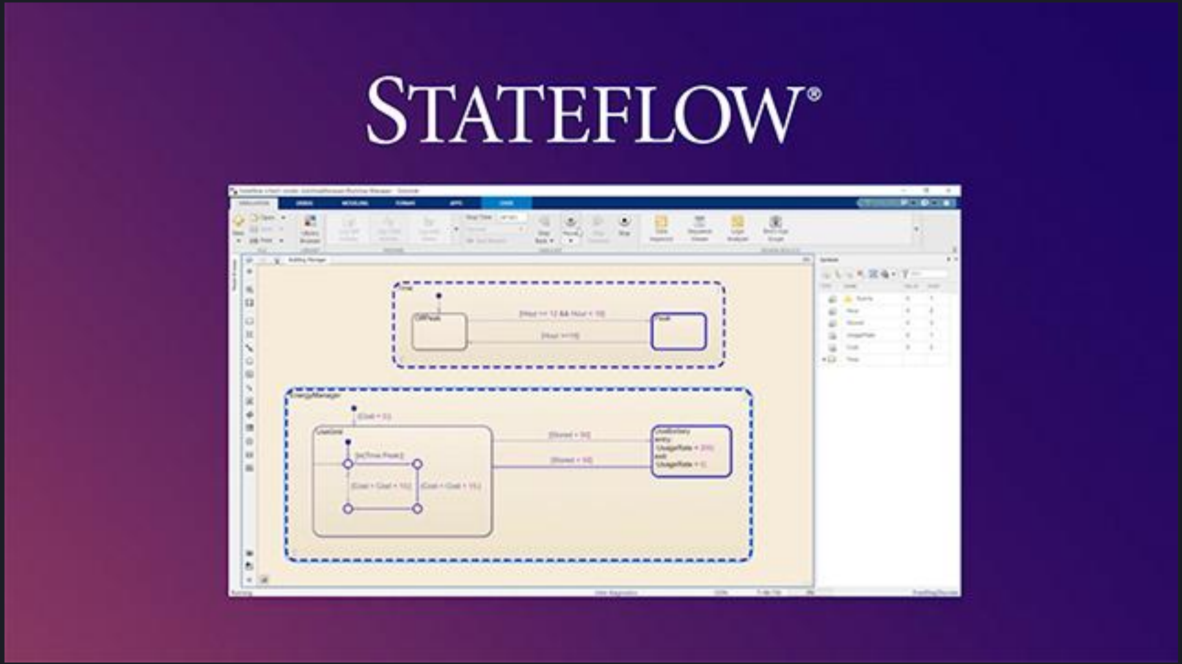
Stateflow Design

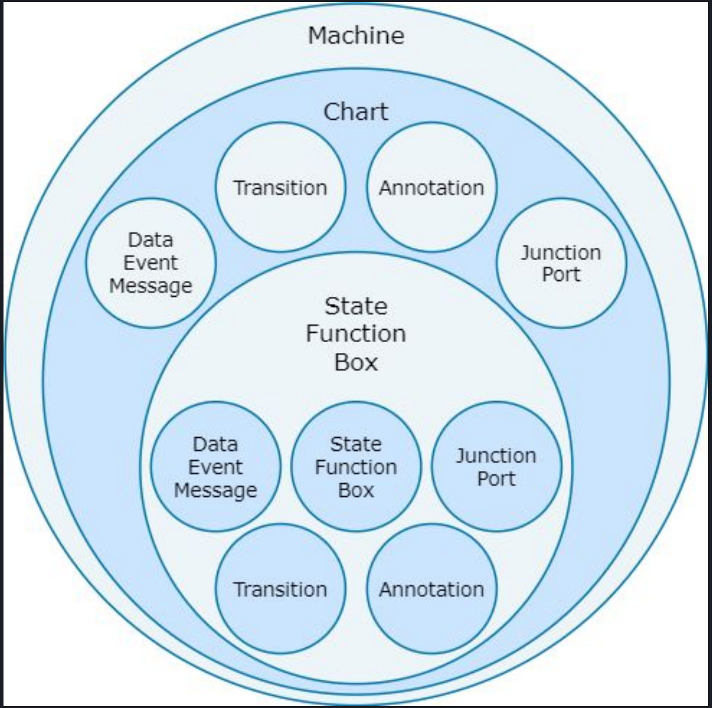


Model-Based Development Program

Stateflow Design

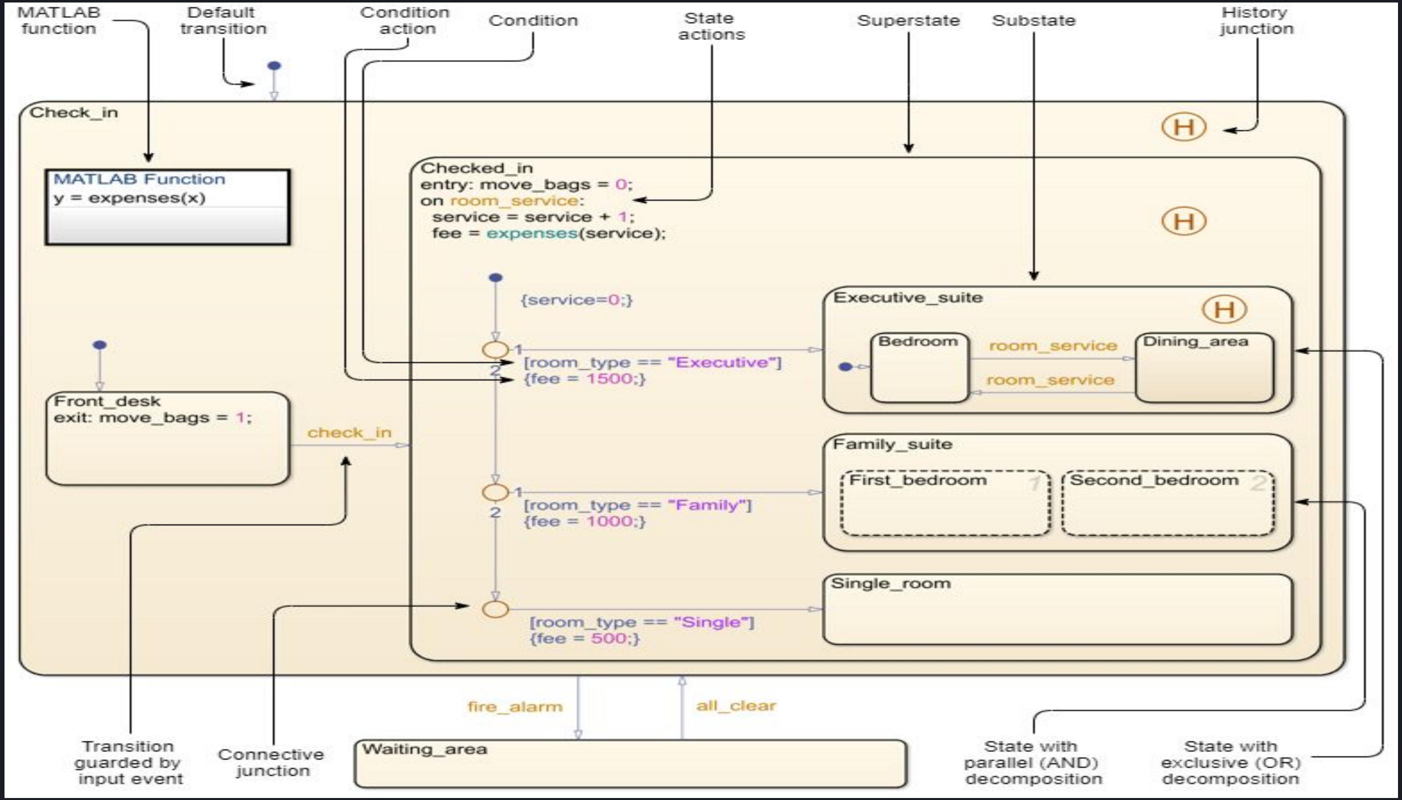
Stateflow Overview

Type of Graphical Object	
State	Represent Operating Modes by Using States
Transition	Transition Between Operating Modes
Connective junction	Combine Transitions and Junctions to Create Branching Paths
Box	Group Chart Objects by Using Boxes
Simulink based state	Create and Edit Simulink Based States
Simulink function	Reuse Simulink Functions in Stateflow Charts
Graphical function	Reuse Logic Patterns by Defining Graphical Functions
MATLAB® function	Reuse MATLAB Code by Defining MATLAB Functions
Truth table function	Use Truth Tables to Model Combinatorial Logic
History junction	Resume Prior Substate Activity by Using History Junctions
Exit junction	Create Entry and Exit Connections Across State Boundaries
Entry junction	Create Entry and Exit Connections Across State Boundaries
Annotation	Add Descriptive Comments in a Chart
Image	Add Descriptive Comments in a Chart



