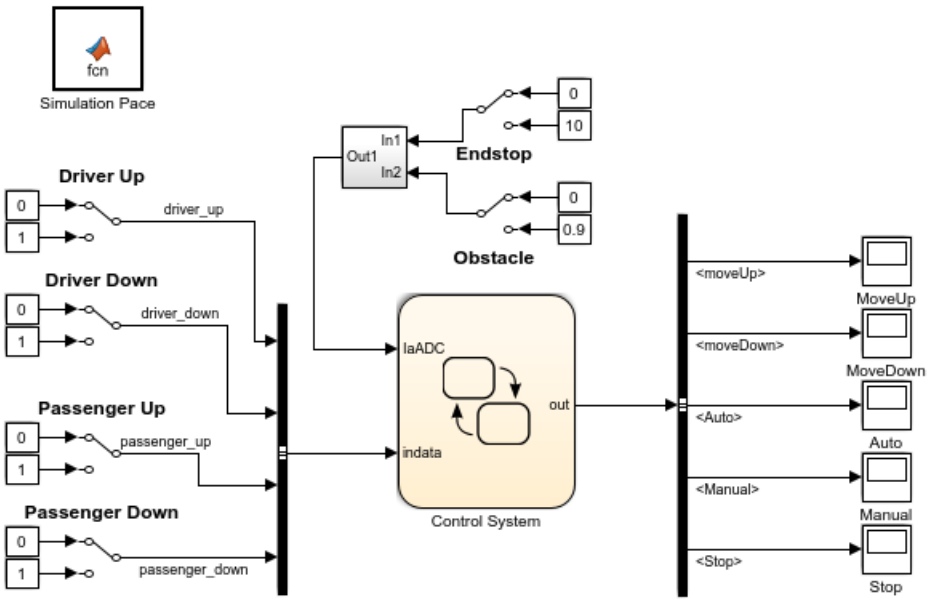
MATLAB Examples

Modeling a Power Window Controller

This example shows how to model an automotive passenger power window system using Model-Based Design with Simulink®, Stateflow®, Fixed-Point Designer™ and DSP System Toolbox™. The Control System design meets a set of requirements.

The power window system models the response of the passenger window to the driver or passenger window controls. To make the system low cost, current sensors are used to provide the sensor inputs. No position sensors are used.



System Requirements

The performance requirements for the system are listed below. The Control System uses fault detection algorithms to protect the window hardware and any obstacles in the window's path. It also uses mode logic to decide when the window should be moved and in which direction.

1) The window must be fully opened or closed within 5[s].

2) The motor must shut off after 5[s] of continuous movement in any

Related Examples

Power Window Control

This example shows how y
use MathWorks® software
Model-Based Development
to go from concept to

direction, as a fail safe protection for the window mechanism, motor, and drive.

3) The window must start moving no later than 0.2[s] after the command is issued.

4) The window must stop when it reaches a fully opened or fully closed position.

5) If the up or down command is issued for a duration of 0.2[s] - 1[s], the window must be fully opened or closed, unless interrupted by a new window command or an obstacle. This requirement represents the automatic-up and automatic-down capability of the power window.

6) The window must be able to detect an obstacle with a force less than 100[N].

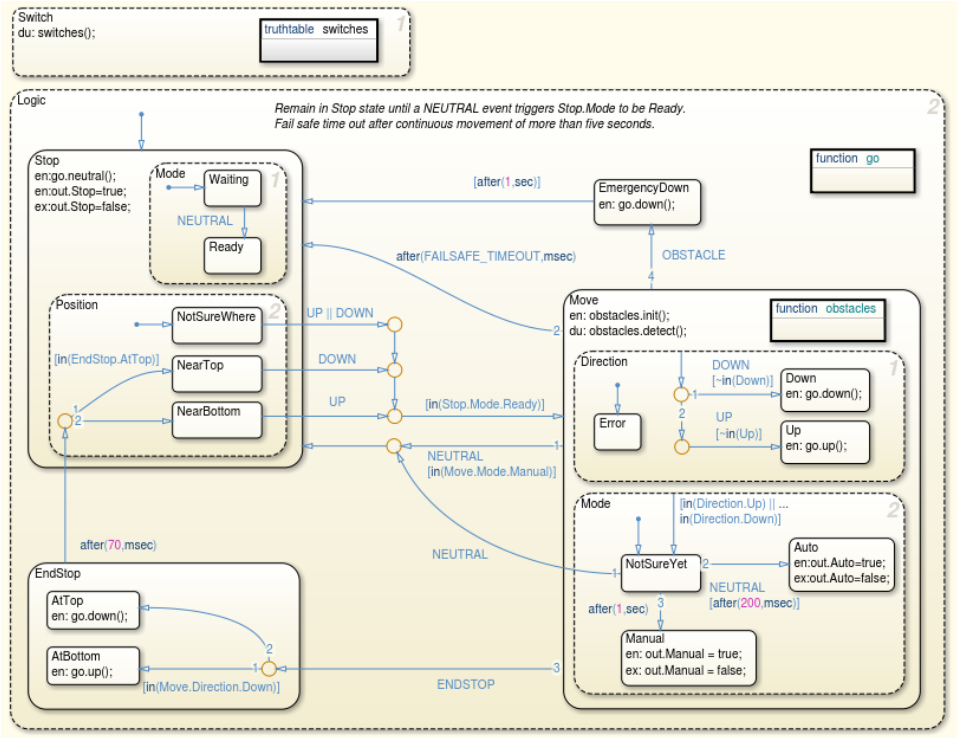
7) The window must be lowered by approximately 10[cm] if an obstacle is detected.

