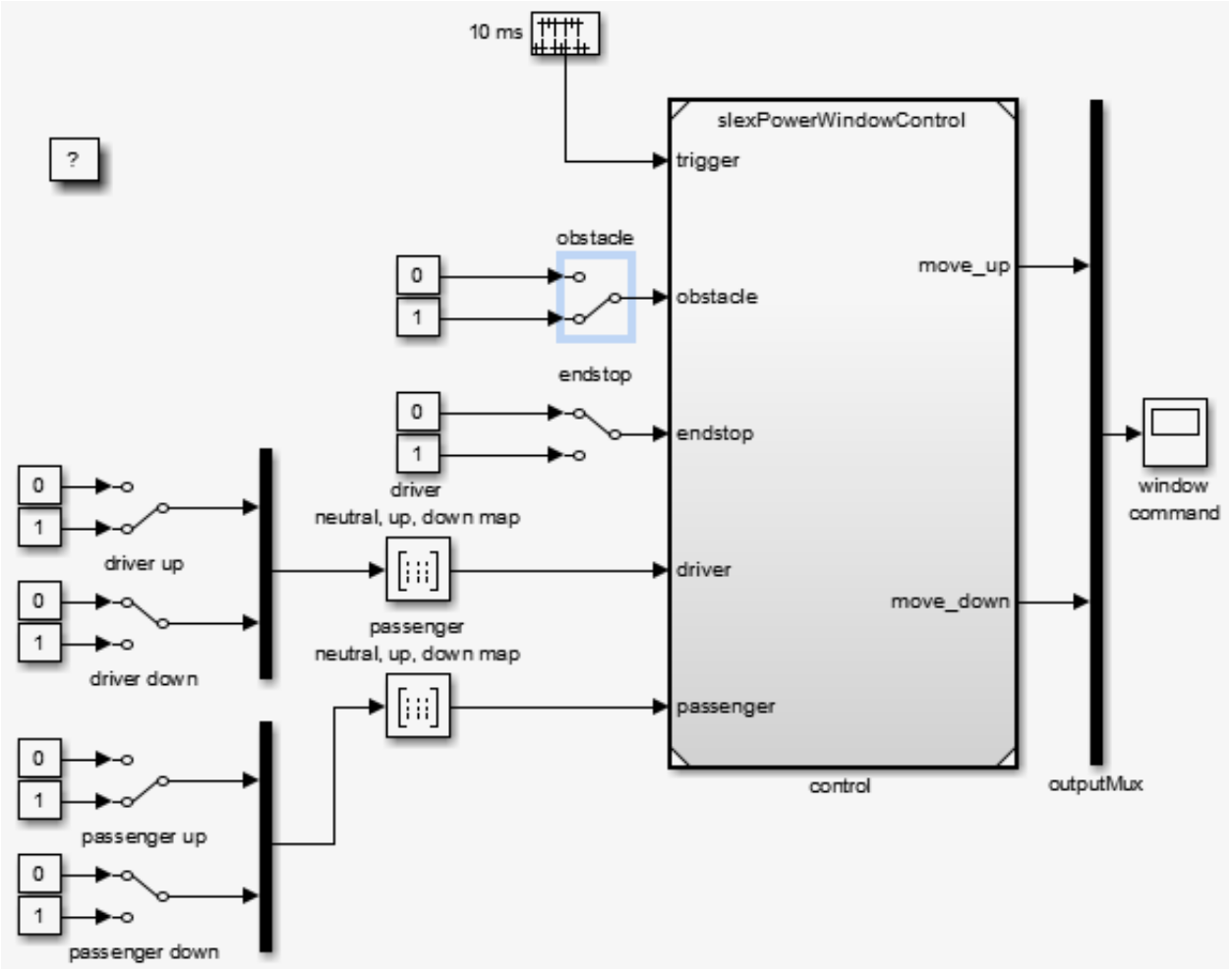


3. Simulate the model.
4. Notice how the state machine moves to the driver control part to generate the window down output instead of the window up output.
5. Double-click the driver control to driver up. Double-click the driver down switch.

   The driver window up state is reached, which generates the window up output again, i.e., `windowUp = 1`.



6. To observe state behavior when an object is between the window and the frame, double-click the obstacle switch.



7. Simulate the model.

   On the next sample time, the state machine moves to its `emergencyDown` state to lower the window a few inches. How far the software lowers the window depends on how long the state machine is in the `emergencyDown` state. This behavior is part of the next analysis phase.

   If a driver or passenger window switch is still active, the state machine moves into the up or down states upon the next sample time after leaving the emergency state. If the obstacle switch is also still active, the software again activates the emergency state at the next sample time.

Model Coverage

Validation of the Control Subsystem. Validate the discrete-event control of the window using the model coverage tool. This tool helps you determine the extent to which a model test case exercises the conditional branches of the controller. It helps evaluate whether all transitions in the discrete-event control are taken, given the test case, and whether all clauses in a condition that enables a particular transition have become true. Multiple clauses can enable one transition, e.g., the transition from emergency back to neutral occurs when either 100 ticks have occurred or if the endstop is reached.

To achieve full coverage, each clause evaluates to true and false for the test cases used. The percentage of transitions that a test case exercises is called its model coverage. Model coverage is a measure of how thoroughly a test exercises a model.

Using Simulink Verification and Validation software, you can apply the following test to the power window controller.

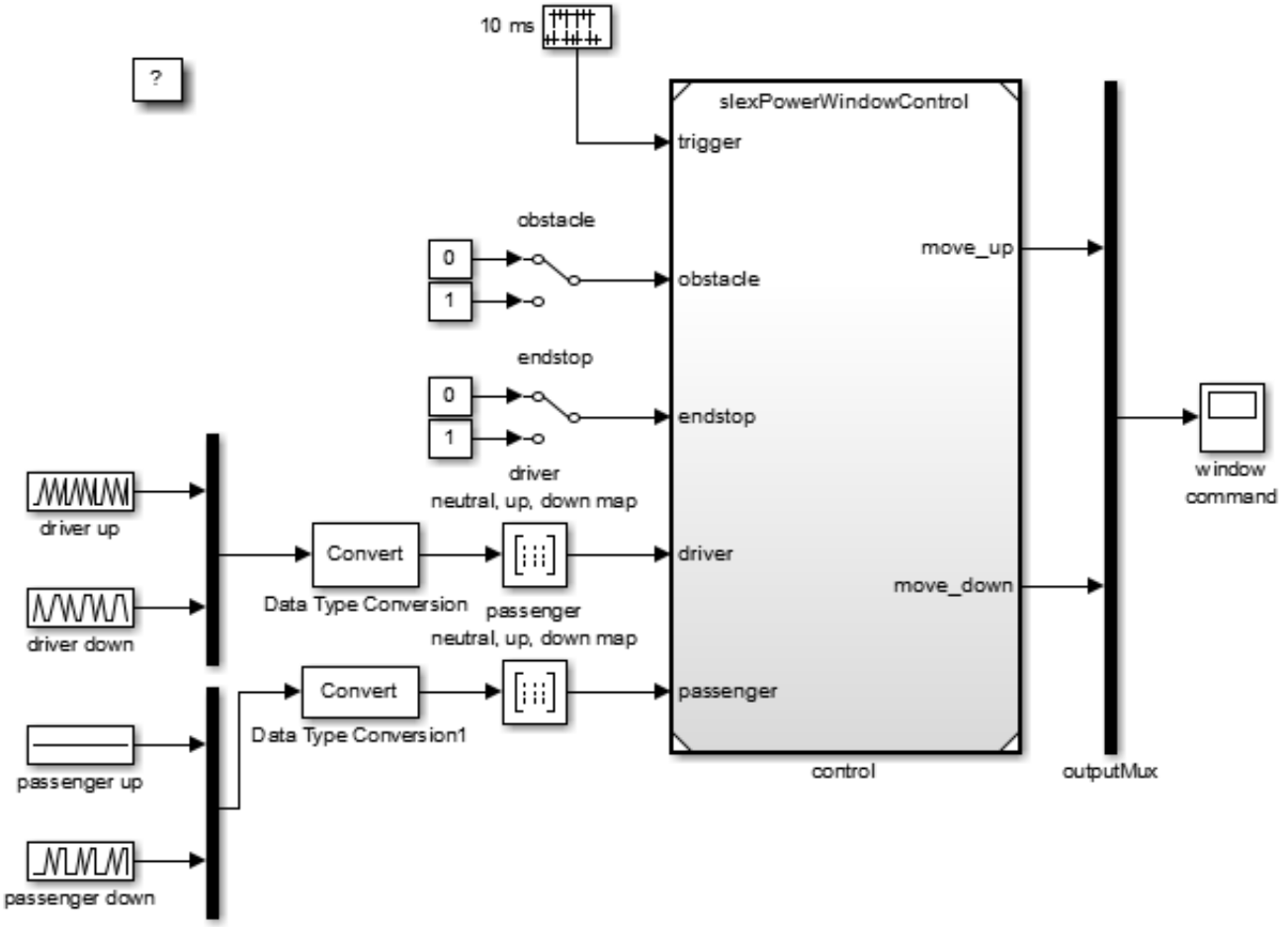
Position	**Step**						
	0	**1**	**2**	**3**	**4**	**5**	**6**
Passenger up	0	0	0	0	0	0	0
Passenger down	0	0	0	1	0	1	1

Driver up	0	0	1	0	1	0	1
Driver down	0	1	0	0	1	1	0

With this test, all switches are inactive at time 0. At regular 1 s steps, the state of one or more switches changes. For example, after 1 s, the driver down switch becomes active. To automatically run these input vectors, replace the manual switches by prescribed sequences of input. To see the preconstructed model:

1. In the MATLAB Command Window, type:

```
slexPowerWindowCntlCoverage
```



2. Simulate the model to generate the Simulink Verification and Validation coverage report.

For the `slexPowerWindowCntrlCoverage` model, the report reveals that this test handles 100% of the decision outcomes from the driver neutral, up, down map block. However, the test achieves only 50% coverage for the passenger neutral, up, down map block. This coverage means the overall coverage for `slexPowerWindowCntrlCoverage` is 45% while the overall coverage for the `slexPowerWindowControl` model is 42%. A few of the contributing factors for the coverage levels are:

- Passenger up block does not change.
- Endstop and obstacle blocks do not change.

Increase Model Coverage. To increase total coverage to 100%, you need to take into account all possible combinations of driver, passenger, obstacle, and endstop settings. When you are satisfied with the control behavior, you can create the power window system. For more information, see Create Model Using Model-Based Design.

This example increases the model coverage for the validation of the discrete-event control of the window. To start, the example uses inputs from `slexPowerWindowCntlCoverage` as a baseline for the model coverage. Next, to further exercise the discrete-event control of the window, it creates more input sets. The spreadsheet file, `inputCntlCoverageIncrease.xlsx`, contains these input sets using one input set per sheet.

In the example, the `slexPowerWindowSpreadsheetGeneration` utility function, which creates a spreadsheet template from the controller model, `slexPowerWindowControl`, creates the `inputCntlCoverageIncrease.xlsx`. In `inputCntlCoverageIncrease.xlsx`, the function uses the block names in the controller model as signal names. `slexPowerWindowSpreadsheetGeneration` defines the sheet names. The `slexWindowSpreadsheetAddInput` utility function populates `inputCntlCoverageIncrease.xlsx` with signal data.