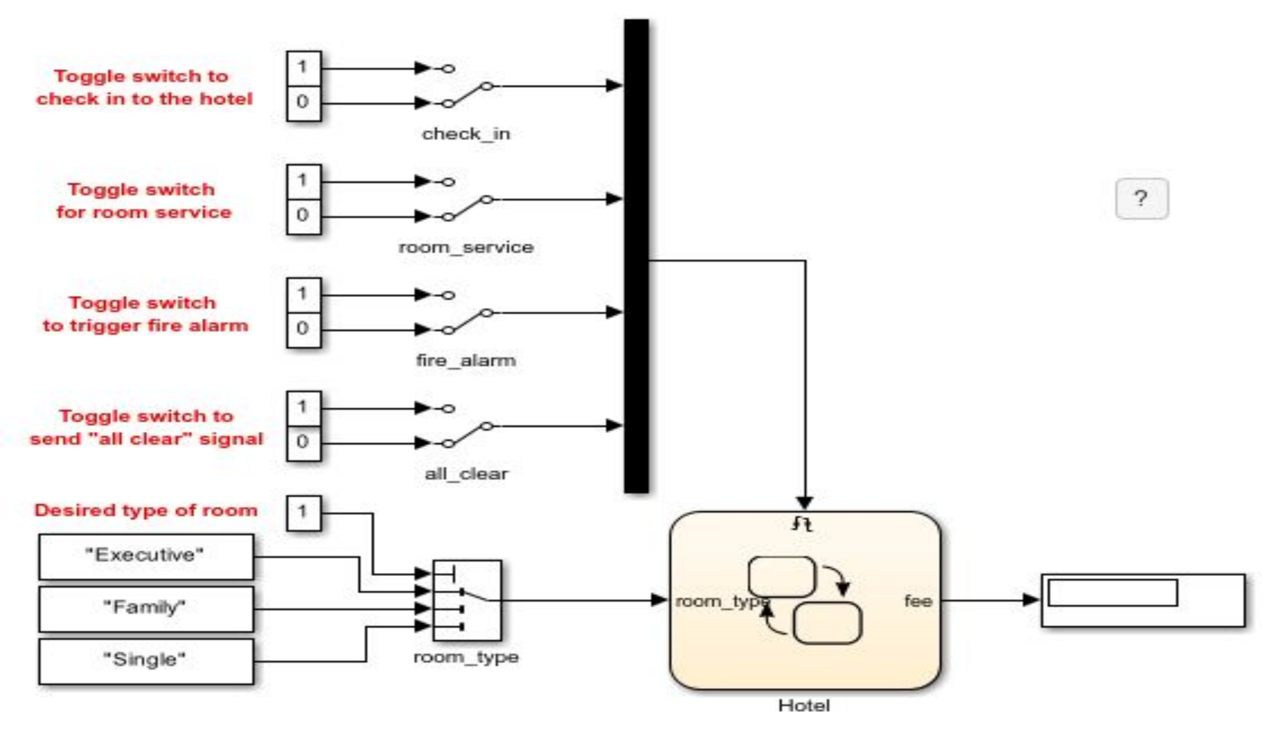
Stateflow Design

Stateflow Overview



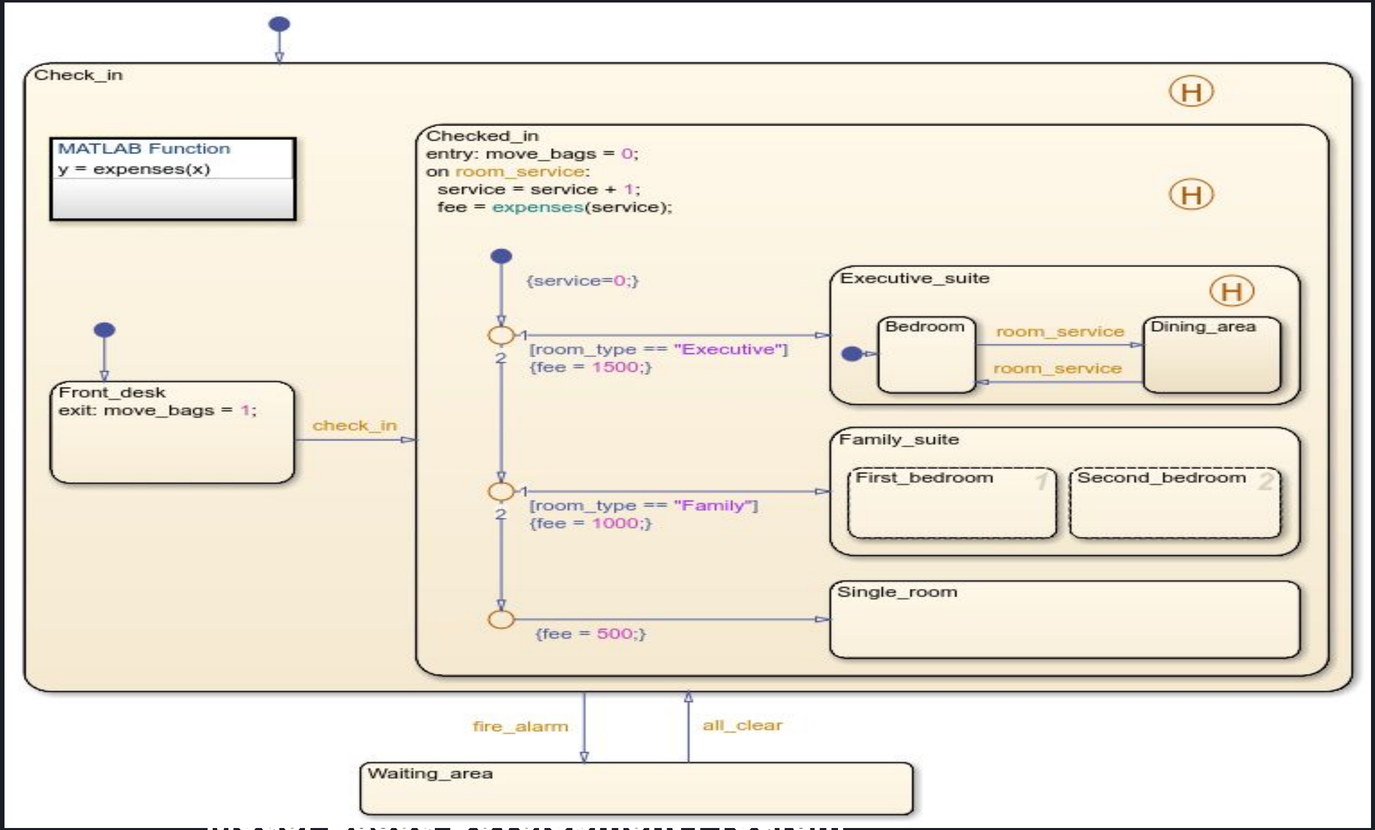
Stateflow Design

Stateflow Overview



Stateflow Design

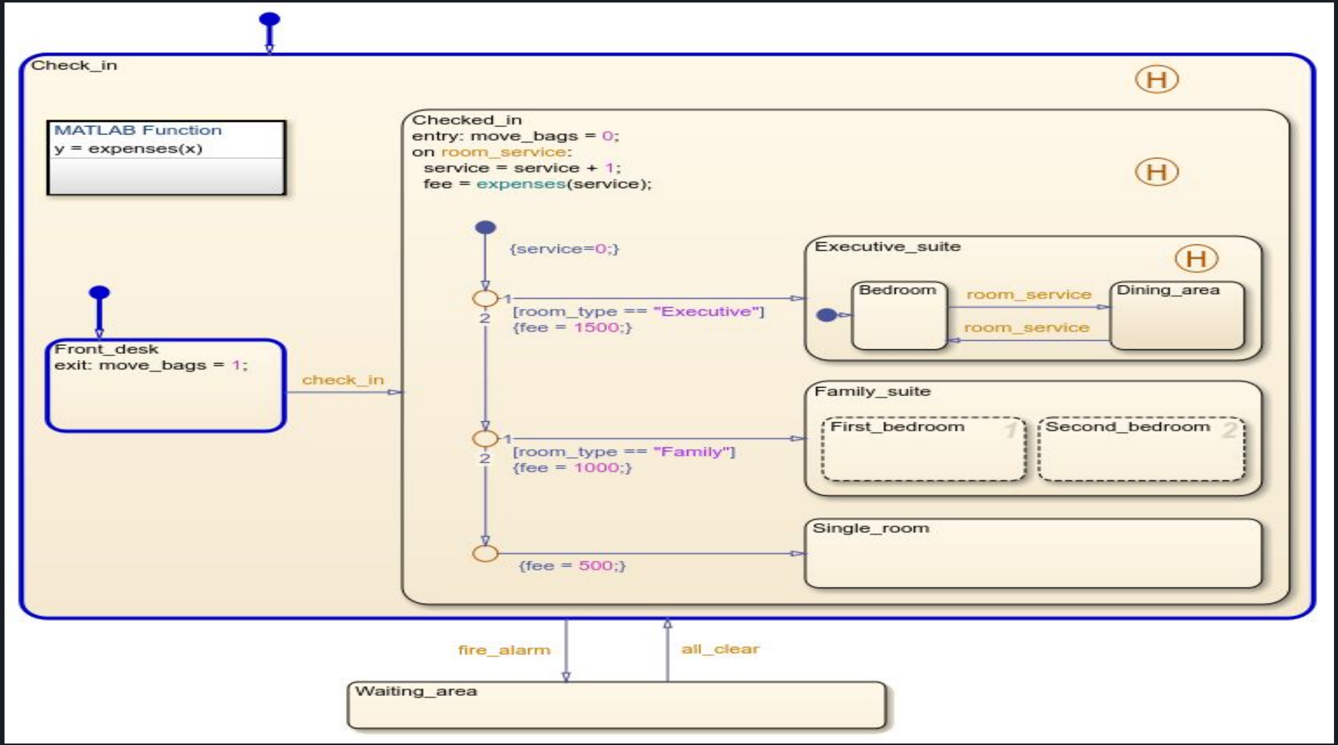
Stateflow Overview



Stateflow Design

Stateflow Overview

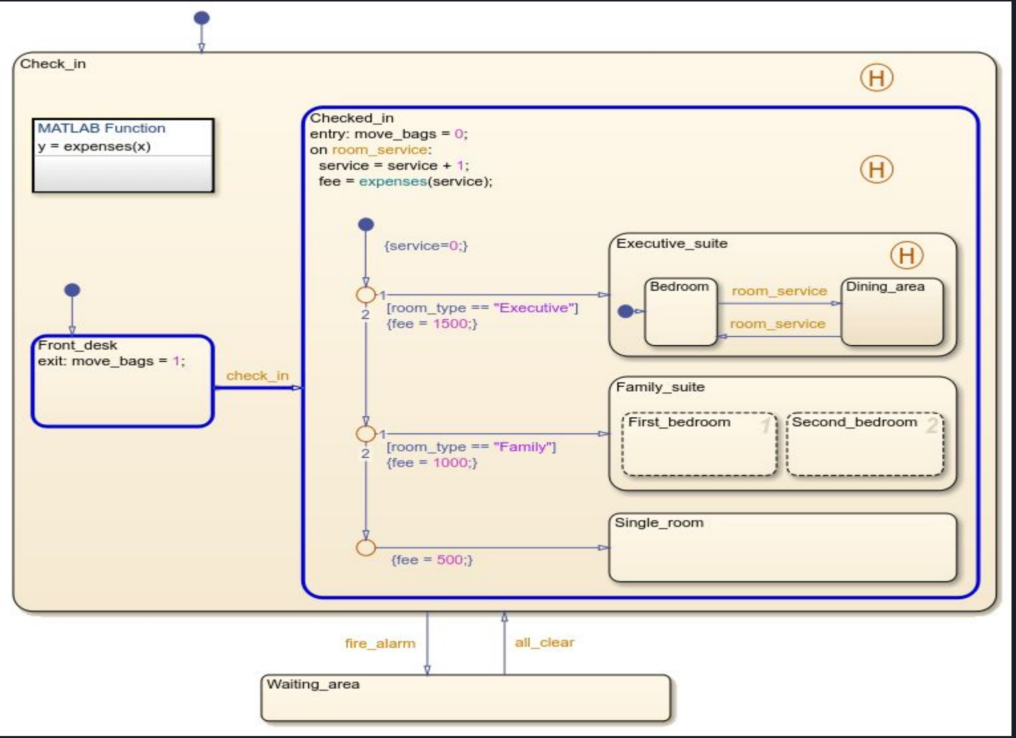
- Chart Initialization



Stateflow Design

Stateflow Overview

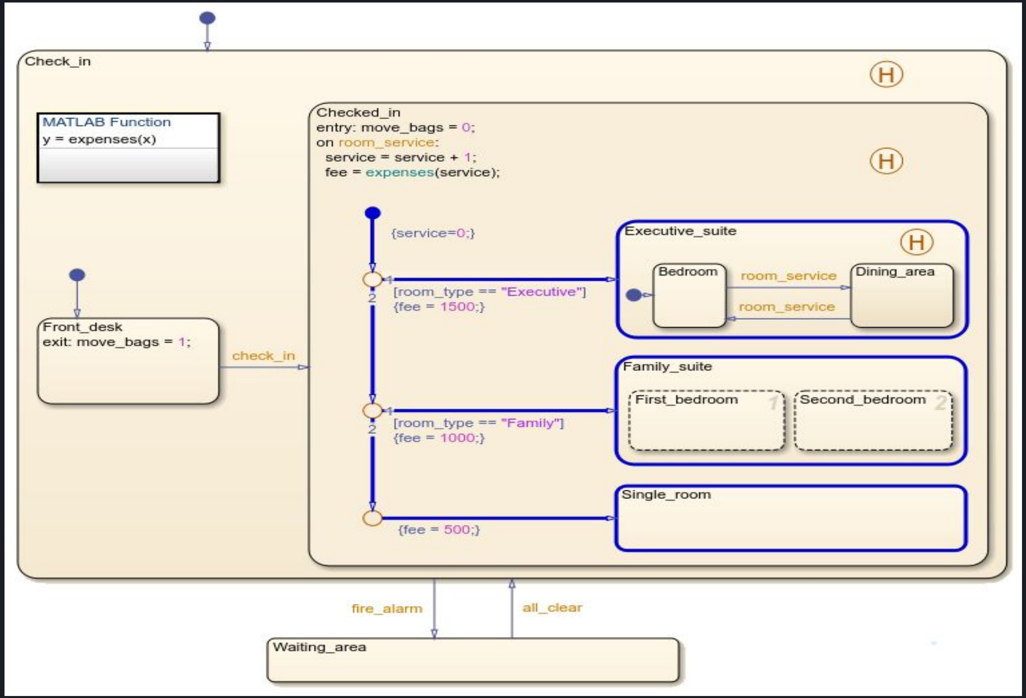
- Transition Between States



Stateflow Design

Stateflow Overview

- Evaluation of default Transition Paths

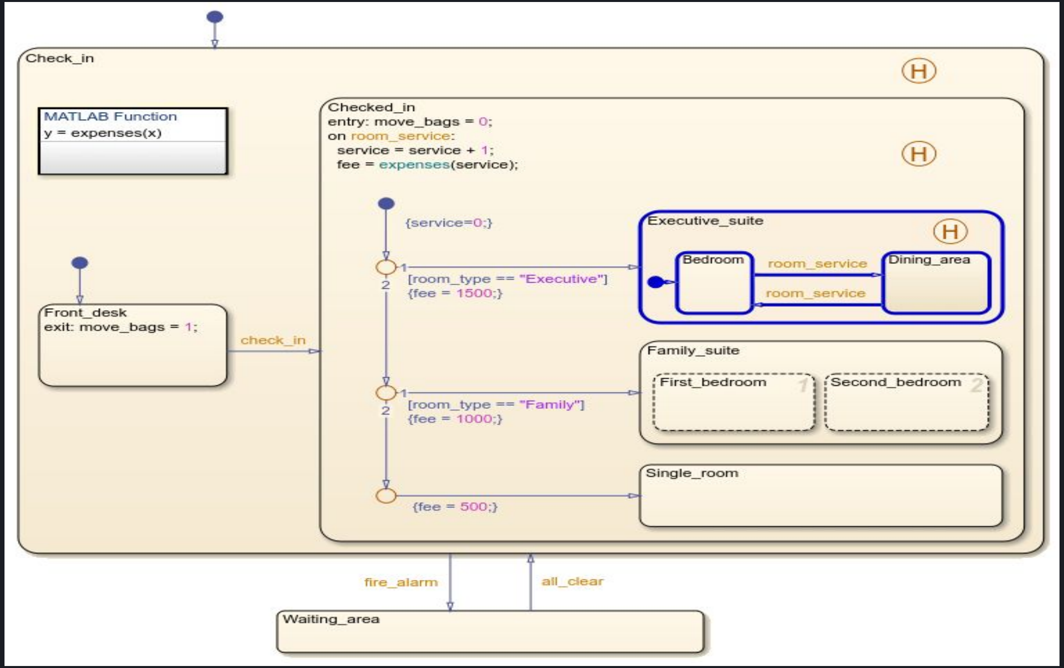


DS-Academy

Stateflow Design

Stateflow Overview

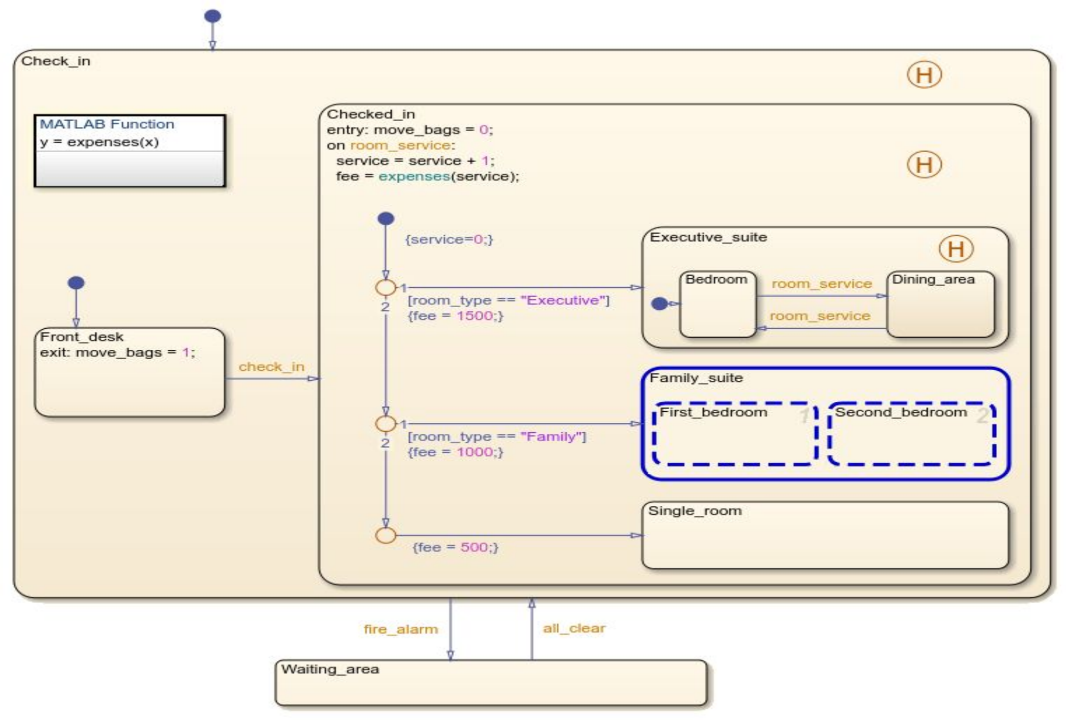
- Execution of states with exclusive Substates



Stateflow Design

Stateflow Overview

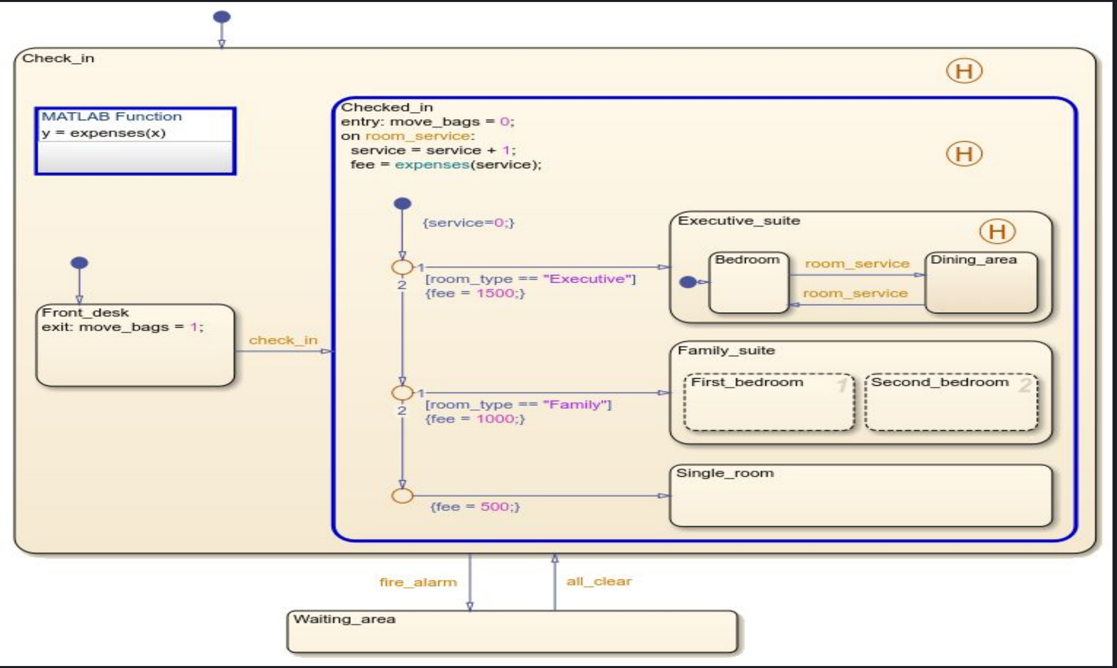
- Execution of states with Parallel Substates



Stateflow Design

Stateflow Overview

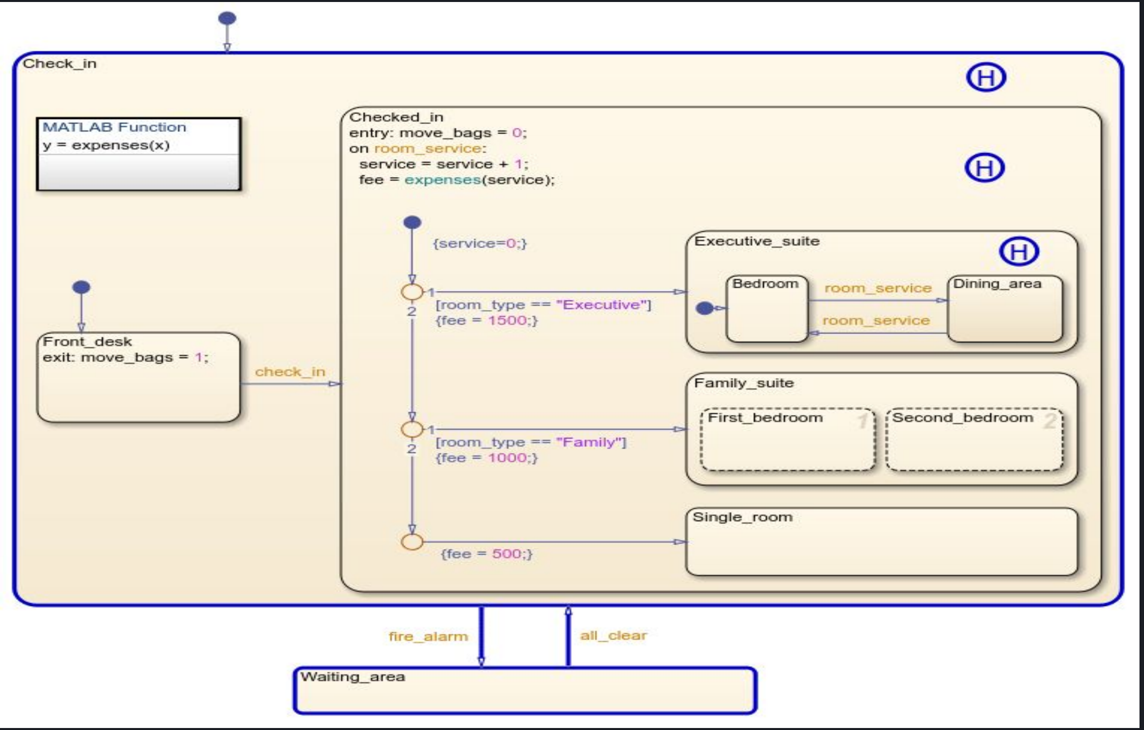
- Function Call From a State Action