8) Obstacle detection has priority over both driver-side and passenger-side controls.

9) Driver-side controls have priority over passenger-side controls.

Control System Design

The following sections provide a description of the event-driven controller implemented in Stateflow®. The following figure shows the complete high-level Stateflow® chart.



Power Window Control Temporal Properties

This example shows how t temporal system requireme power window controller m property proving and test c

NOTE: Bold text in the description below indicates a reference to a specific state, truth table, or graphical function.

Controller Input

The chart, Control System, must determine the proper window motion based upon the following input values and the specified system requirements. The up and down motion of the power window switches are treated as separate inputs to the controller.

Input 1: Driver-side window switch pressed up

Input 2: Driver-side window switch pressed down

Input 3: Passenger-side window switch pressed up

Input 4: Passenger-side window switch pressed down

Input 5: Motor armature current (sensor input)

The four window control buttons are represented by logical data values true/false signifying on/off. The motor armature current is represented by a fixed-point data value. With these input values, the controller determines whether the window moves up, down, or not at all.

Obstacle Detection

The system monitors the load on the motor to detect obstacles. When the window encounters an obstacle, the applied force on the window increases the load on the motor and hence the armature current. By monitoring for sharp increases in the armature current, the system can detect obstacles in the window's path.

Controller States and Events

The chart has two parent, parallel states named **Switch** and **Logic**. Parallel states are active simultaneously. **Logic** contains four exclusive states **Stop**, **Move**, **EmergencyDown**, and **EndStop**. As the names suggest, these states handle the control logic for the window to stop, move, move down in an emergency, or detect the endstops. Events based on time or user inputs cause changes in state. The time-based events, called temporal logic events, invoke safety-critical features for obstacle detection and hardware protection. The user-driven events are as follows.

1) UP: Window up event

2) DOWN: Window down event

3) NEUTRAL: Window hold event

4) ENDSTOP: End of range detected event when window is fully open or closed

5) OBSTACLE: Window obstacle event

A brief description of how these events occur as well as the four major states follows.

Event Broadcasting - Truth Table and Graphical Functions

The events in this controller are broadcast from the truth table, **switches**, and the graphical function, **obstacles** (respectively in states **Switch** and **Move**). The purpose of **switches** is to broadcast the UP, DOWN, and NEUTRAL events based upon a button or combination of buttons pressed. The purpose of the **obstacles** function is to decide if an obstacle is present or if the window is fully open or fully closed. This decision is based upon the magnitude and rate of change of the armature current. If an obstacle is present, the OBSTACLE event is broadcast, and if the window reaches a fully open or fully closed position, the ENDSTOP event is broadcast. A diagram of the truth table is shown in the figure below. As you can see, the driver's inputs take precedence over the passenger's. If the driver and passenger both press their window buttons, the passenger input will be ignored.