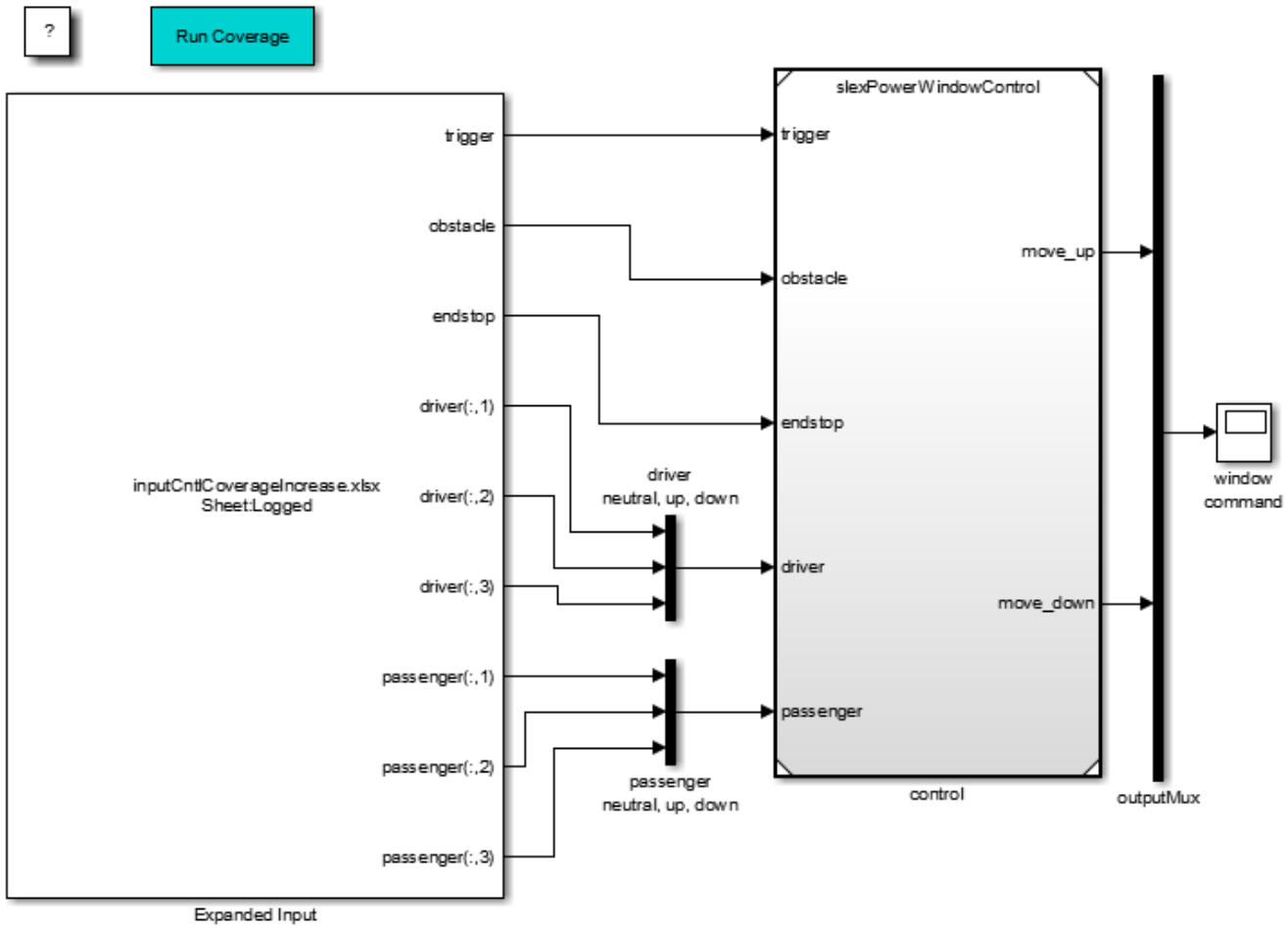
The sheet names of these input sets and their descriptions are:

Sheet Name	**Description**
`Logged`	Inputs logged from `slexPowerWindowCntlCoverage`
`LoggedObstacleOffEndStopOn`	Inputs logged from `slexPowerWindowCntlCoverage` with ability to hit endstop
`LoggedObstacleOnEndStopOff`	Inputs logged from `slexPowerWindowCntlCoverage` with obstacle in window
`LoggedObstacleOnEndStopOn`	Inputs logged from `slexPowerWindowCntlCoverage` with obstacle in window and ability to hit endstop
`DriverLoggedPassengerNeutral`	Inputs logged from `slexPowerWindowCntlCoverage` for driver only and passenger takes no action
`DriverDownPassengerNeutral`	Driver is putting down window and passenger takes no action
`DriverUpPassengerNeutral`	Driver is putting up window and passenger takes no action
`DriverAutoDownPassengerNeutral`	Driver is putting down window for one second (auto-down) and passenger takes no action
`DriverAutoUpPassengerNeutral`	Driver is putting up window for one second (auto-up) and passenger takes no action
`PassengerAutoDownDriverNeutral`	Passenger is putting down window for one second (auto-down) and driver takes no action
`PassengerAutoUpDriverNeutral`	Passenger is putting up window for one second (auto-up) and driver takes no action

To automatically run these input vectors, replace the inputs to the discrete-event control with the From Spreadsheet block using the file, `inputCntlCoverageIncrease.xlsx`. This file contains the multiple input sets. To see the preconstructed model:

1. In the MATLAB Command Window, type:

   ```
   slexPowerWindowCntlCoverageIncrease
   ```



2. To generate the Simulink Verification and Validation coverage report for multiple input set, double click the Run Coverage subsystem in the model.

   For the `slexPowerWindowCntrlCoverageIncrease` model, the report reveals that using multiple input sets has successfully raised the overall coverage for the `slexPowerWindowControl` model from 42% to 78%. Coverage levels are less than 100% because of missing input sets for:

   - Passenger up state
   - Driver up and down states
   - Passenger automatic down and automatic up states

Create Model Using Model-Based Design

Why Use Model-Based Design?

In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. Use Model-Based Design to:

- Use a common design environment across project teams.
- Link designs directly to requirements.
- Integrate testing with design to continuously identify and correct errors.
- Refine algorithms through multidomain simulation.
- Automatically generate embedded software code.
- Develop and reuse test suites.
- Automatically generate documentation for the model.
- Reuse designs to deploy systems across multiple processors and hardware targets.

For more information, see Model-Based Design.

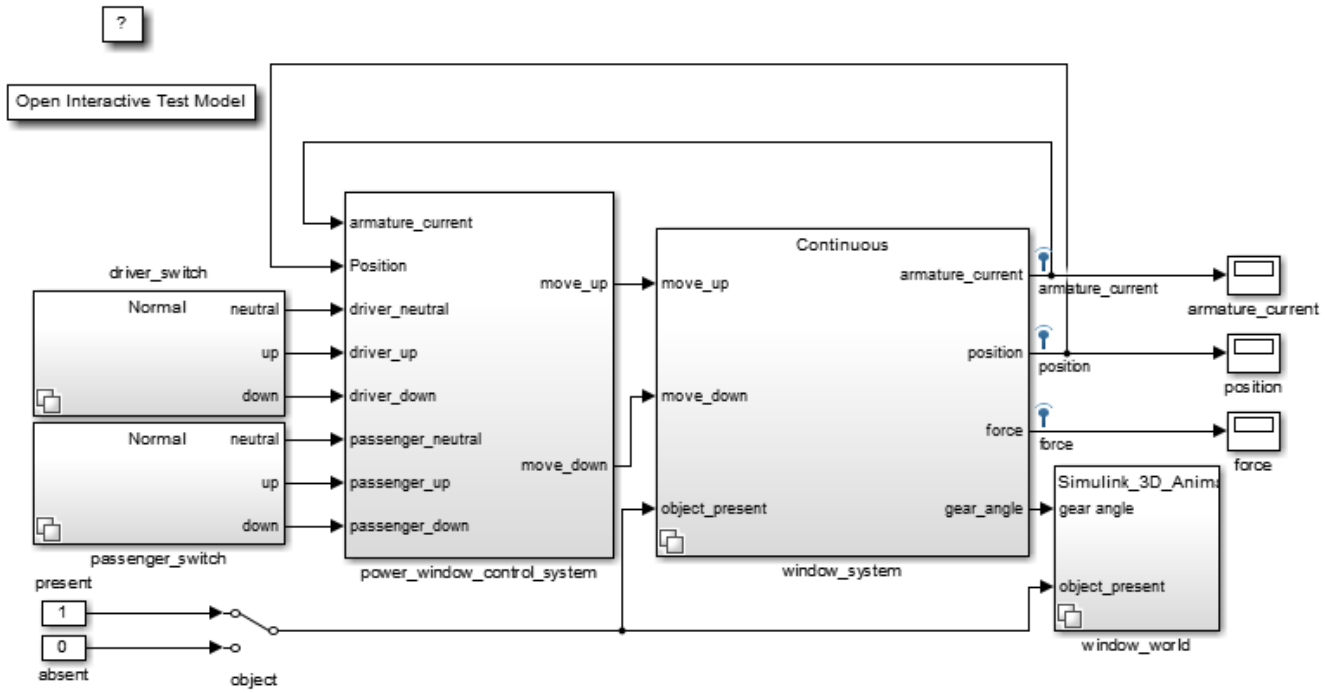
Implementation of Context Diagram: Power Window System

For requirements presented as a context diagram, see Context Diagram: Power Window System.

Create a Simulink model to resemble the context diagram.

1. Place the plant behavior into one subsystem.
2. Create two subsystems that contain the driver and passenger switches.
3. Add a control mechanism to conveniently switch between the presence and absence of the object.
4. Put the control in one subsystem.
5. Connect the new subsystems.
6. To see an implementation of this model, in the MATLAB Command Window, type:

```
slexPowerWindowStart
```



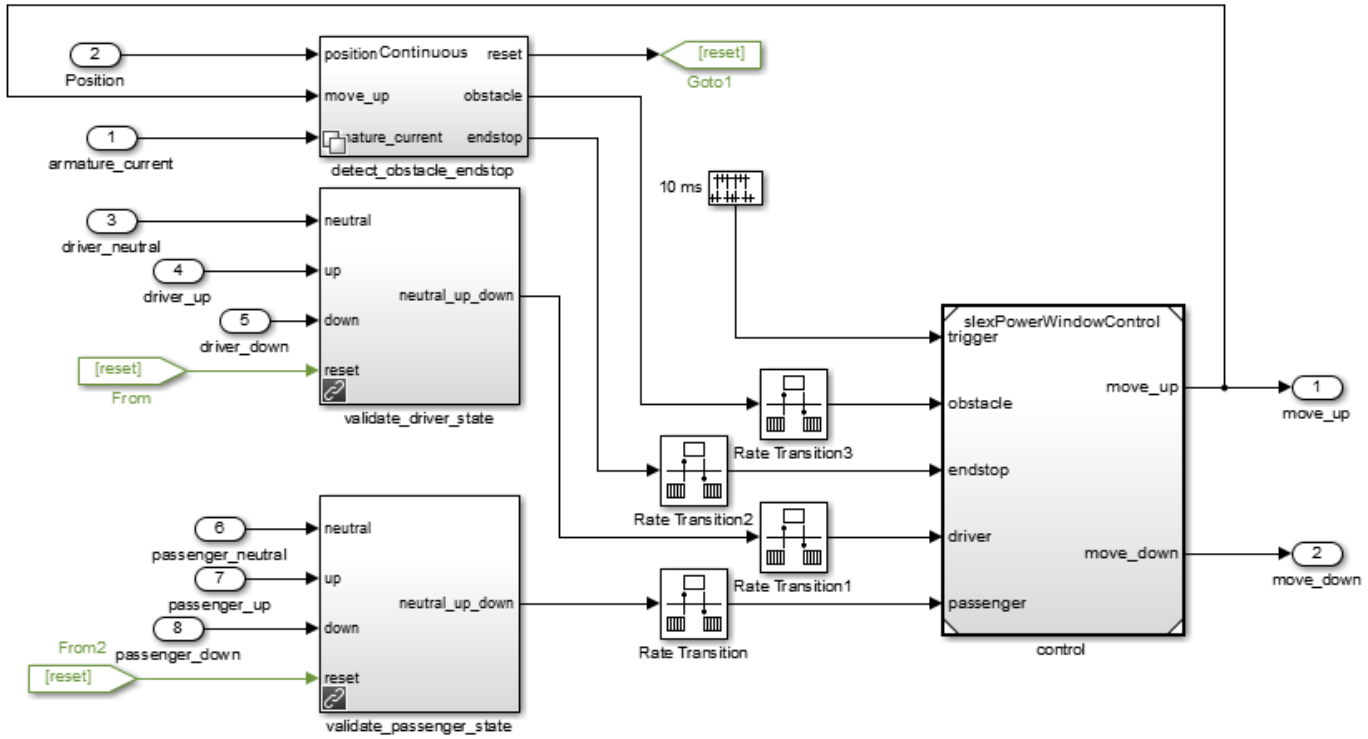
You can use the power window control activity diagram (Activity Diagram: Power Window Control) to decompose the power window controller of the context diagram into parts. This diagram shows the input and output signals present in the context diagram for easier tracing to their origins.