Stateflow Design

Stateflow Overview

- Execution of States with History



Model-Based Development Program

Stateflow Design

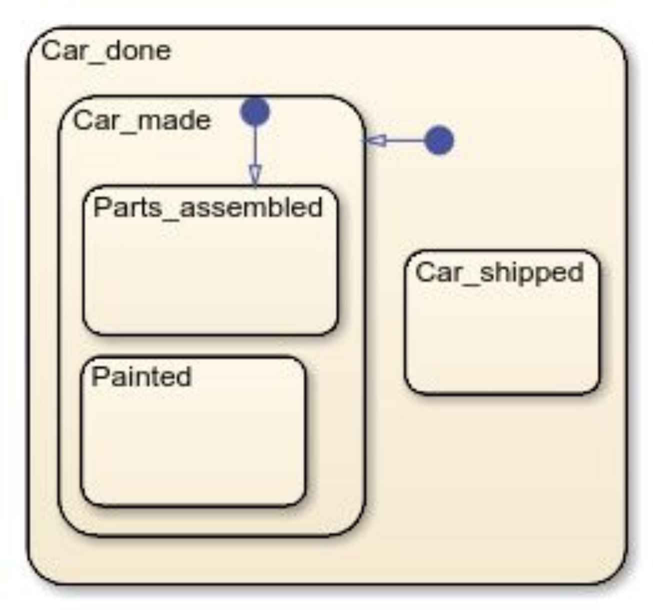
State Machines

- What is a state?
  - A state describes an operating mode of reactive system.
    - Each state shall describe a stable state of the system
- State Name
  - A state label starts with the name of the state.
  - Valid state names consist of alphanumeric characters and can include the underscore(_) character.
  - the state name shall not be a stateflow keyword, neither i nor j an no reserved c experistions
  - within a chart, state named must be different from other data names
  - the state name shall correspond to the state

Stateflow Design

State Machines

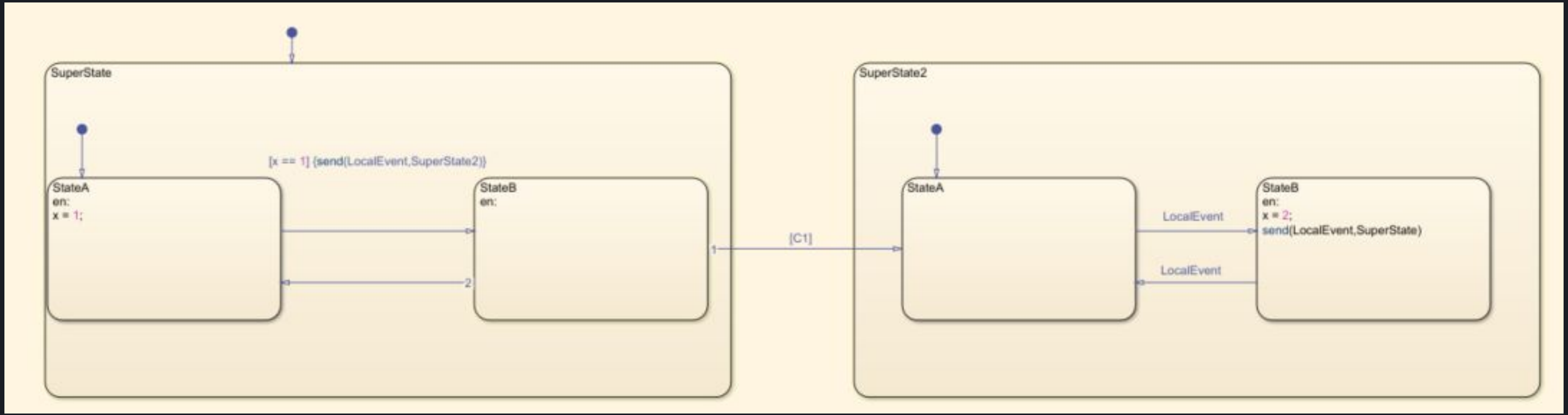
- State Hierarchy
  - A state is a “superstate” if it contains other states
  - A state is a “substate” if it is contained by another state