Stateflow (truth table) sf_power_window/Contrc

File Edit Settings Add Help

Condition Table

	Description	Condition	D1	D2	D3	D4	D5
1		indata.driver_up	T	F	F	F	-
2		indata.driver_down	F	T	F	F	-
3		indata.passenger_up	-	-	T	F	-
4		indata.passenger_down	-	-	F	T	-
		Actions: Specify a row from the Action Table	1	2	1	2	3

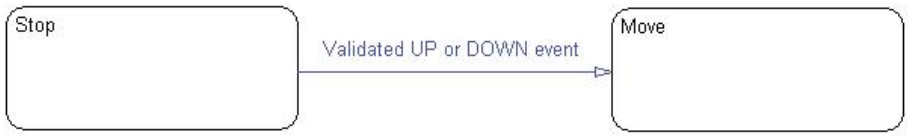
Action Table

#	Description	Action
1	Move Up	send(UP,Logic);
2	Move Down	send(DOWN,Logic);
3	Stay Neutral	send(NEUTRAL,Logic);

The final graphical function, **go**, does not broadcast any events to the Stateflow® chart. The purpose of **go** is to set the chart output values, which correspond to the direction of window motion. For example, if the controller decides that the window should move up, it passes the value "true" to the first output port. Likewise, the value "true" is sent to the second output port if the controller decides that the window should move down.

Stop State

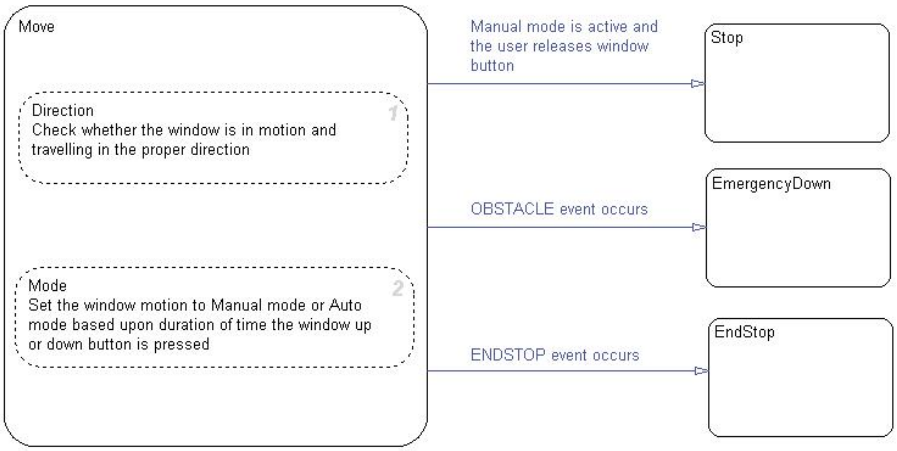
The **Stop** state is active whenever the window is not moving. **Stop** is the first state that is activated when the simulation begins, since by default the window is not moving at the start of simulation. A transition from **Stop** to **Move** occurs if a validated UP or DOWN event is broadcast. An UP event is invalid if it occurs when the window is already fully closed. Likewise, a DOWN event is invalid if it occurs when the window is fully open.



Move State

The **Move** state is active whenever the window is in motion. This state implements several of the power window requirements related to object detection and automated window movement. At each time step, **Move** checks whether the window has run into an obstacle or reached a fully open or fully closed position. If an object is detected, the OBSTACLE event is broadcast and a state transition from **Move** to **EmergencyDown** occurs. If an end position is detected, the ENDSTOP event is broadcast and a state transition from **Move** to **EndStop** occurs.

Move has two parallel states named **Direction** and **Mode**. The **Direction** state checks whether the window is in motion and traveling up or down. The **Mode** state implements the auto-up and auto-down capability of the window. If a window button is held for a duration from 0.2 to 1 second in either direction, the window should automatically open or close completely, unless interrupted by a new window command or an obstacle. If the user presses a window button for the duration specified above, the **Auto** state is entered. If the button is pressed for a duration that does not land within this range, the **Manual** state is entered. If the **Manual** state is active, a transition from **Move** to **Stop** is executed when the user releases the button.

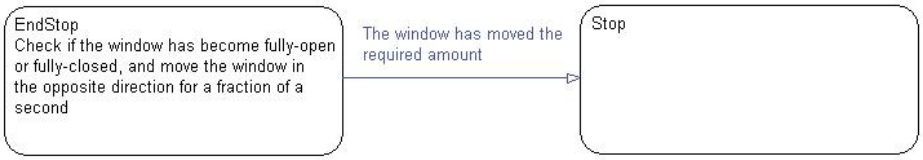


EmergencyDown State

In the **EmergencyDown** state, the window moves down about 10 [cm] if an obstacle is detected. When this state is activated, the **go** function is called to move the window down.

EndStop State

EndStop first determines whether the window is fully opened or closed, then commands the window to move in the opposite direction for a fraction of a second to relieve the force on the window. Once the window has moved the required amount, a state transition from **EndStop** to **Stop** occurs.



mathworks.com