Implement Power Window Control System

To satisfy full requirements, the power window control must work with the validation of the driver and passenger inputs and detect the endstop.

For requirements presented as an activity diagram, see Activity Diagram: Power Window Control.

Double-click the slexPowerWindowExample/power_window_control_system block to open the following subsystem:



Implementation of Activity Diagram: Validate

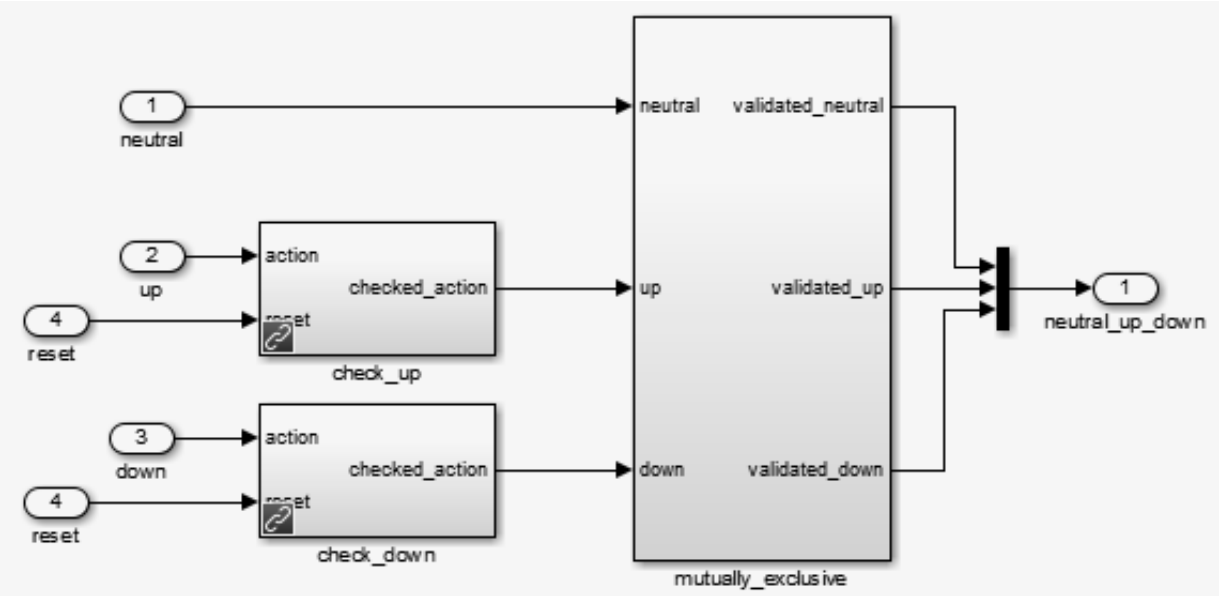
For requirements presented as activity diagrams, see Activity Diagram: Validate Driver and Activity Diagram: Validate Passenger.

The activity diagram adds data validation functionality for the driver and passenger commands to ensure correct operation. For example, when the window reaches the top, the software blocks the `up` command. The implementation decomposes each validation process in new subsystems. Consider the validation of the driver commands (validation of the passenger commands is similar). Check if the model can execute the `up` or `down` commands, according to the following:

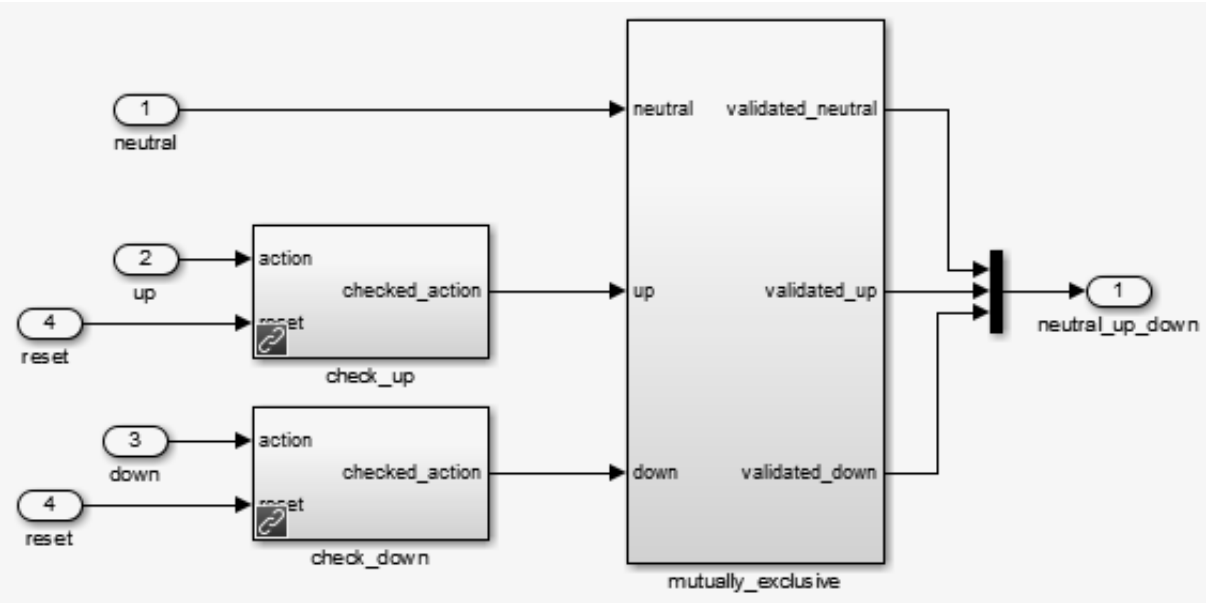
- The model allows the `down` command only when the window is not completely opened.
- The model allows the `up` command only when the window is not completely closed and no object is detected.

The third activity diagram process checks that the software sends only one of the three commands (`neutral`, `up`, `down`) to the controller. In an actual implementation, both `up` and `down` can be simultaneously true (for example, because of switch bouncing effects).

From the power_window_control_system subsystem, this is the validate_driver_state subsystem:



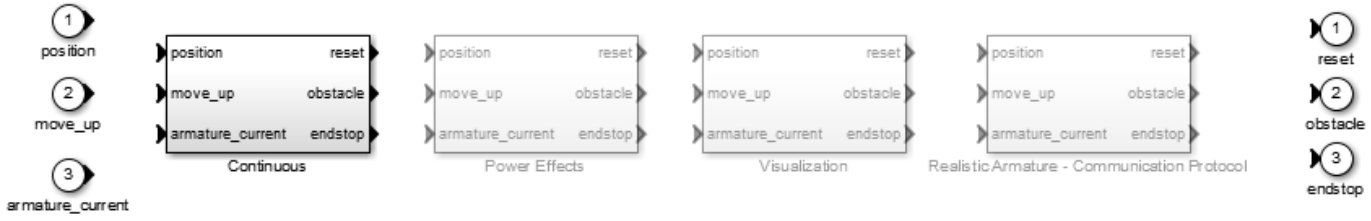
From the power_window_control_system subsystem, this is the validate_passenger_state subsystem:



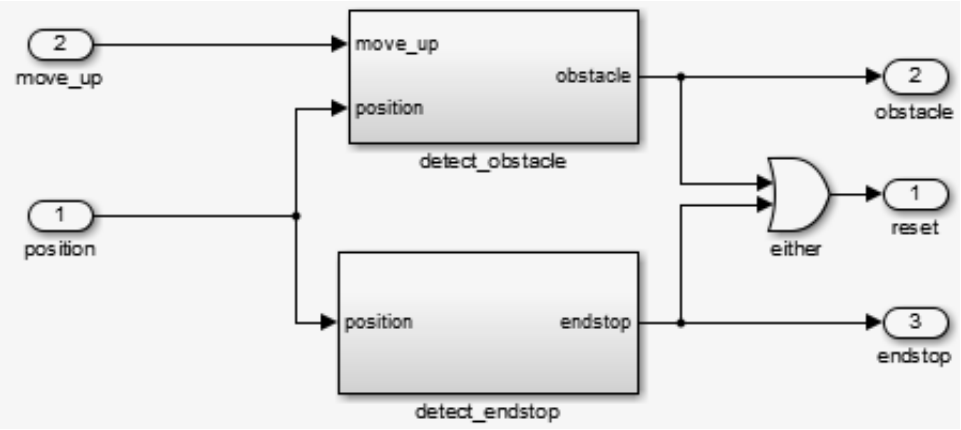
Implementation of Activity Diagram: Detect Obstacle Endstop

For requirements presented as an activity diagram, see Activity Diagram: Detect Obstacle Endstop.

In the `slexPowerWindowExample` model, the power_window_control_system/detect_obstacle_endstop block implements this activity diagram in the continuous variant of the Variant Subsystem block. During design iterations, you can add additional variants.



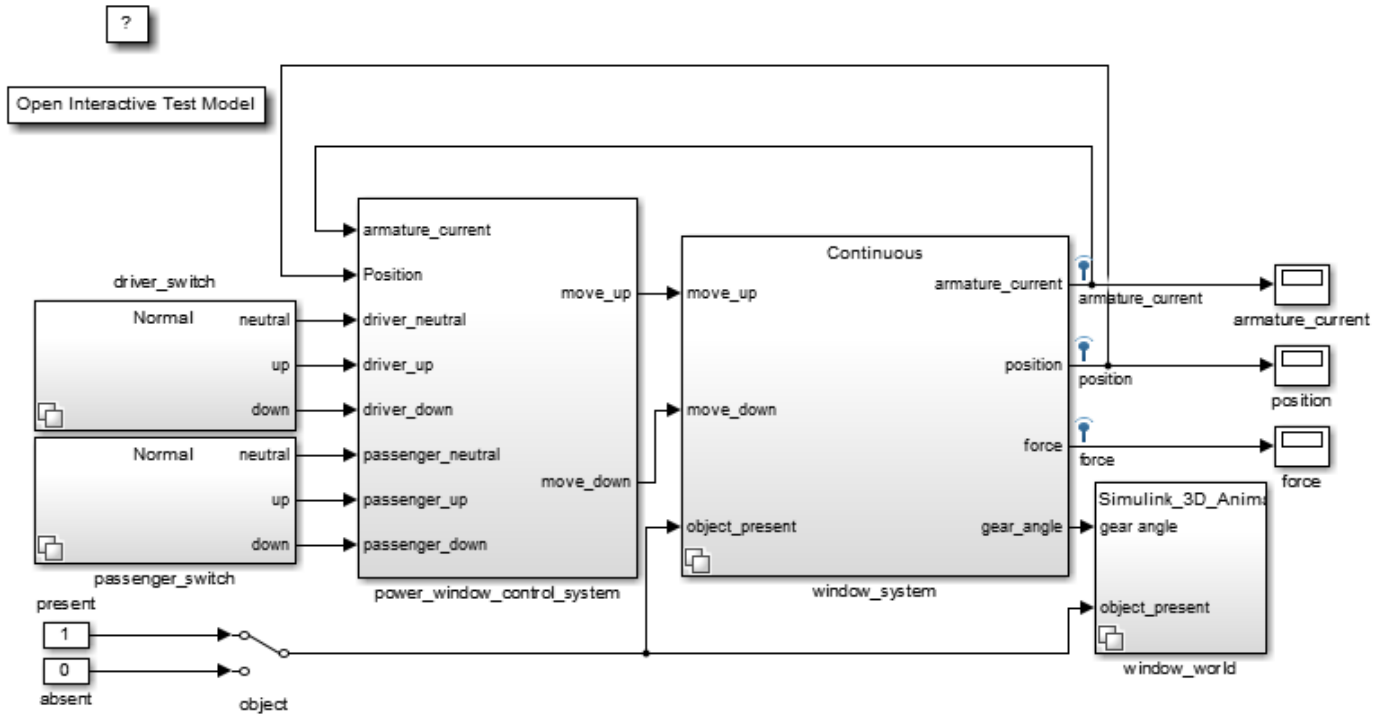
Double-click the `slexPowerWindowExample` model power_window_control_system/detect_obstacle_endstop/Continuous/verify_position block:



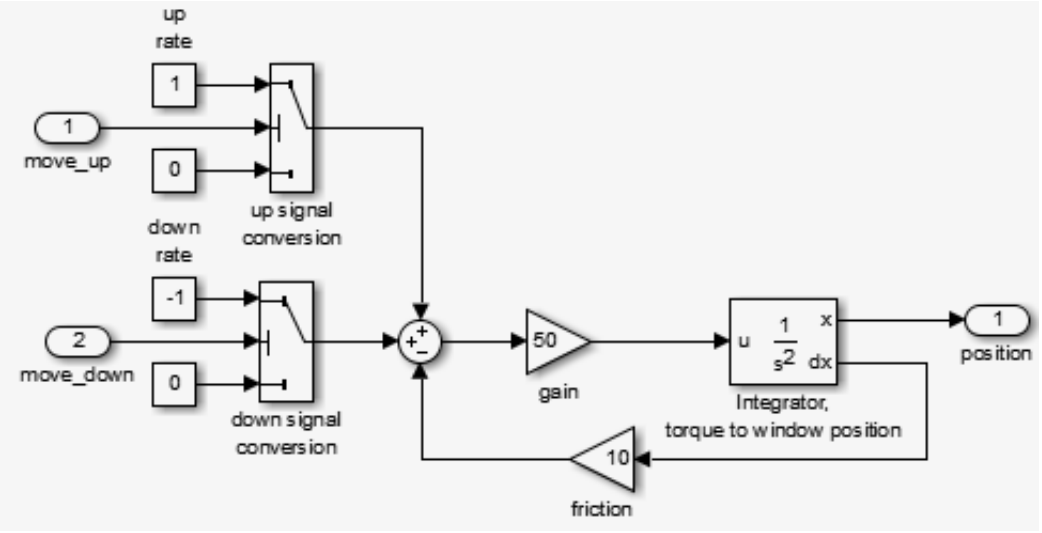
Hybrid Dynamic System: Combine Discrete-Event Control and Continuous Plant

After you have designed and verified the discrete-event control, integrate it with the continuous-time plant behavior. This step is the first iteration of the design with the simplest version of the plant.

In Simulink Project, navigate to **Files** and click **Project Files**. In the **configureModel** folder, run the `slexPowerWindowContinuous` utility to open and initialize the model.



The window_system block uses the Variant Subsystem block to allow for varying levels of fidelity in the plant modeling. Double-click the window_system/Continuous/2nd_order_window_system block to see the continuous variant.



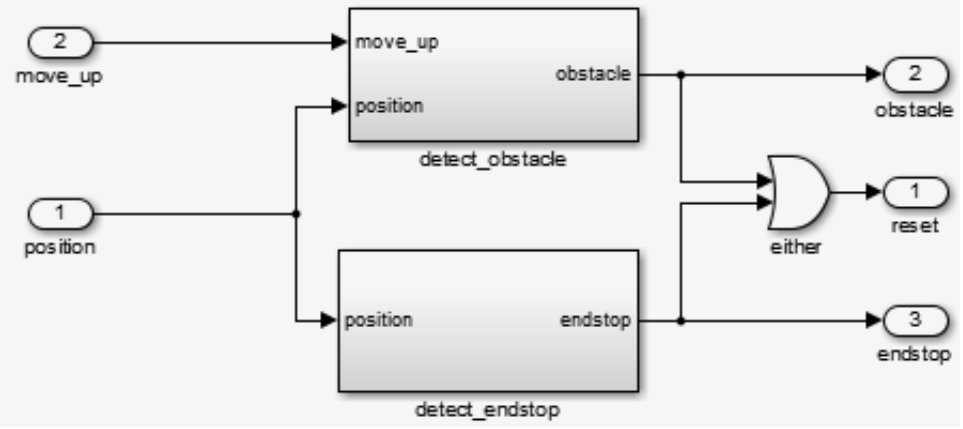
The plant is modeled as a second-order differential equation with step-wise changes in its input:

- When the Stateflow chart generates `windowUp`, the input is 1.
- When the Stateflow chart generates `windowDown`, the input is –1.
- Otherwise, the input is 0.

This phase allows analysis of the interaction between the discrete-event state behavior, its sample rate, and the continuous behavior of the window movement. There are threshold values to generate the window frame top and bottom:

- `endStop`
- Event when an obstacle is present, that is, `obstacle`
- Other events

Double-click the `slexPowerWindowExample` model power_window_control_system/detect_obstacle_endstop/Continuous/verify_position block to see the continuous variant.



When you run the `slexPowerWindowContinuous` configureModel utility, the model uses the continuous time solver ode23 (Bogacki-Shampine).

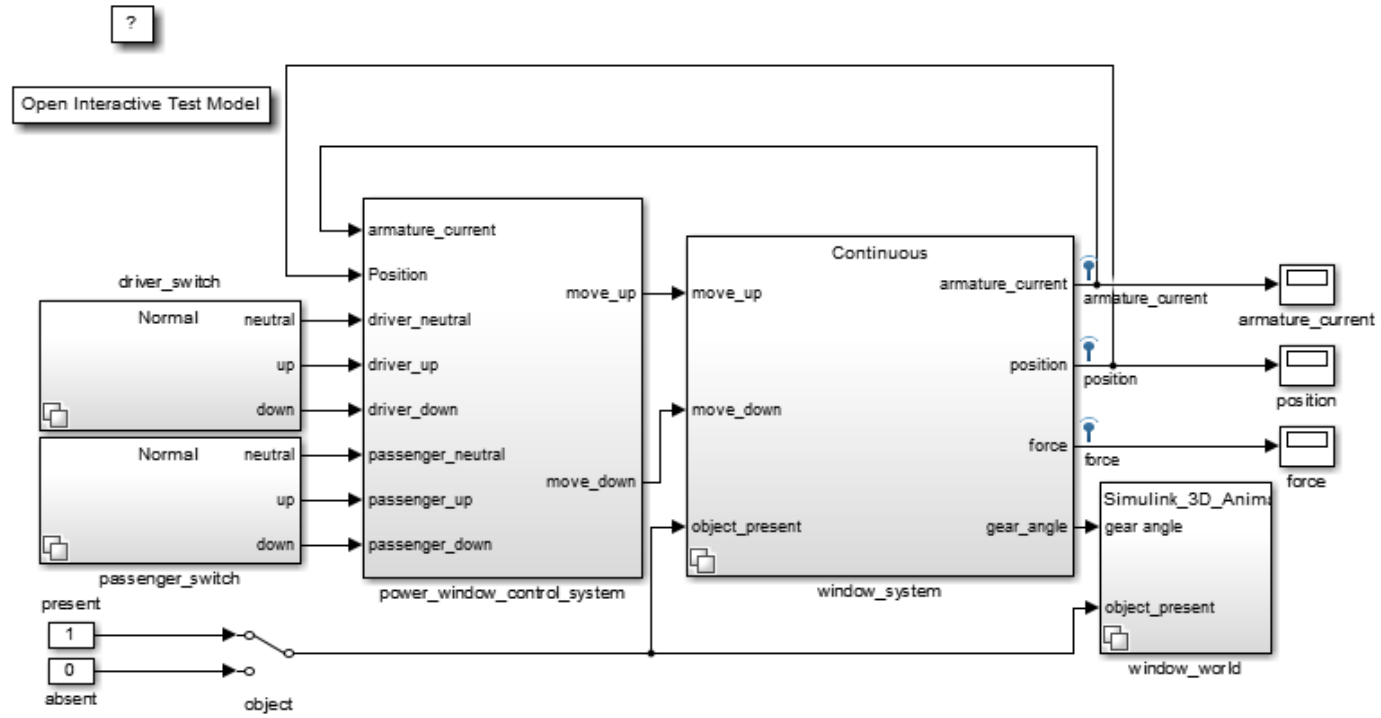
A structure analysis of a system results in:

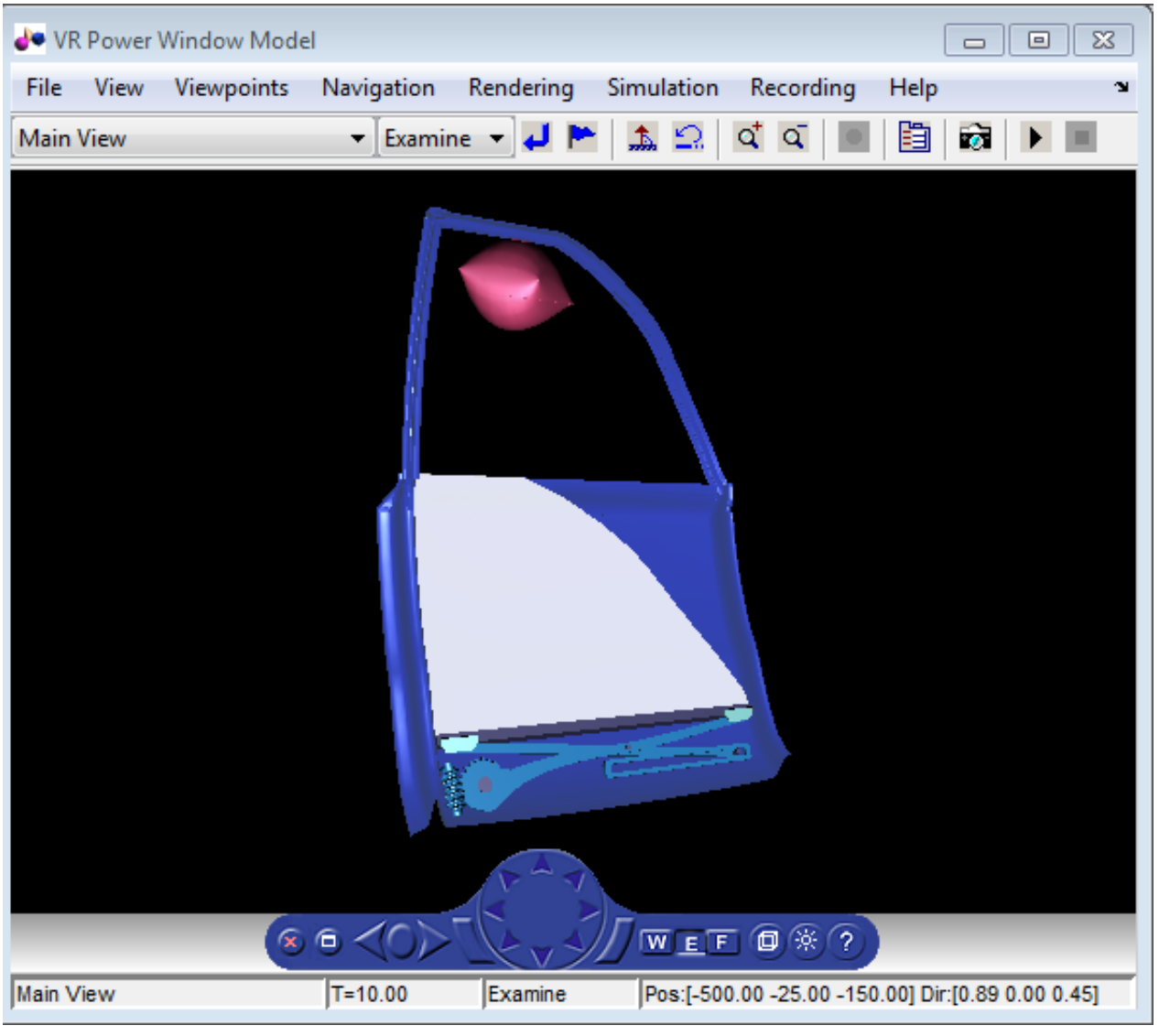
- A functional decomposition of the system
- Data definitions with the specifics of the system signals
- Timing constraints

A structure analysis can also include the implementation architecture (outside the scope of this discussion).

The implementation also adds a control mechanism to conveniently switch between the presence and absence of the object.

Expected Controller Response. To view the window movement, in Simulink Project in the **Shortcut Management** section, right-click `SimHybridPlantLowOrder`, and select **Run**. Alternatively, you can run the task `slexPowerWindowContinuousSim`.