Stateflow Design

State Machines

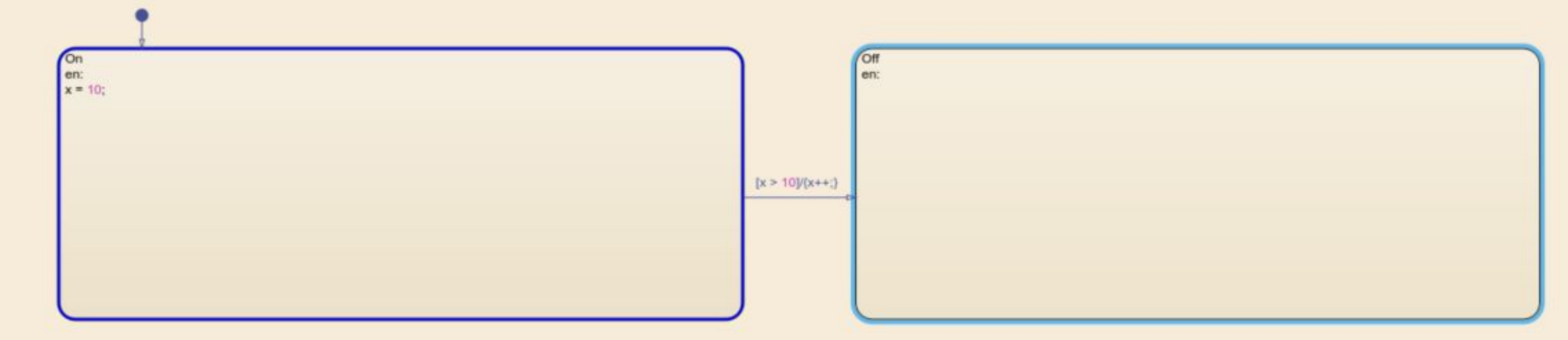
- State Hierarchy
  - A state is a “superstate” if it contains other states
  - A state is a “substate” if it is contained by another state



Stateflow Design

State Machines

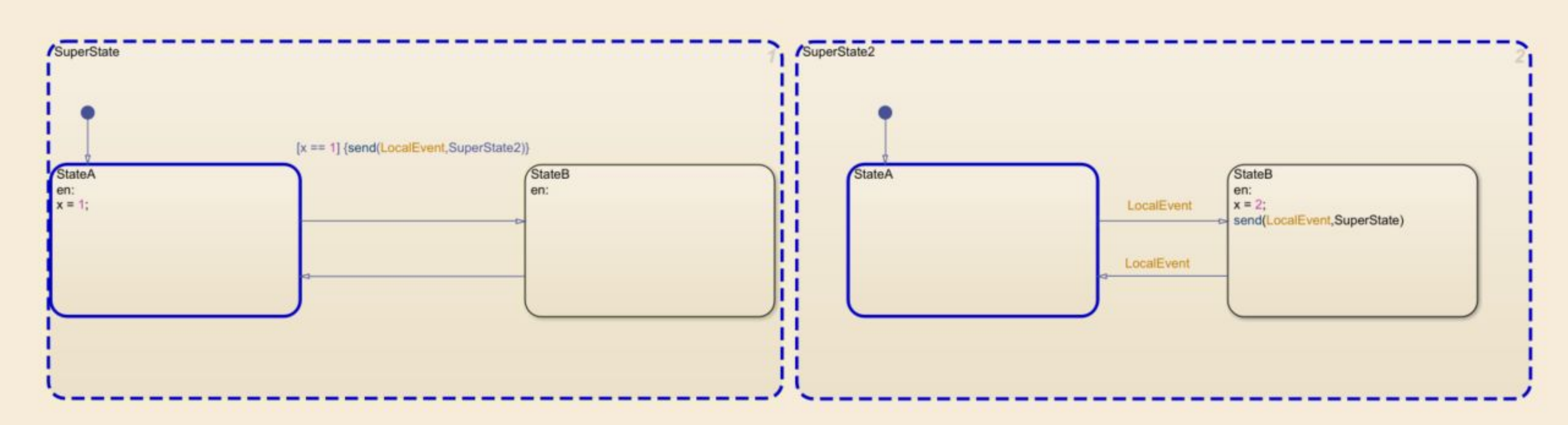
- State Decomposition
  - What are is an exclusive (OR) state?
    - OR States (exclusive states) correspond to the classic state diagram.
    - At a given time only one state is active.



Stateflow Design

State Machines

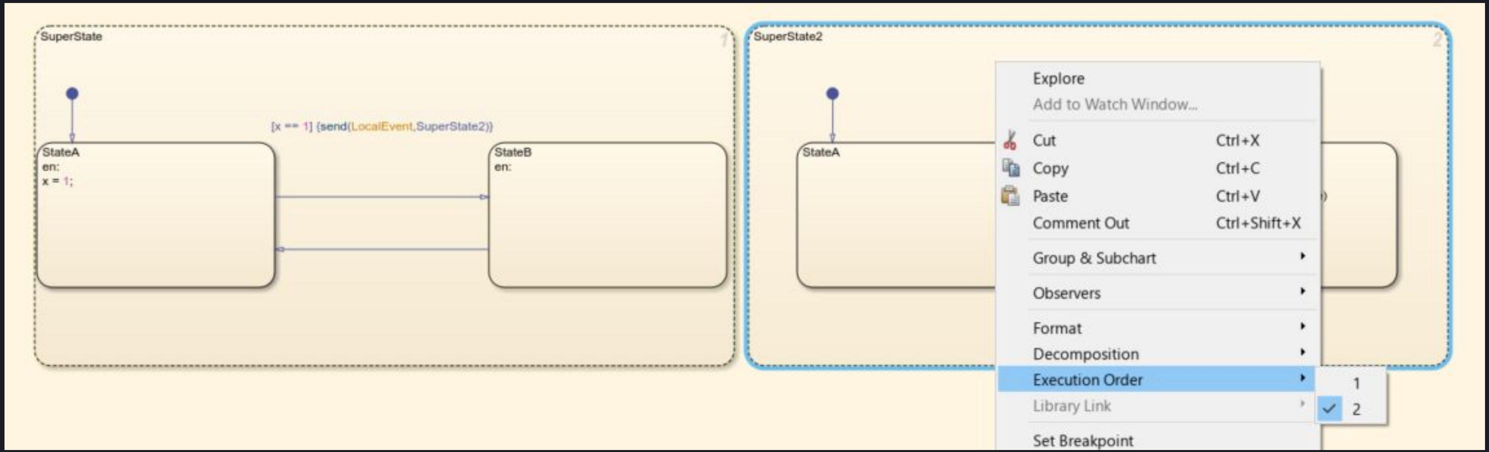
- State Decomposition
  - What are is Parallel (AND) state?
    - And states (Parallel states are states that are executed in
    - the execution is sequential but all active parallel states are executed



Stateflow Design

State Machines

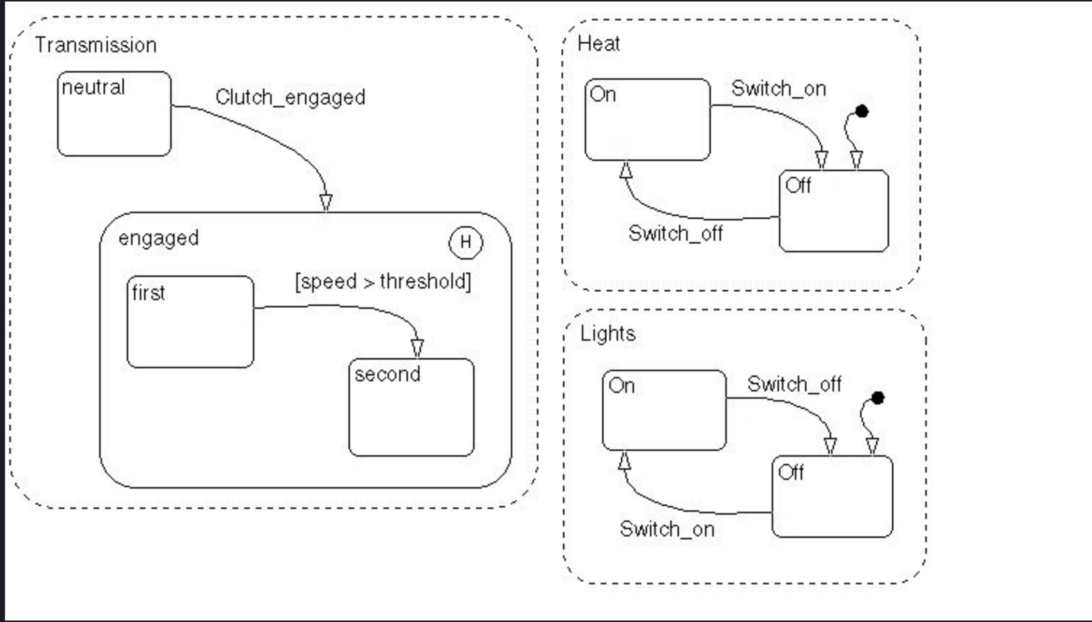
- State Decomposition
  - Setting the execution order
    - by right click in the state and then via the ‘Execution order’ context menu



Stateflow Design

State Machines

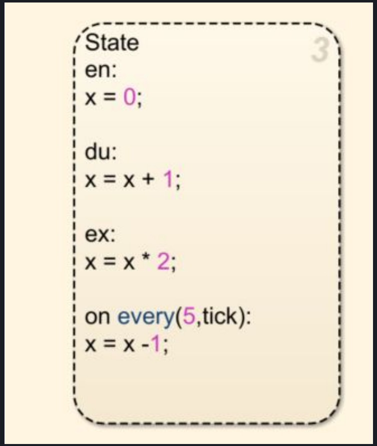
- State Decomposition



Stateflow Design

State Machines

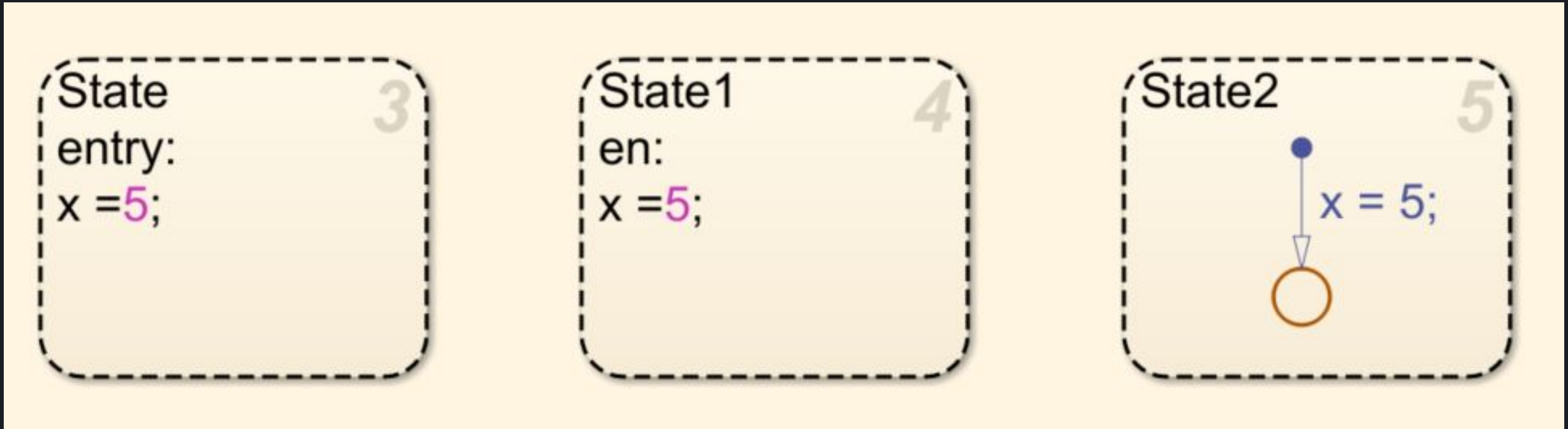
- State Actions
  - State can have different action types initiated by keywords
    - entry (en): executes when the state is entered
    - during (du) :executes when the state is active and no valid state transition to another state is activated
    - exit(ex): executes when the state is active and a transition out of the state occurs.
    - on ‘event_name’: executes when the state is active and the event ‘event_name is received by the state
  - The following rules apply for writing actions:
    - No actions without initiating keyword
    - No action type keyword without action
    - Actions start after a line break
    - Actions are followed with semicolon ;
    - actions can be designed with flows except **exit.**