The position scope shows the expected result from the controller. After 30 cm, the model generates the `obstacle` event and the Stateflow chart moves into its `emergencyDown` state. In this state, `windowDown` is output until the window lowers by about 10 cm. Because the passenger window up switch is still on, the window starts moving up again and this process repeats. Stop the simulation and open the position scope to observe the oscillating process. In case of an emergency, the discrete-event control rolls down the window approximately 10 cm.

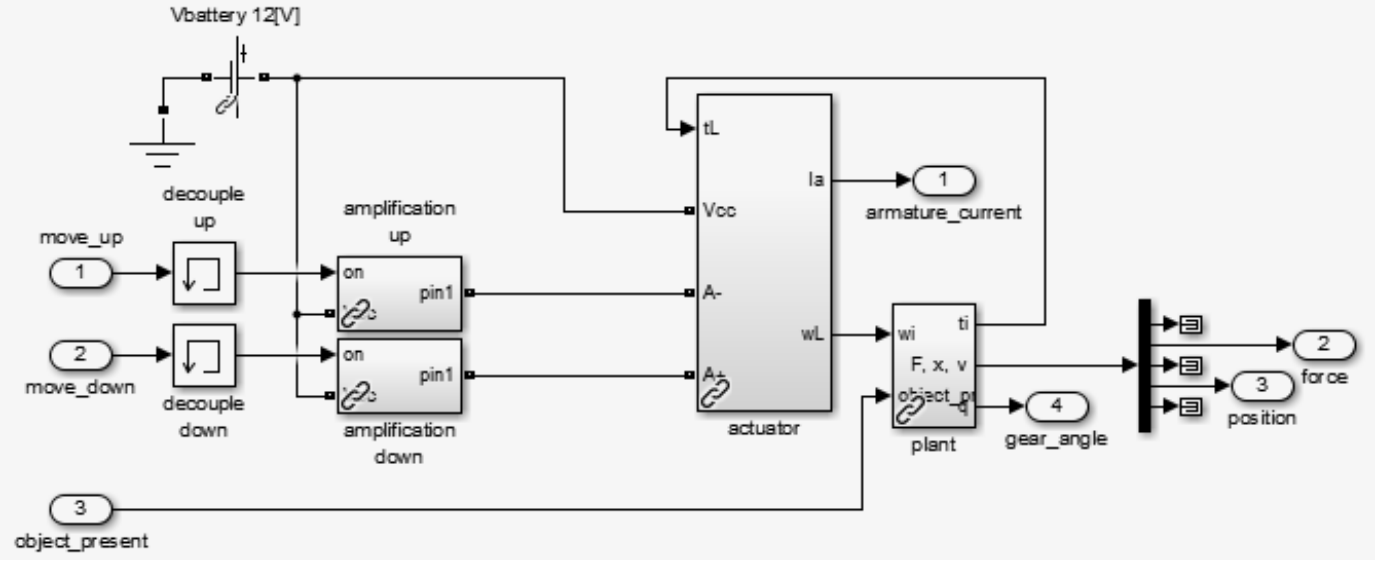
Detailed Modeling of Power Effects

After an initial analysis of the discrete-event control and continuous dynamics, you can use a detailed plant model to evaluate performance in a more realistic situation. It is best to design models at such a level of detail in the power domain, in other words, as energy flows. Several domain-specific MathWorks blocksets can help with this.

To take into account energy flows, add a more detailed variant consisting of power electronics and a multibody system to the window_system variant subsystem.

To open the model and explore the more detailed plant variant, in Simulink Project, run configureModel `slexPowerWindowPowerEffects`.

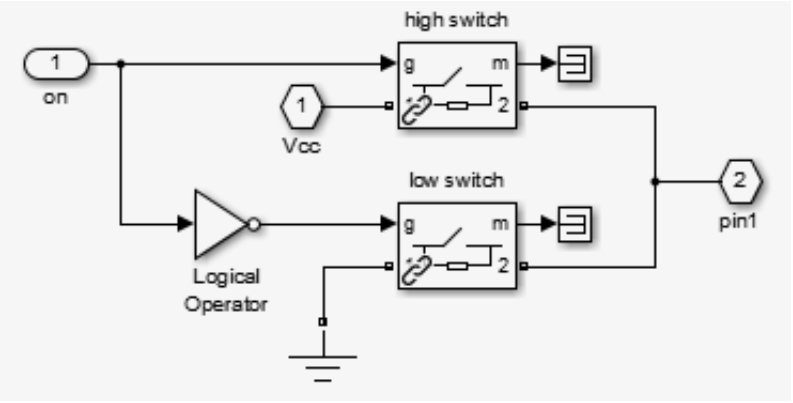
Double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system block.



Power Electronics Subsystem. The model must amplify the control signals generated by the discrete-event controller to be powerful enough to drive the DC motor that moves the window.

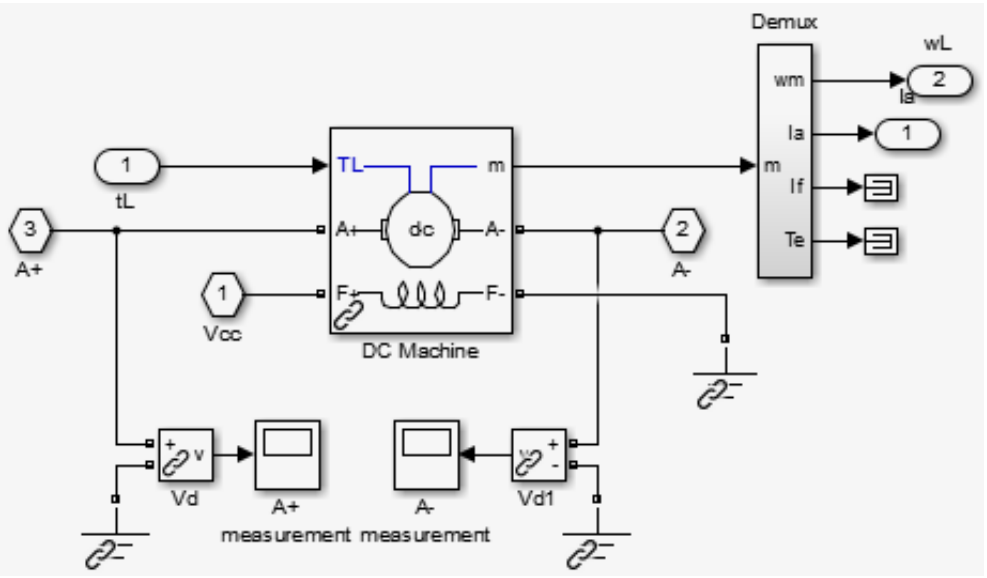
The amplification modules model this behavior. They show that a switch either connects the DC motor to the battery voltage or ground. By connecting the battery in reverse, the system generates a negative voltage and the window can move up, down, or remain still. The window is always driven at maximum power. In other words, no DC motor controller applies a prescribed velocity.

To see the implementation, double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system/amplification_up block.

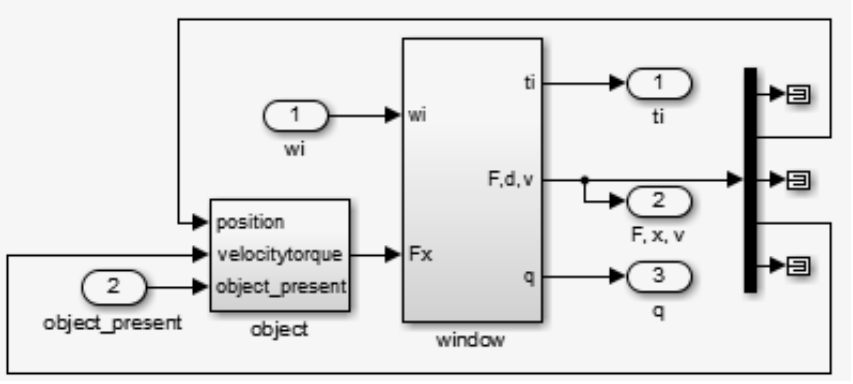


Multibody System. This implementation models the window using Simscape Multibody multibody blocks.

To see the actuator implementation, double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system/actuator block.



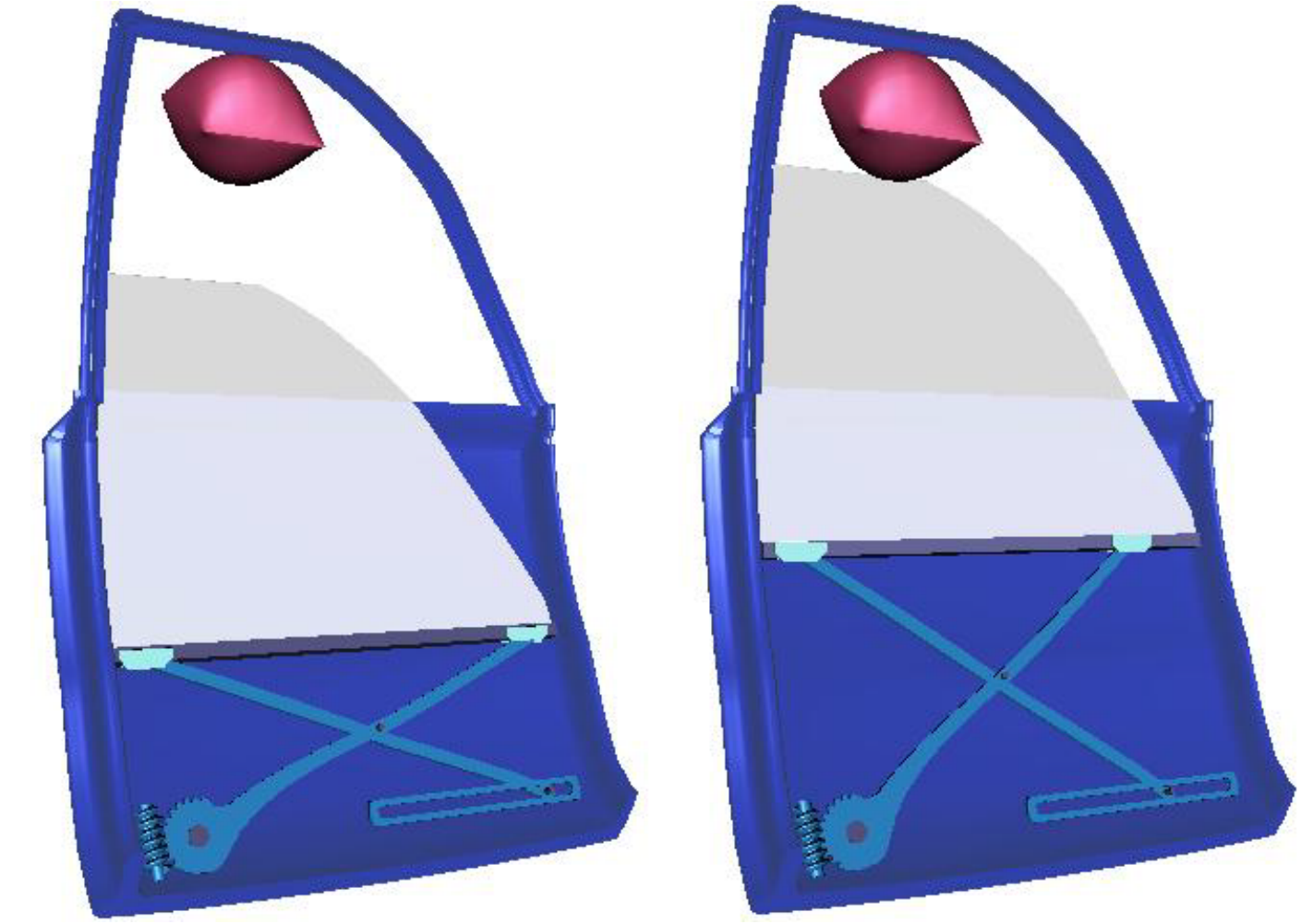
To see the window implementation, double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system/plant block.



This implementation uses Simscape Multibody multibody blocks for bodies, joints, and actuators. The window model consists of:

- A worm gear
- A lever to move the window holder in the vertical direction

The figure shows how the mechanical parts move.



Iterate on the Design. An important effect of the more detailed implementation is that there is no window position measurement available. Instead, the model measures the DC motor current and uses it to detect the endstops and to see if an obstacle is present. The next stage of the system design analyzes the control to make sure that it does not cause excessive force when an obstacle is present.

In the original system, the design removes the obstacle and endstop detection based on the window position and replaces it with a current-based implementation. It also connects the process to the controller and position and force measurements. To reflect the different signals used, you must modify the data definition. In addition, observe that, because of power effects, the units are now amps.

```
PSPEC 1.3.1: DETECT ENDSTOP
  ENDSTOP = ARMATURE_CURRENT > ENDSTOP_MAX

PSPEC 1.3.2: DETECT OBSTACLE
  OBSTACLE = (ARMATURE_CURRENT > OBSTACLE_MAX) and MOVE_UP for 500 ms

PSPEC 1.3.3: ABSOLUTE VALUE
  ABSOLUTE_ARMATURE_CURRENT = abs(ARMATURE_CURRENT)
```

This table lists the additional signal for the Context Diagram: Power Window System data definitions.

Context Diagram: Power Window System Data Definition Changes

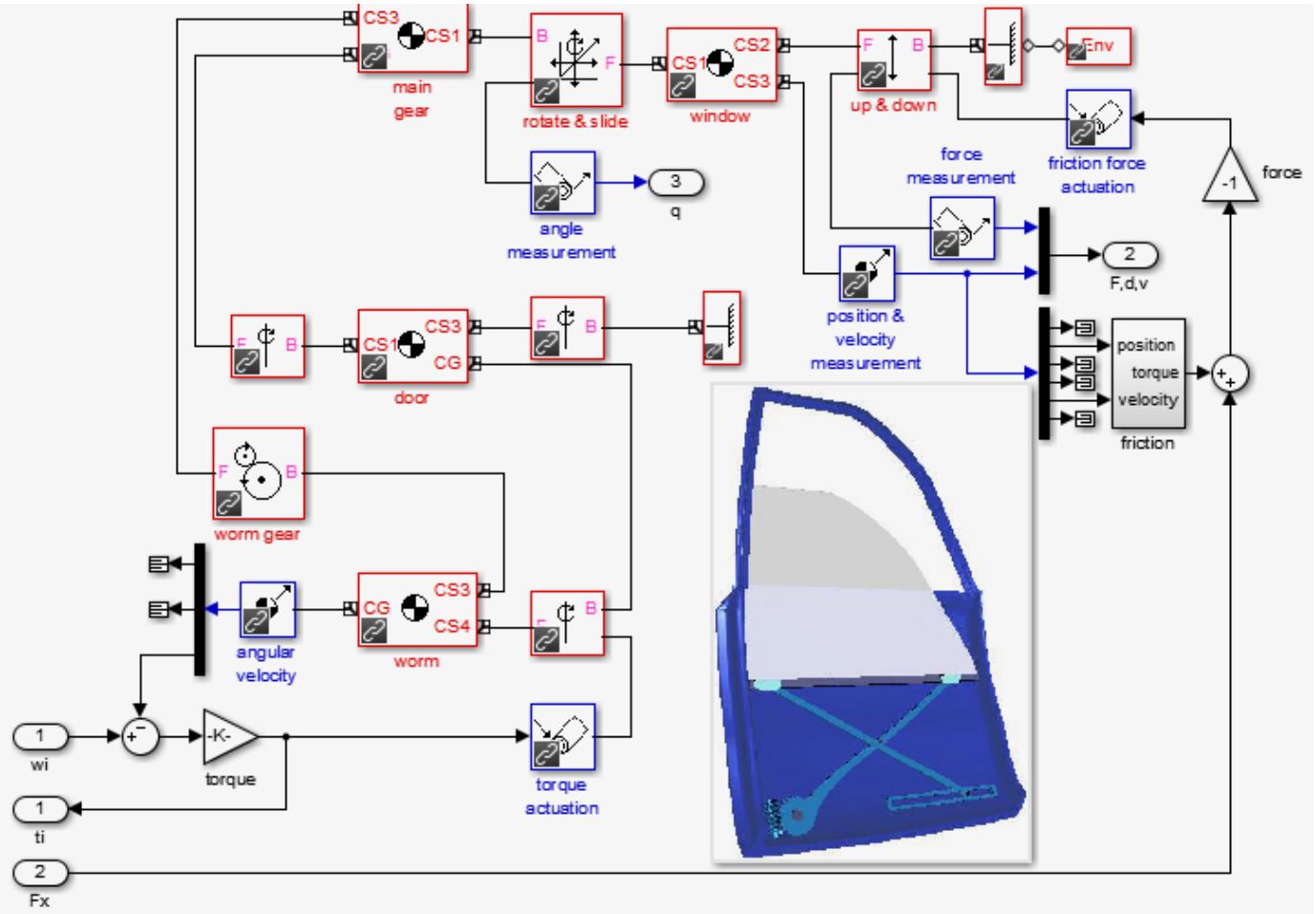
Signal	Information Type	Continuous/ Discrete	Data Type	Values
ARMATURE_CURRENT	Data	Continuous	Real	–20 to 20 A

This table lists the changed signals for the Activity Diagram: Detect Obstacle Endstop data definitions.

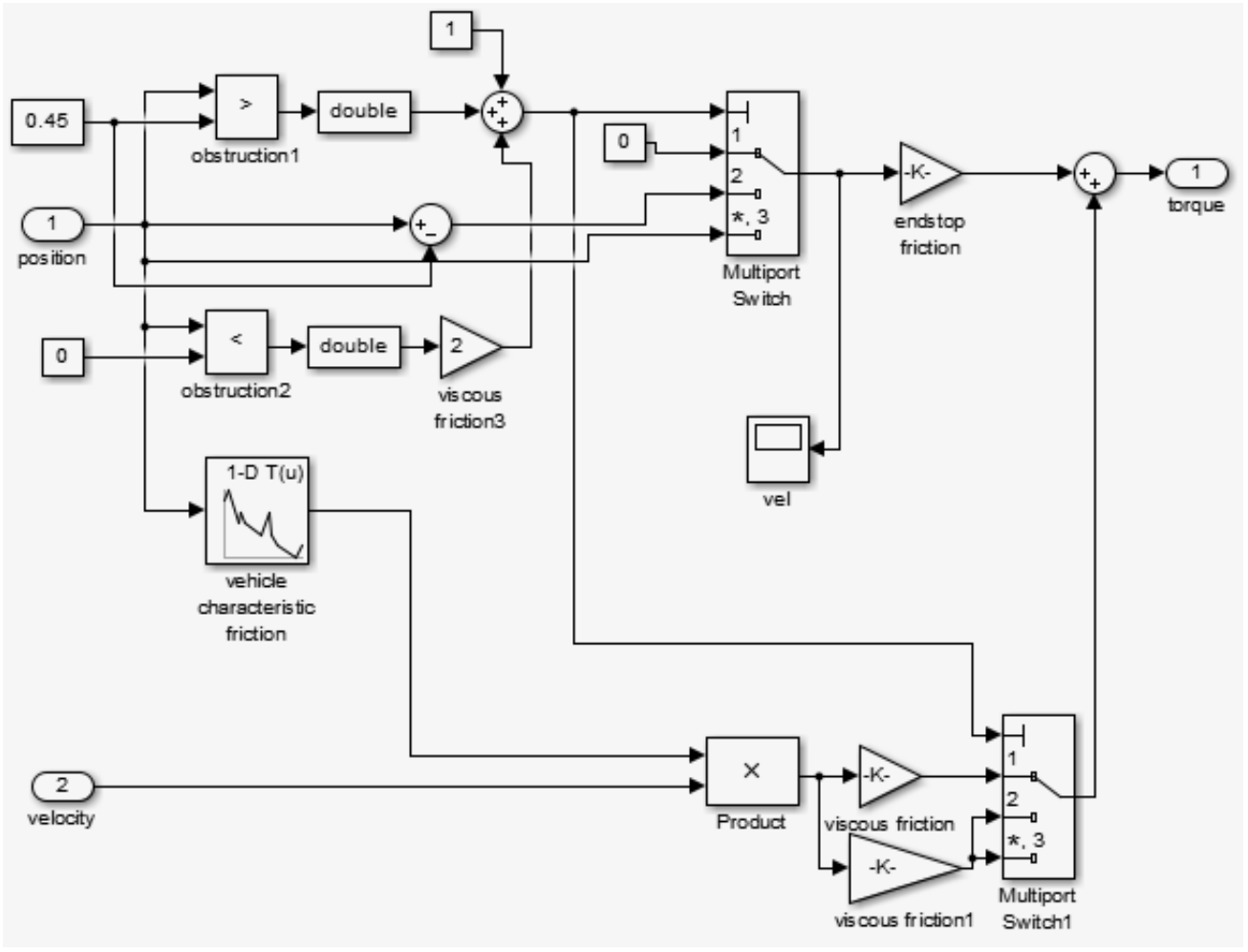
Activity Diagram: Detect Obstacle Endstop Data Definition Changes

Signal	Information Type	Continuous/ Constant	Data Type	Values
ABSOLUTE_ARMATURE_CURRENT	Data	Continuous	Real	0 to 20 A
ENDSTOP_MAX	Data	Constant	Real	15 A
OBSTACLE_MAX	Data	Constant	Real	2.5 A

To see the window subsystem, double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system/plant/window block.



The implementation uses a lookup table and adds noise to allow evaluation of the control robustness. To see the implementation of the friction subsystem, double-click the `slexPowerWindowExample` model window_system/Power Effects - Visualization/detailed_window_system/plant/window/friction block.



Control Law Evaluation

The idealized continuous plant allows access to the window position for `endStop` and `obstacle` event generation. In the more realistic implementation, the model must generate these events from accessible physical variables. For power window systems, this physical variable is typically the armature current, `Ia`, of the DC motor that drives the worm gear.

When the window is moving, this current has an approximate value of 2 A. When you switch the window on, the model draws a transient current that can reach a value of approximately 10 A. When the current exceeds 15 A, the model activates endstop detection. The model draws this current when the angular velocity of the motor is kept at almost 0 despite a positive or negative input voltage.