Stateflow Design

State Machines

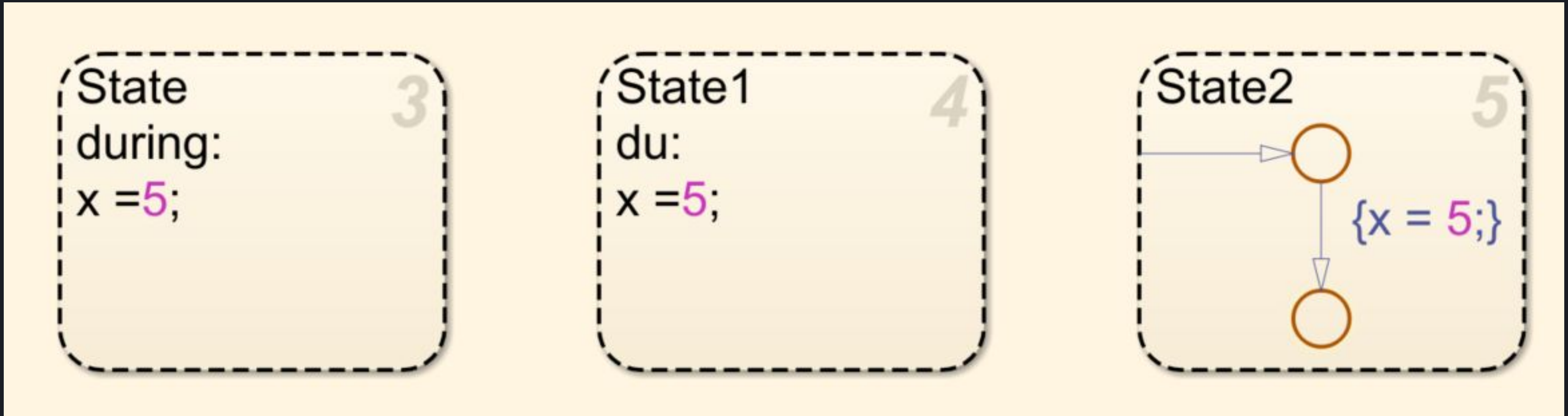
- State Actions
  - Entry Action



Stateflow Design

State Machines

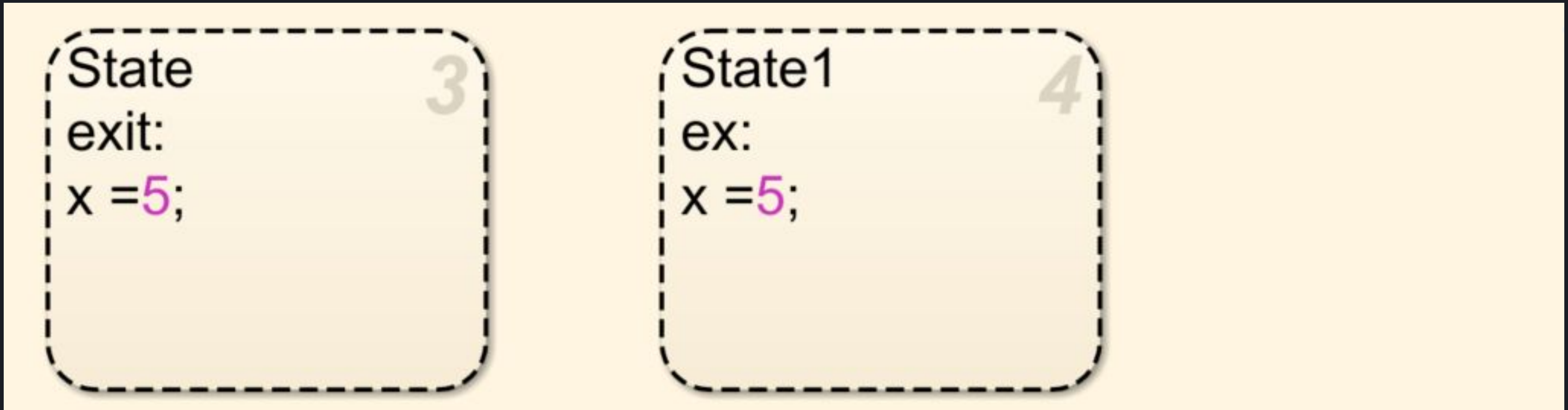
- State Actions
  - During Action



Stateflow Design

State Machines

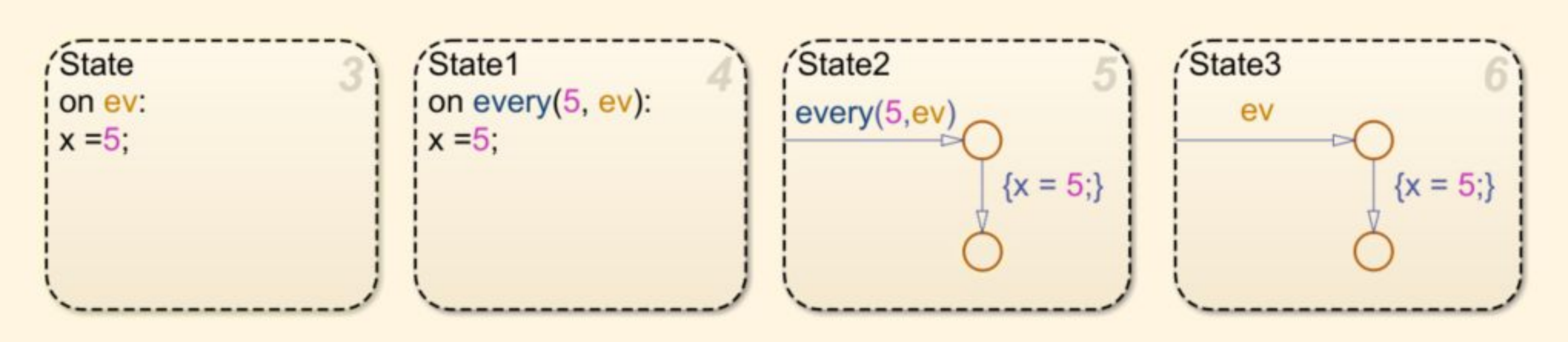
- State Actions
  - Exit Action



Stateflow Design

State Machines

- State Actions
  - on ‘evnet_name’ action



Stateflow Design

State Machines

- Transitions,Conditions & Actions
  - Transition is a line with an arrowhead that links one graphical object to another.
  - a transition connects a source and a destination object.
  - junctions divide a transition into transition segments. in this case, a full transition consists of the segments taken from the origin to the destination
  - each segment is evaluated in the process of determining the validity of a full transition.
  - a transition is characterized by its label. the label can consist of an event, a condition, a condition action, and /or a transition actions

event [condition]{condition_action}/{transition_action}

All Elements are optional

Stateflow Design

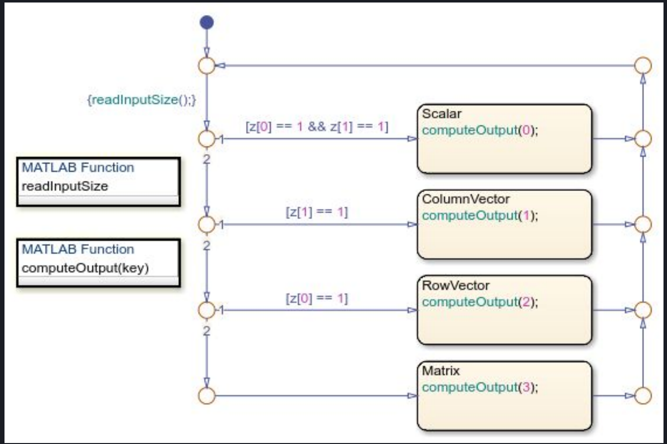
State Machines

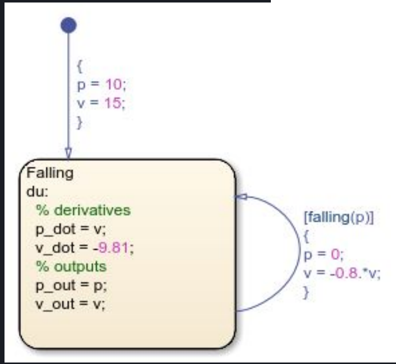
- Transitions,Conditions & Actions
  - Condition Actions
    - Action that is executed whenever its transition segment is valid
    - executed even if it is not on the final transition path
  - Transition Actions
    - Action that is executed when it's in the final validated transition path.
    - The destination must be a state

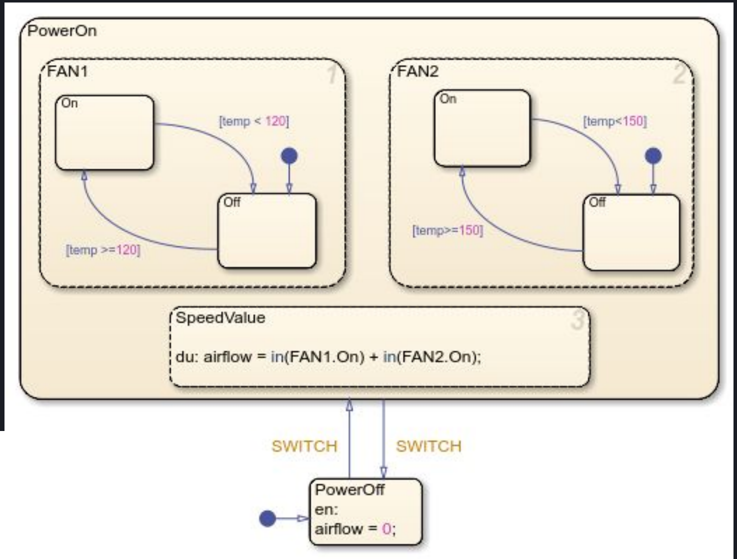
Stateflow Design

State Machines

- Transitions,Conditions & Actions
    - Default Transition to a State
    - default Transition to a junction
    - default Transition with label



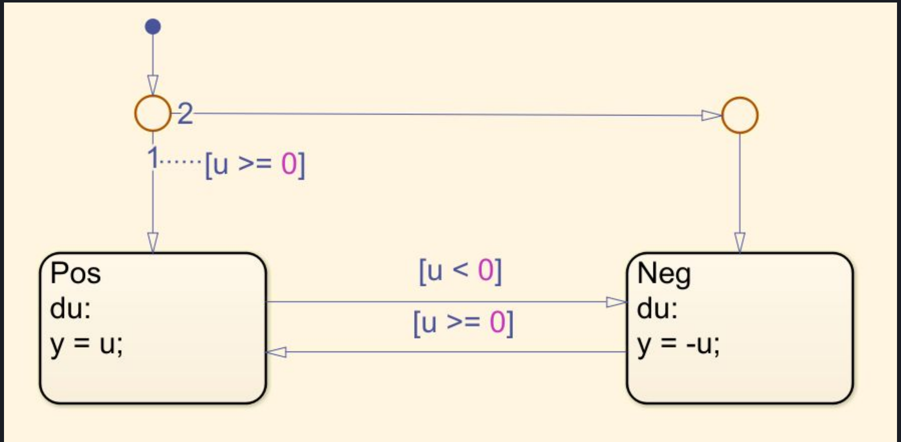




Stateflow Design

State Machines

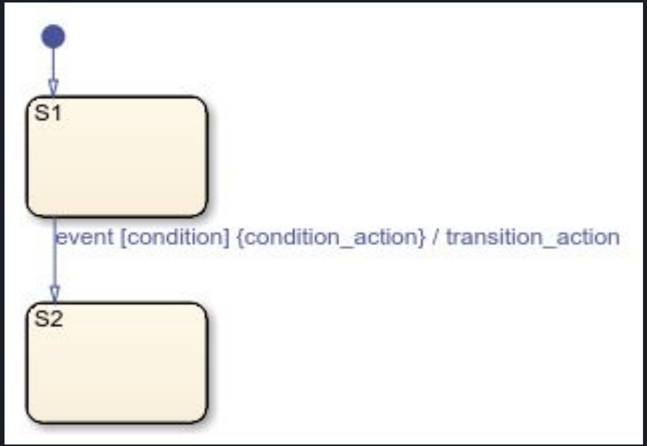
- Transitions,Conditions & Actions
  - Default Transition to a State
  - default Transition to a junction
  - default Transition with label

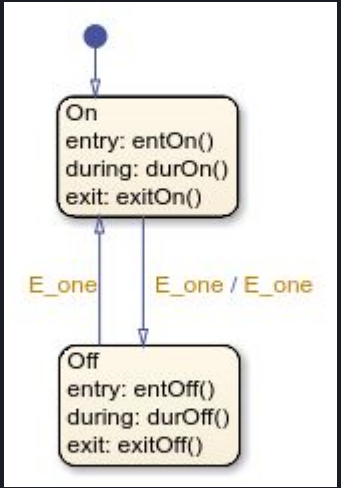


Stateflow Design

State Machines

- Transitions,Conditions & Actions
  - Transitions to and from exclusive states

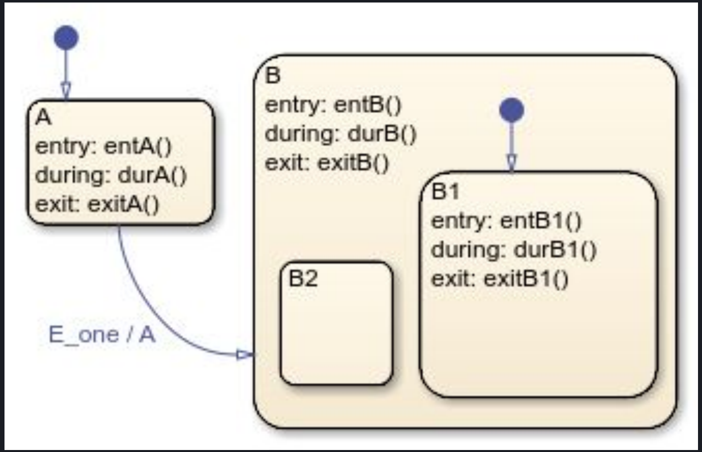


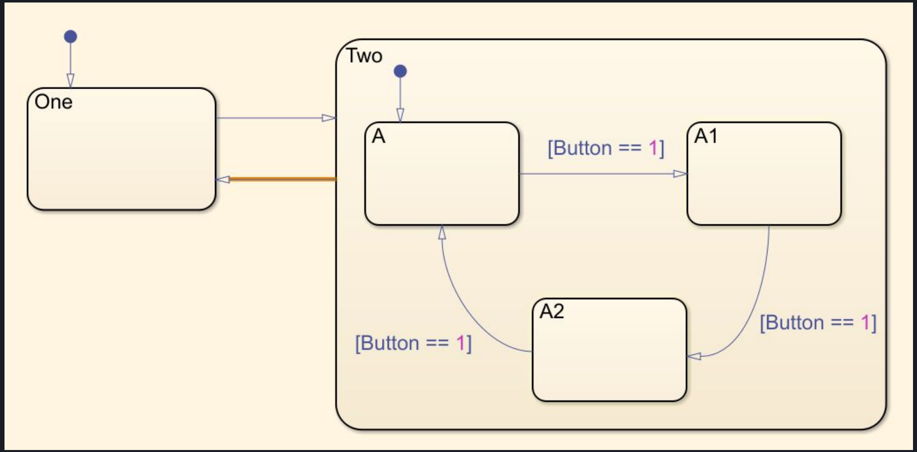


Stateflow Design

State Machines

- Transitions,Conditions & Actions
  - Transitions to and from Exclusive (OR) Superstates

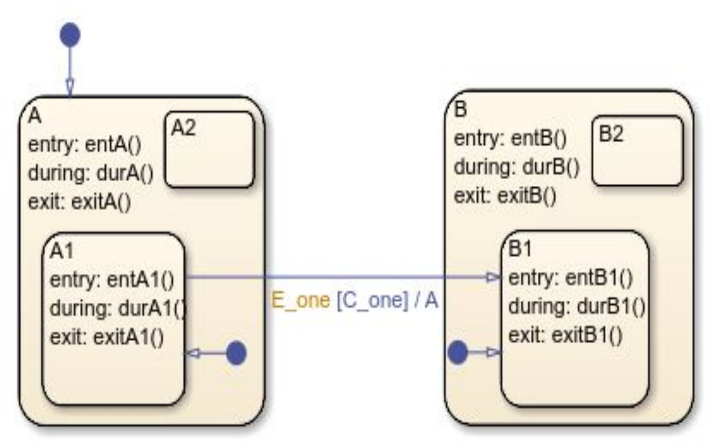


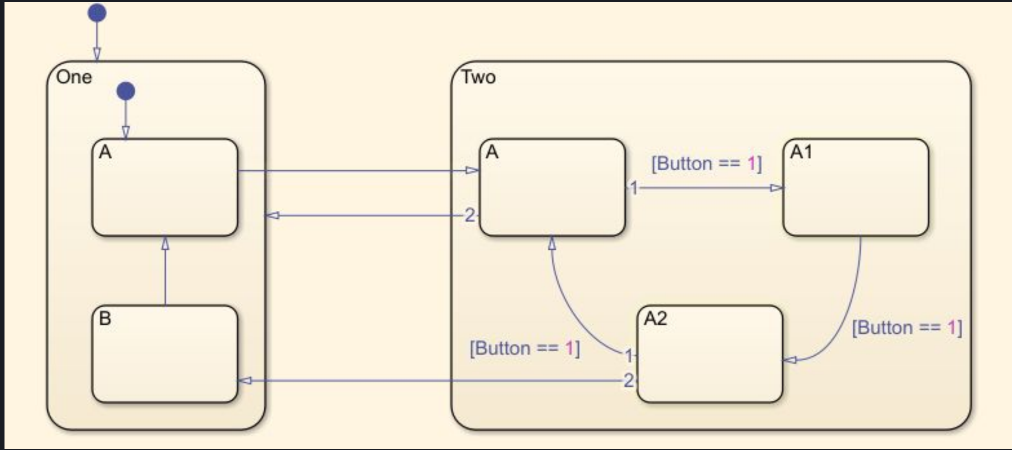


Stateflow Design

State Machines

- Transitions,Conditions & Actions
  - Transitions to and from Exclusive (OR) Substates

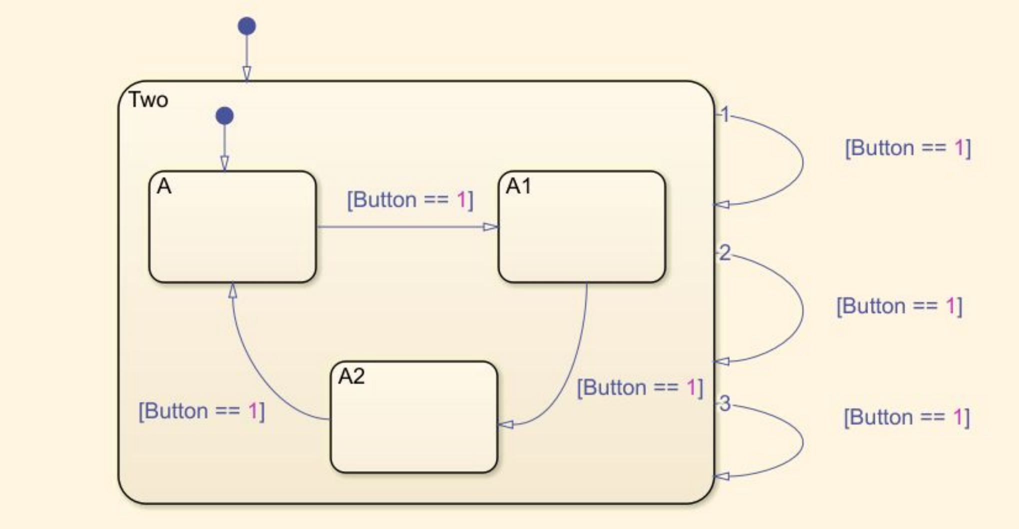




Stateflow Design

State Machines

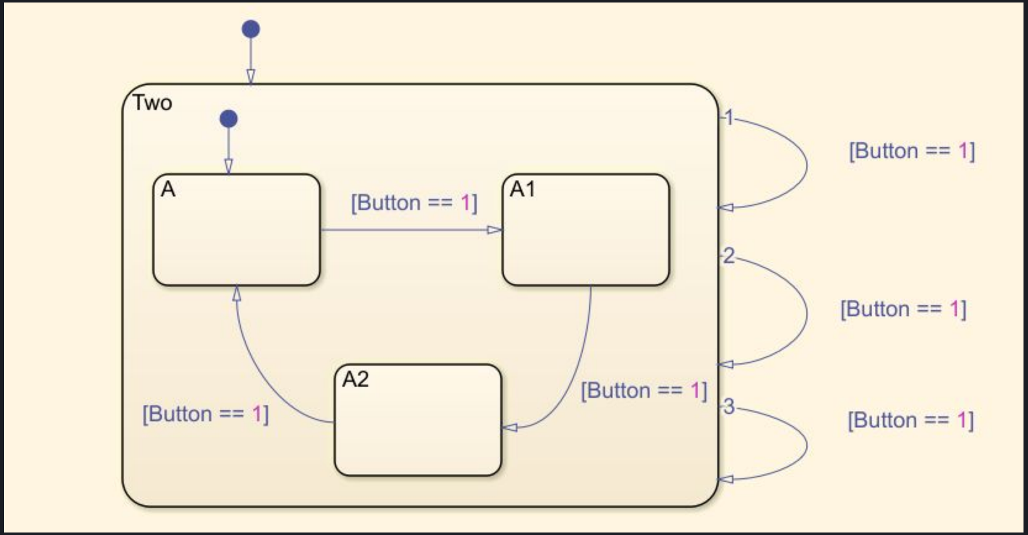
- Transitions,Conditions & Actions
    - Self Loop Transitions



Stateflow Design

State Machines

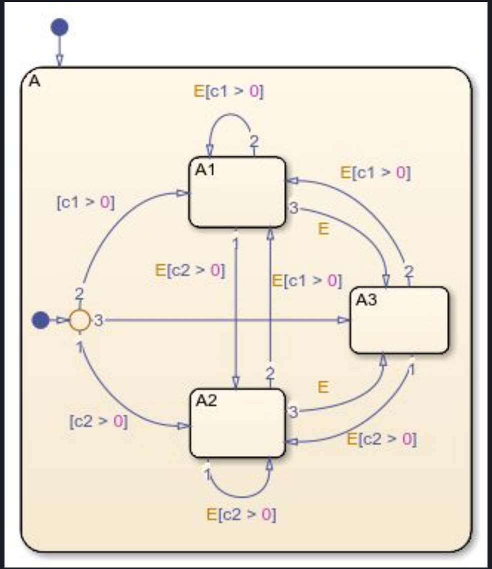
- Transitions,Conditions & Actions
  - Self Loop Transitions