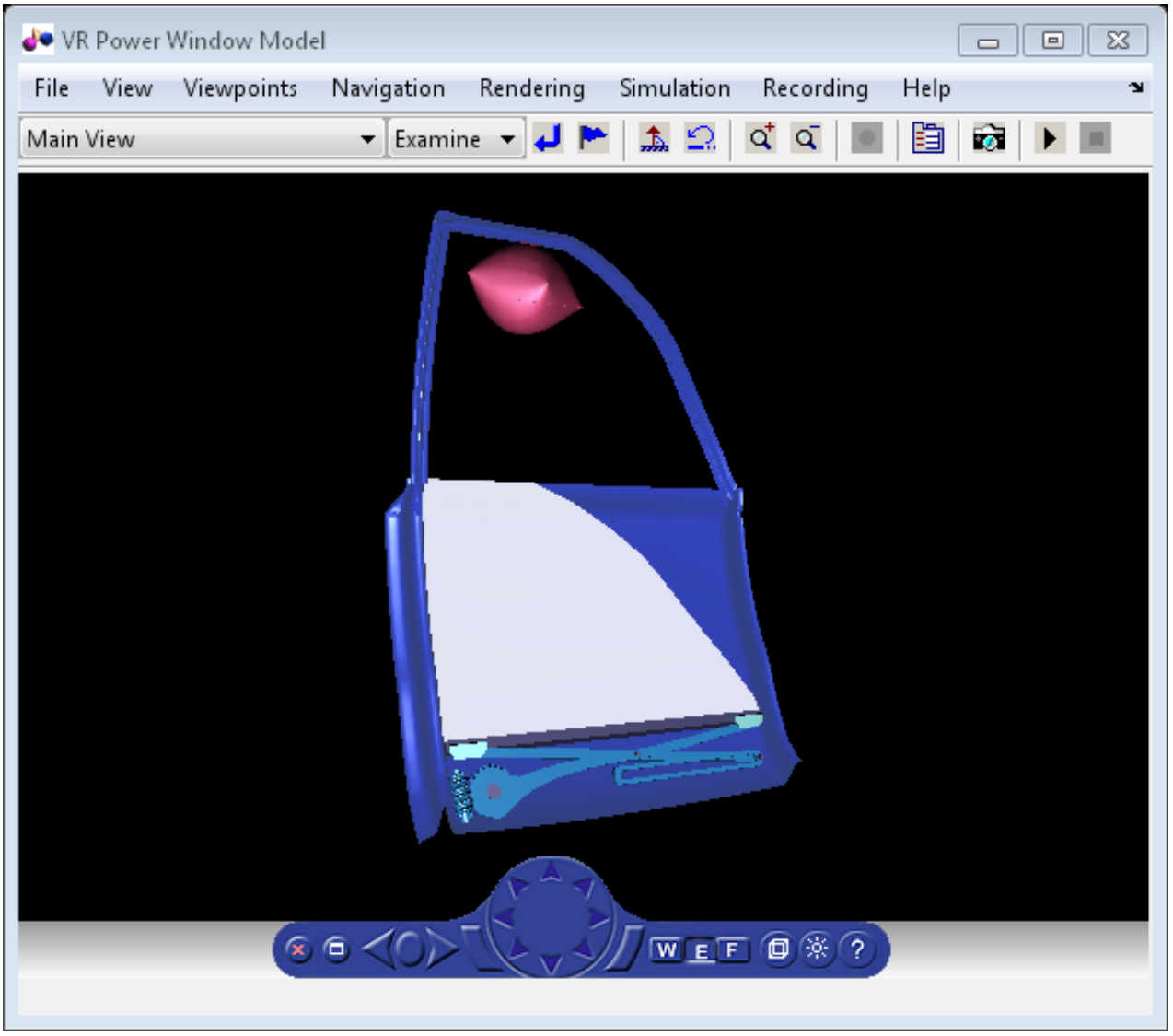
Detecting the presence of an object is more difficult in this setup. Because safety concerns restrict the window force to no more than 100 N, an armature current much less than 10 A should detect an object. However, this behavior conflicts with the transient values achieved during normal operation.

Implement a control law that disables object detection during achieved transient values. Now, when the system detects an armature current more than 2 A, it considers an object to be present and enters the `emergencyDown` state of the discrete-event control. Open the force scope window (measurements are in newtons) to check that the force exerted remains less than 100 N when an object is present and the window reverses its velocity.

In reality, far more sophisticated control laws are possible and implemented. For example, you can implement neural-network-based learning feedforward control techniques to emulate the friction characteristic of each individual vehicle and changes over time.

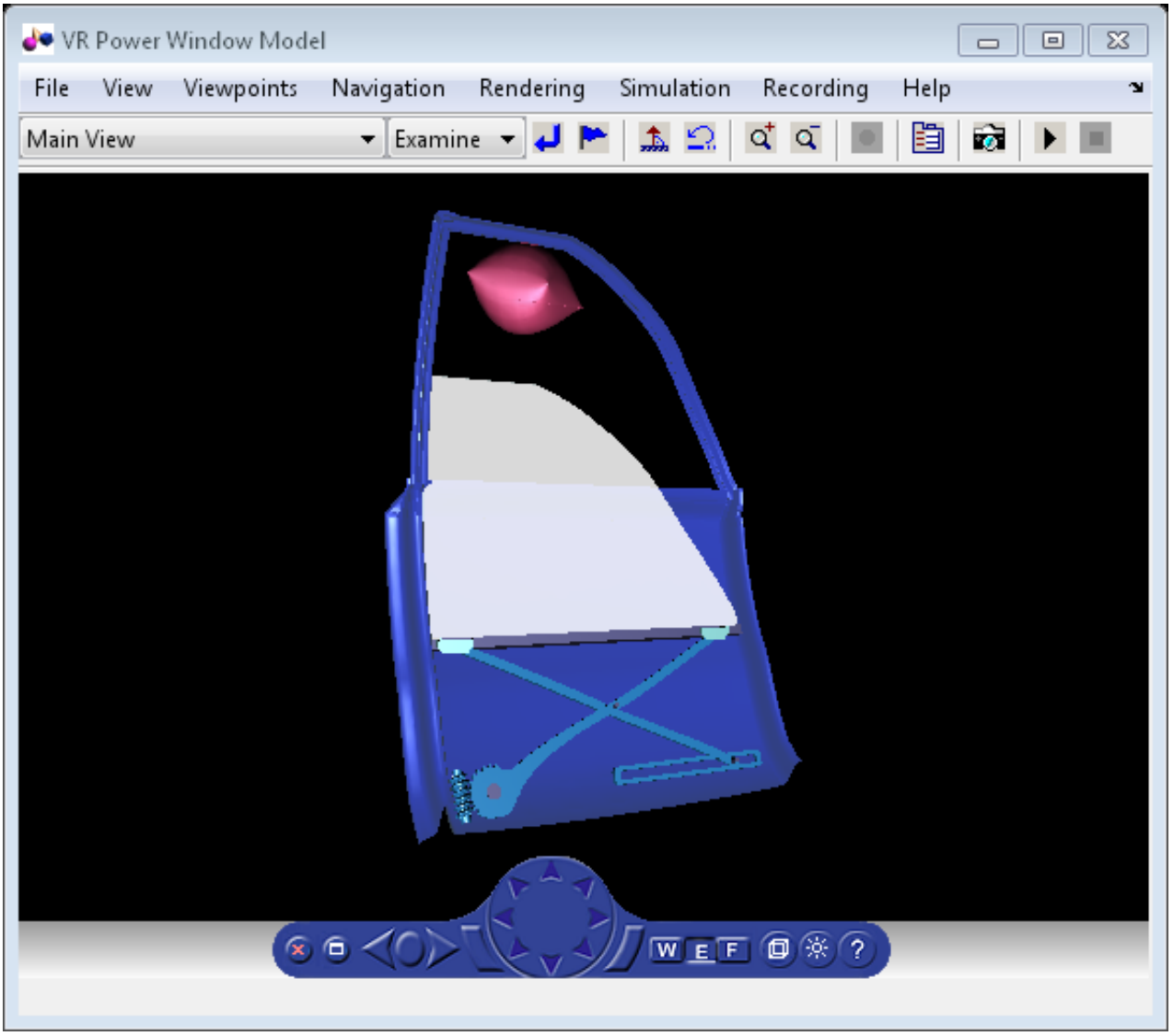
Visualization of the System in Motion

If you have Simulink 3D Animation software installed, you can view the geometrics of the system in motion via a virtual reality world. If the VR Sink block is not yet open, in the `slexPowerWindowExample/window_world/Simulink_3D_Animation View` model, double-click the VR Sink block.

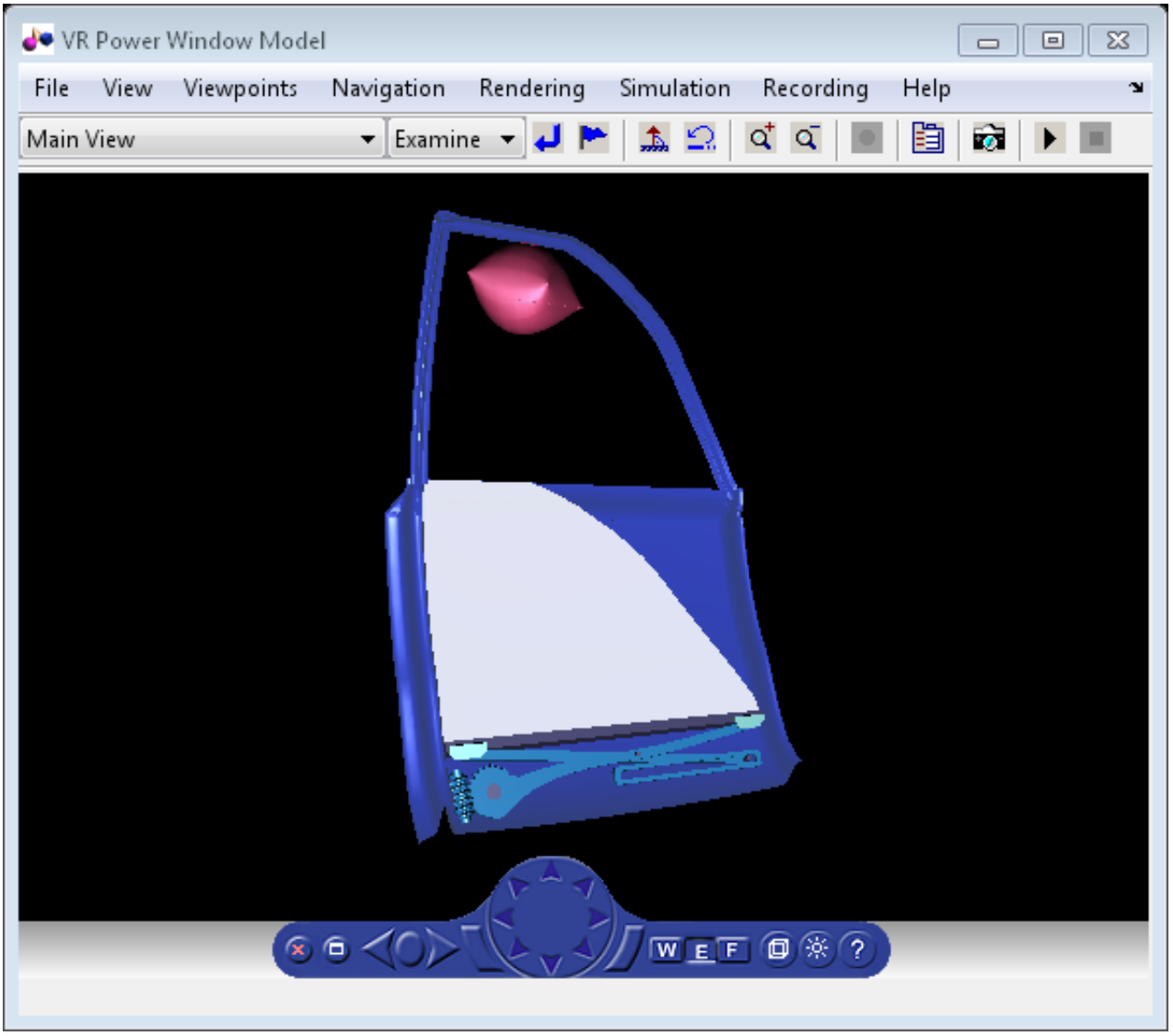


To simulate the model with a stiff solver:

1. In Simulink Project, run the task, `slexPowerWindowPowerEffectsSim`. This batch job sets the solver to ode23tb (stiff/TR-BDF2).
2. In the `slexPowerWindowExample` model passenger_switch/Normal block, set the passenger up switch to on.
3. In the `slexPowerWindowExample` model driver_switch/Normal block, set the driver up switch to off.
4. Simulate the model.
5. Between 10 ms and 1 s in simulation time, switch off the slexPowerWindowExample/passenger_switch/Normal block passenger up switch to initiate the auto-up behavior.



6. Observe how the window holder starts to move vertically to close the window. When the model encounters the object, it rolls the window down.
7. Double-click the `slexPowerWindowExample` model passenger_switch/Normal block driver down switch to roll down the window completely and then simulate the model. In this block, at less than one second simulation time, switch off the driver down switch to initiate the auto-down behavior.



8. When the window reaches the bottom of the frame, stop the simulation.
9. Look at the position measurement (in meters) and at the armature current (Ia) measurement (in amps).

> **Note:** The absolute value of the armature current transient during normal behavior does not exceed 10 A. The model detects the obstacle when the absolute value of the armature current required to move the window up exceeds 2.5 A (in fact, it is less than –2.5 A). During normal operation, this is about 2 A. You might have to zoom into the scope to see this measurement. The model detects the window endstop when the absolute value of the armature current exceeds 15 A.

Variation in the armature current during normal operation is due to friction that is included by sensing joint velocities and positions and applying window specific coefficients.