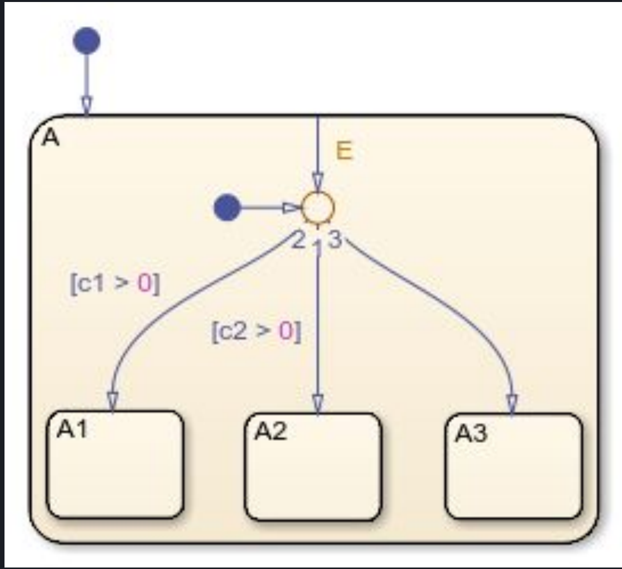


Stateflow Design

State Machines

- Transitions,Conditions & Actions
  - Inner Transitions

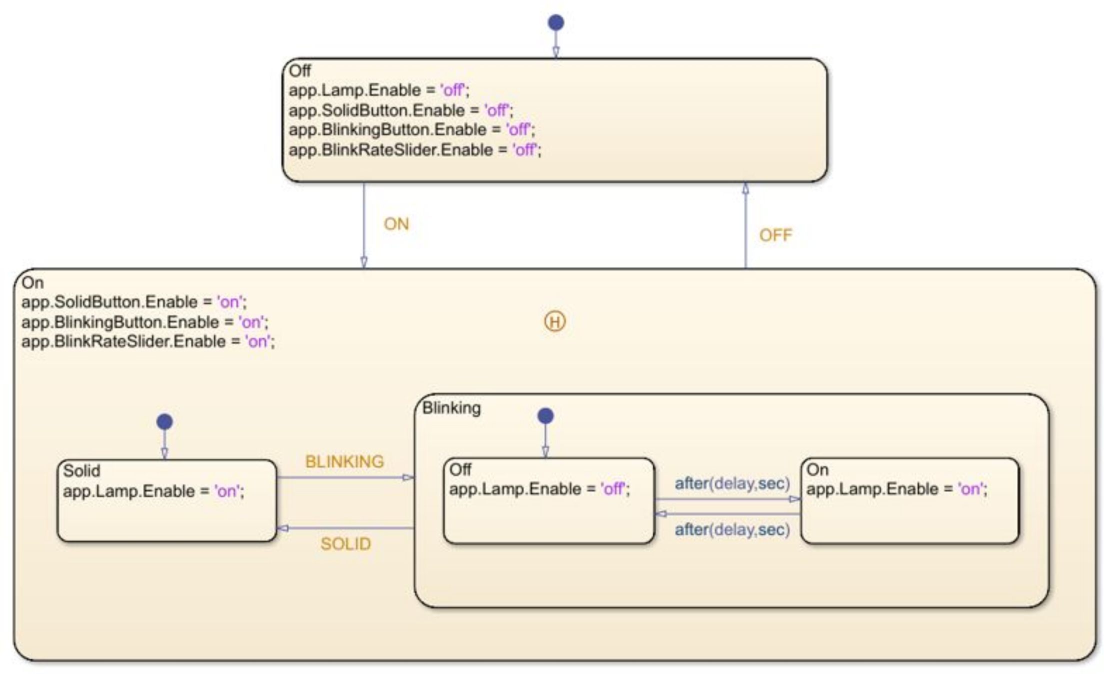




Stateflow Design

State Machines

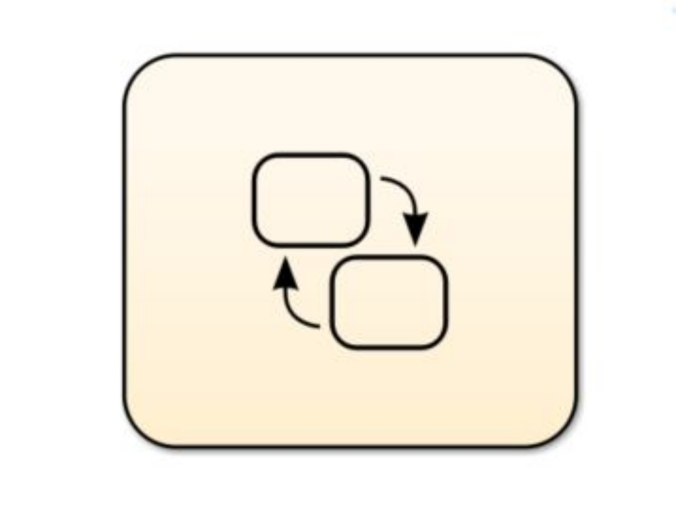
- History Junction



Stateflow Design

State Machines

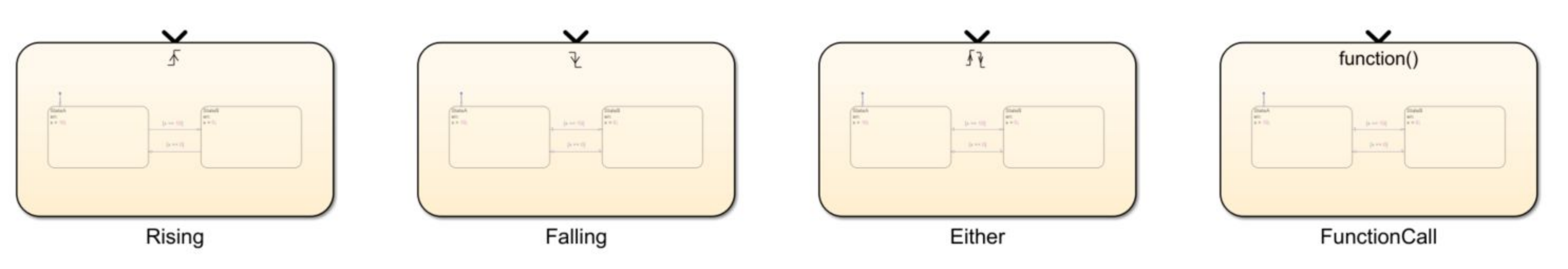
- Synchronous chart
  - Called with the surrounding simulink blocks
  - Contains no visible trigger on top
  - provides the implicit ’tick’ event
    - Invisible
    - Active at every chart call



Stateflow Design

State Machines

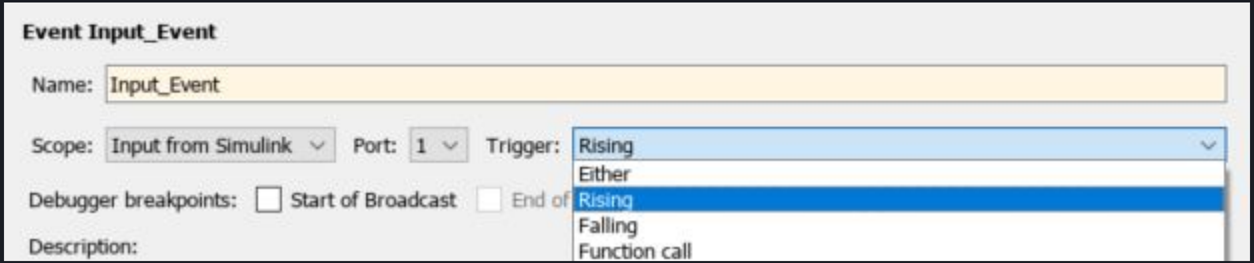
- ASynchronous chart
  - Is called via input triggers (Events):
    - Function call
    - Rising Edge
    - Falling Edge
    - Either Edge

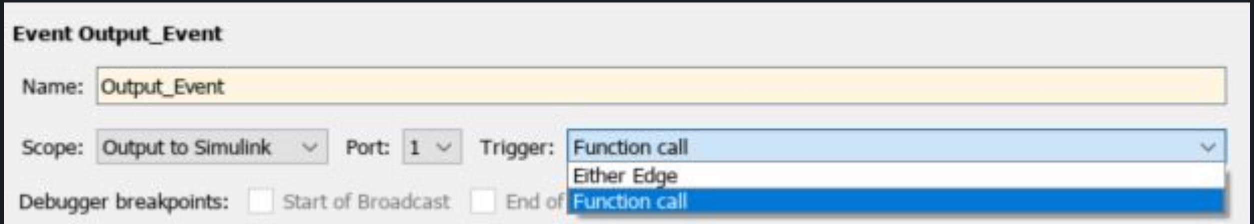


Stateflow Design

State Machines

- Events
  - Inputs
    - Function Call
    - Rising Edge
    - Falling Edge
    - Either Edge
  - Output
    - Function Call
    - Either Edge
  - Local



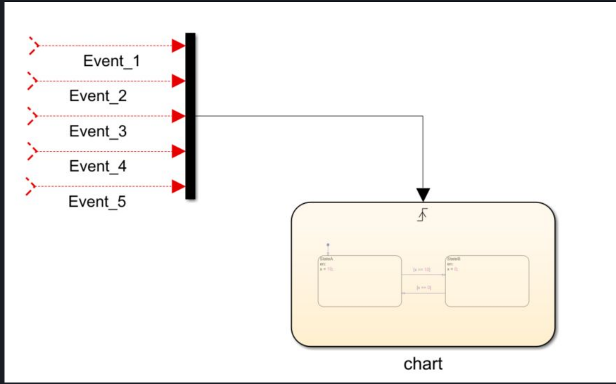


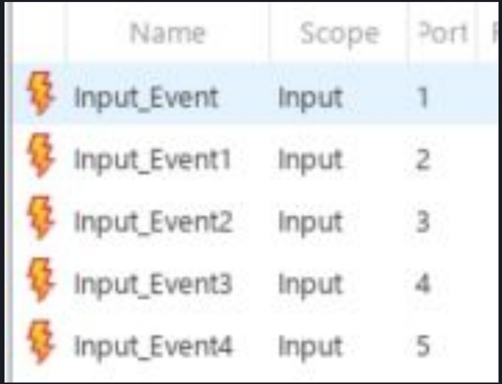
  - Local Event work as function calls
  - Events shall be named to fit their purpose

Stateflow Design

State Machines

- Events
  - Multiple Input events
    - Create MUX that combines the external events
    - create the same number of stateflow input events as mixed external events
    - ensure that the event order of the mux and the input events is identical
    - every active input event will trigger the chart to execute





Name	Scope	Port
Input_Event	Input	1
Input_Event1	Input	2
Input_Event2	Input	3
Input_Event3	Input	4
Input_Event4	Input	5

Stateflow Design

State Machines

- Events
  - Local Events Broadcast
    - Applies to local events
    - syntax: send(Event_Name, Destination_State)
  - External Event Broadcast
    - Applies to output events
    - Need to be un-directed
    - syntax: send(Event_Name)

```
send(LocalEvent,SuperState2)
```

Stateflow Design

State Machines

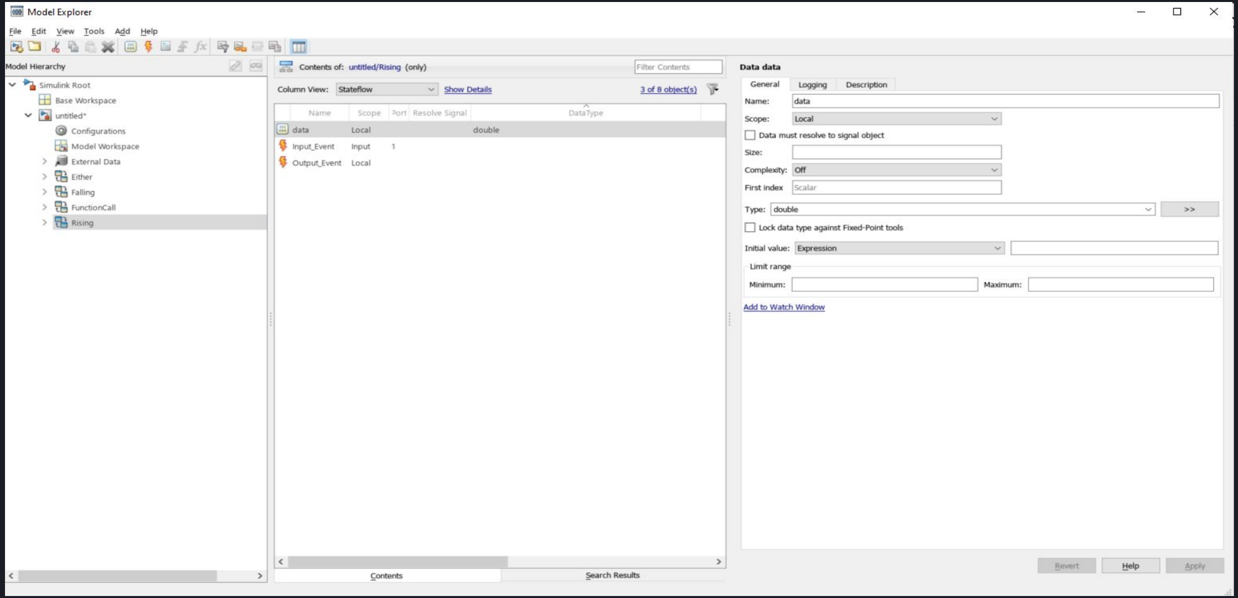
- Data
  - Scopes
    - Input
    - Output
    - Local
    - Parameters
    - Constant
  - Types
    - Inherit data type from simulink
    - double, single
    - int32 / int16 / int8
    - uint32 /uint16 / uint8
    - boolean
    - fixdt()
    - Enum
    - Buses
    - Array