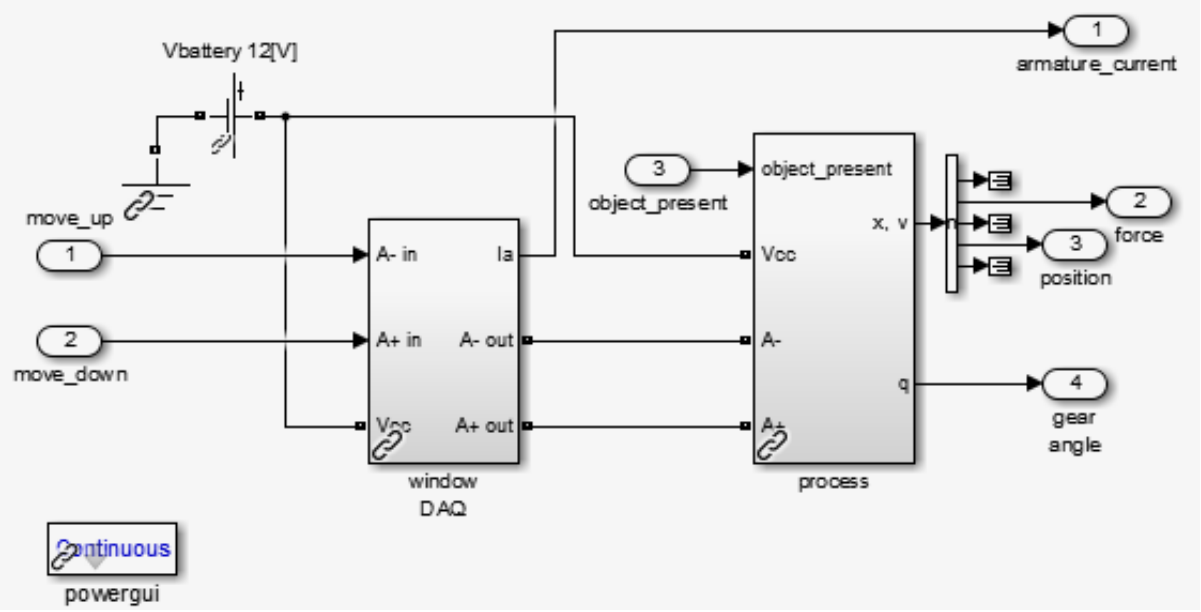
Realistic Armature Measurement

The armature current as used in the power window control is an ideal value that is accessible because of the use of an actuator model. In a more realistic situation, data acquisition components must measure this current value.

To include data acquisition components, add the more realistic measurement variant to the window_system variant subsystem. This realistic measurement variant contains a signal conditioning block in which the current is derived based on a voltage measurement.

To open a model and configure the realistic measurement, in Simulink Project, run the configureModel task `slexPowerWindowRealisticArmature`.

To view the contents of the Realistic Armature - Communications Protocol block, double-click the `SlexPowerWindowExample` model window_system/Realistic Armature - Communications Protocol/detailed_window_system_with_DAQ.



The measurement voltage is within the range of an analog-to-digital converter (ADC) that discretizes based on a given number of bits. You must scale the resulting value based on the value of the resistor and the range of the ADC.

Include these operations as fixed-point computations. To achieve the necessary resolution with the given range, 16 bits are required instead of 8.

Study the same scenario:

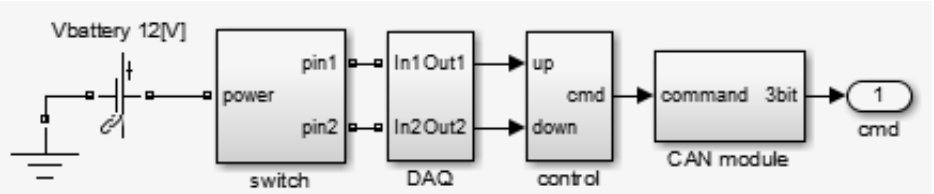
1. In the slexPowerWindowExample/passenger_switch/Normal block, set the passenger up switch.
2. Run the simulation.
3. After some time, in the slexPowerWindowExample/passenger_switch/Normal block, switch off the passenger up switch.
4. When the window has been rolled down, click the slexPowerWindowExample/passenger_switch/Normal block driver down switch.
5. After some time, switch off the slexPowerWindowExample/passenger_switch/Normal block driver down switch.
6. When the window reaches the bottom of the frame, stop the simulation.
7. Zoom into the armature_current scope window and notice the discretized appearance.

Communication Protocols

Similar to the power window output control, hardware must generate the input events. In this case, the hardware is the window control switches in the door and center control panels. Local processors generate these events and then communicate them to the window controller via a CAN bus.

To include these events, add a variant containing input from a CAN bus and switch components that generate the events delivered on the CAN bus to the driver switch and passenger switch variant subsystems. To open the model and configure the CAN communication protocols, run the configureModel task, `slexPowerWindowCommunicationProtocolSim`.

To see the implementation of the switch subsystem, double-click the slexPowerWindowExample/driver_switch/Communication Protocol/driver window control switch block.



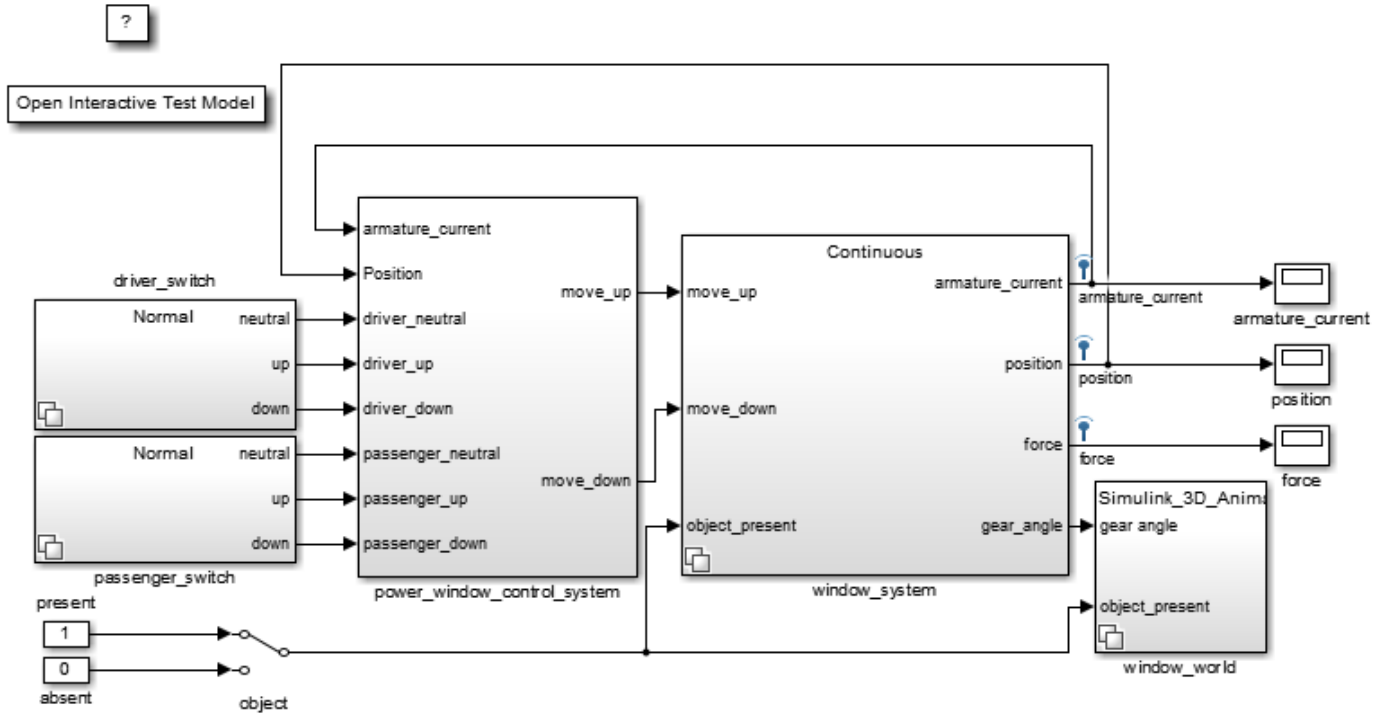
Observe a structure that is very similar to the window control system. This structure contains a:

- Plant model that represents the control switch
- Data acquisition subsystem that includes, among other things, signal conditioning components
- Control module to map the commands from the physical switch to logical commands
- CAN module to post the events to the vehicle data bus

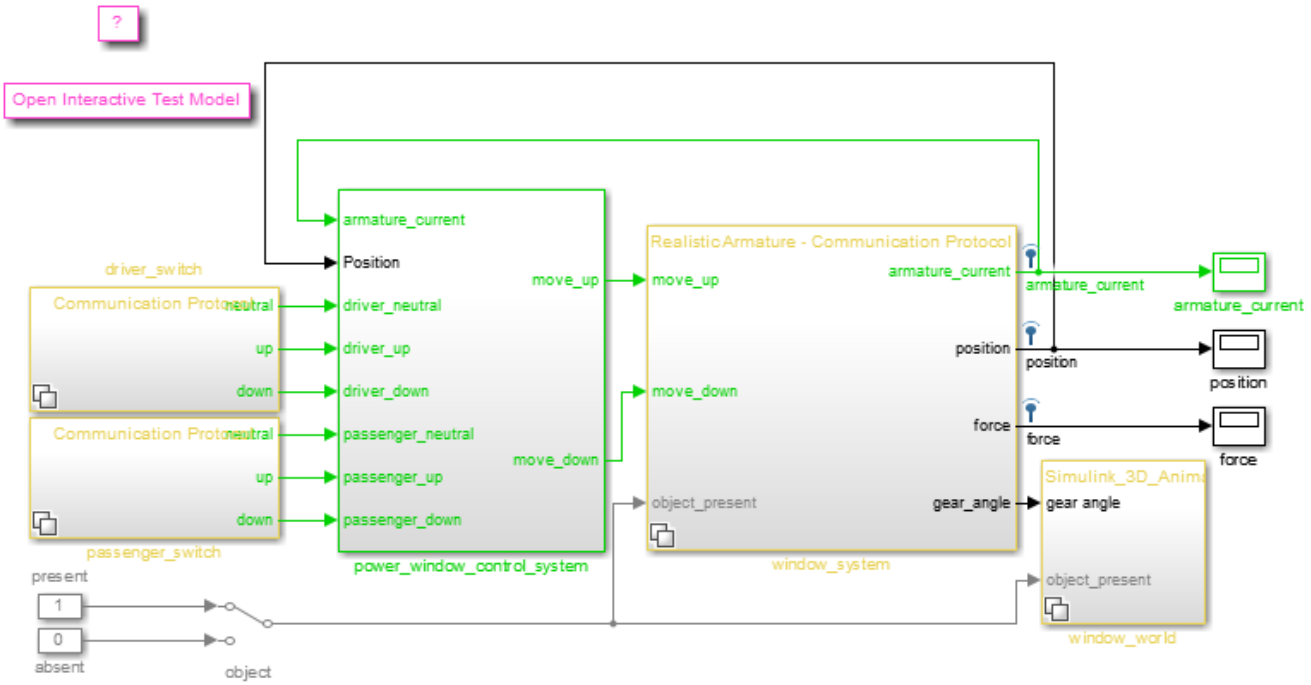
You can add communication effects, such as other systems using the CAN bus, and more realism similar to the described phases. Each phase allows analysis of the discrete-event controller in an increasingly realistic situation. When you have enough detail, you can automatically generate controller code for any specific target platform.

Automatic Code Generation for Control Subsystem

You can generate code for the designed control model, `slexPowerWindowExample`.



1. Display the sample rates of the controller. In the Simulink Editor, select **Display** > **Sample Time** > **Colors**. Observe that the controller runs at a uniform sample rate.



2. Right-click the power_window_control_system block and select **C/C++ Code** > **Build This Subsystem**.

References

Mosterman, Pieter J., Janos Sztipanovits, and Sebastian Engell, "Computer-Automated Multiparadigm Modeling in Control Systems Technology," IEEE Transactions on Control Systems Technology, Vol. 12, Number 2, 2004, pp. 223–234.

Related Examples

- Power Window Control Project

More About

- Project Management