Data objects shall be named to fit their purpose

Stateflow Design

State Machines

- Adding Events & Data
  - Select chart in the model explorer
  - Add Events & Data

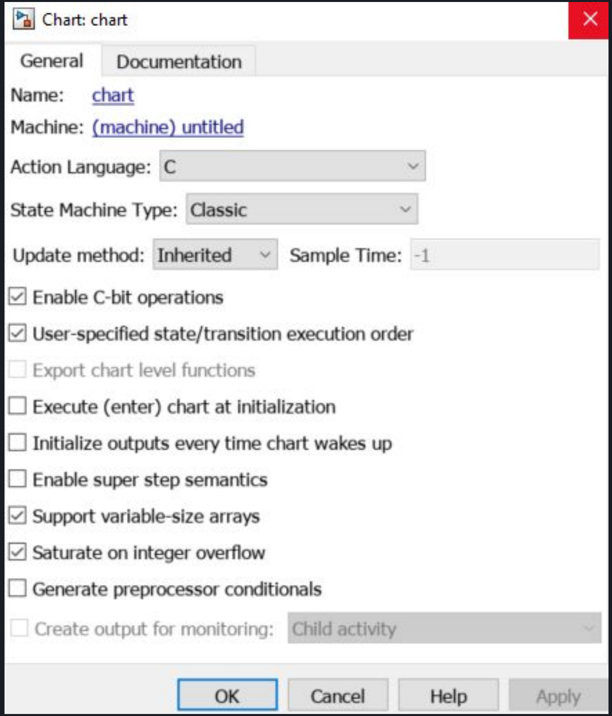


Model-Based Development Program

Stateflow Design

State Machines

- Operators
  - Arithmetic Operators
  - Logical Operators
  - Comparison Operators
  - Unary Operators
  - Unary Actions
  - Bitwise Operators
  - Bitwise Unary Operators

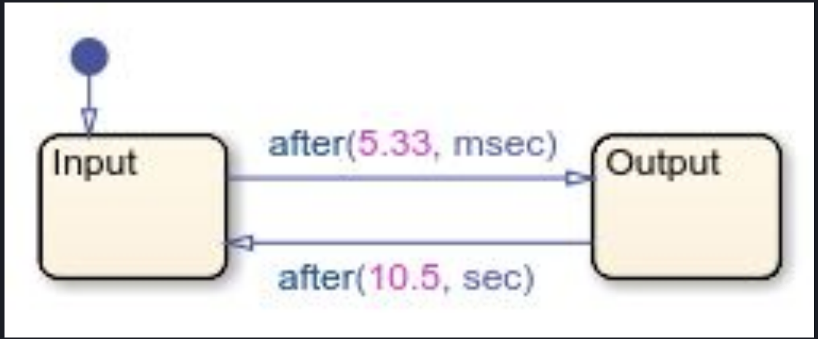


Enable C-bit Operation are required

Stateflow Design

State Machines

- Operators
  - Temporal Operators
    - before() : validate a given positive number of events
    - at() : validate a given positive number of events
    - after(): validate a given positive number of events
    - every(): validate a given positive number of events

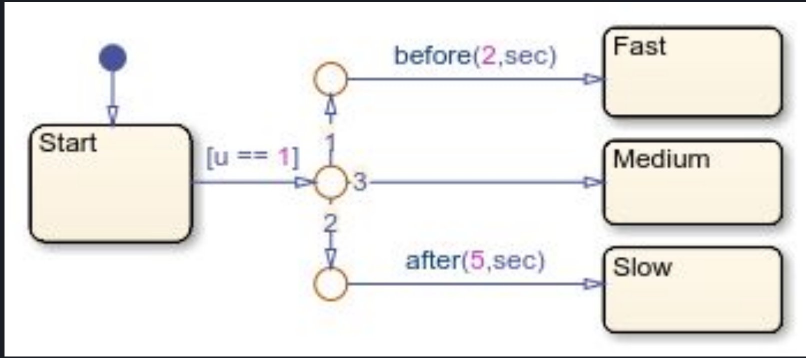


Stateflow Design

State Machines

- Operators
  - Temporal Operators
    - before() : validate a given positive number of events
    - at() : validate a given positive number of events
    - after(): validate a given positive number of events
    - every(): validate a given positive number of events

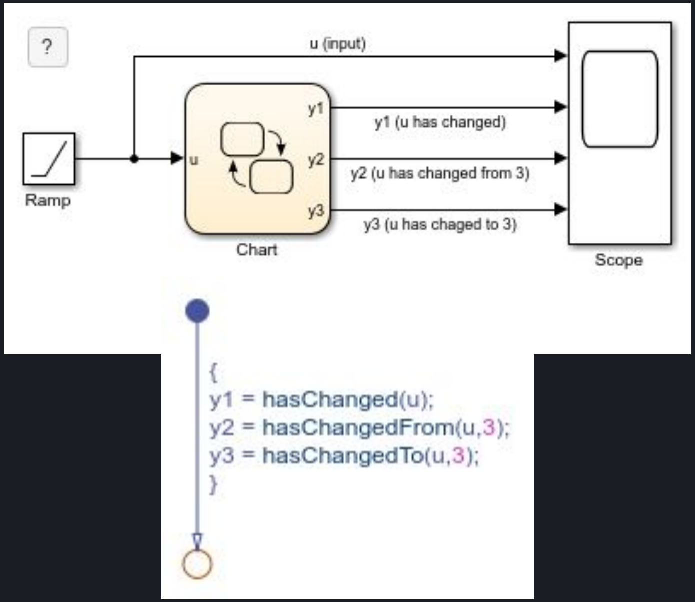
- If the input equals 1 before t = 2 seconds, a transition occurs from `Start` to `Fast`.
- If the input equals 1 between t = 2 and t = 5 seconds, a transition occurs from `Start` to `Medium`.
- If the input equals 1 after t = 5 seconds, a transition occurs from `Start` to `Slow`.



Stateflow Design

State Machines

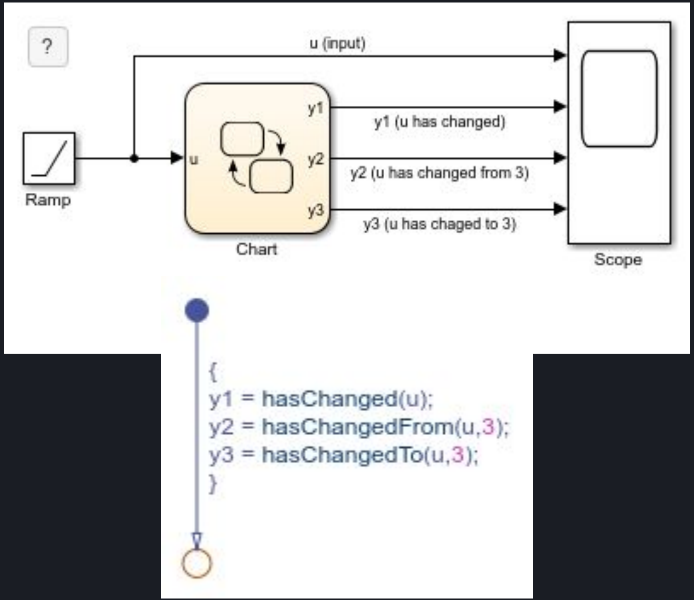
- Operators
  - Change Detection Operators:
    - hasChanged()
      - hasChanged(VAR)
    - hasChangedTo()
      - hasChangedTo(var, TargetVal)
    - hasChangedFrom()
      - hasChangedFrom(var, OriginalVal)

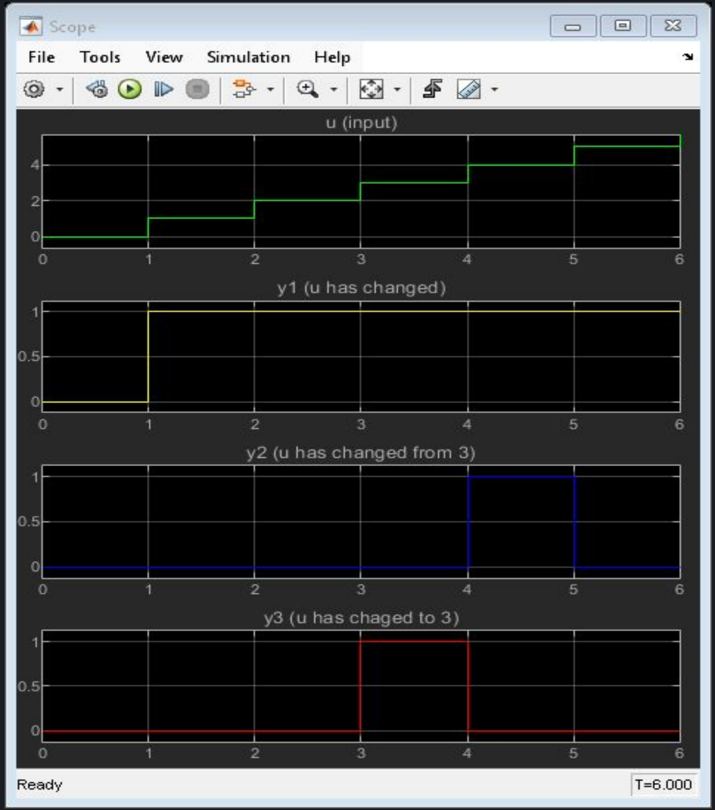


Stateflow Design

State Machines

- Operators





Stateflow Design

State Machines

- Operators
  - Change Detection Operators:
    - hasChanged()
      - hasChanged(VAR)
    - hasChangedTo()
      - hasChangedTo(var, TargetVal)
    - hasChangedFrom()
      - hasChangedFrom(var, OriginalVal)

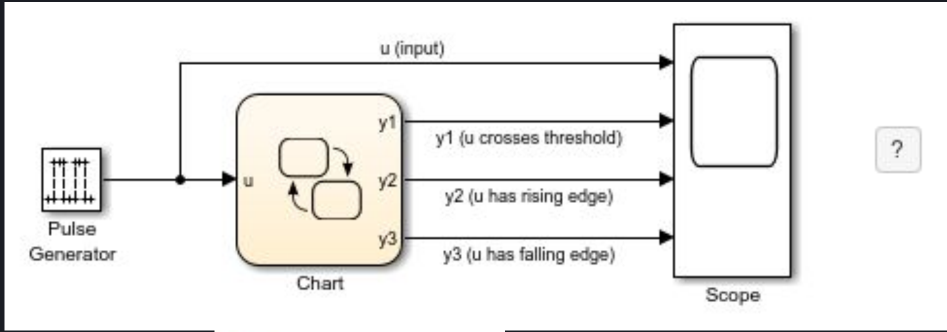
TYPE	NAME	VALUE	PORT
	x	5	1

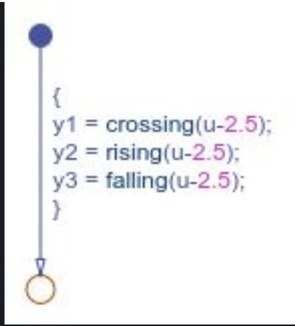
TYPE	NAME	VALUE	PORT
	x	10	1

Stateflow Design

State Machines

- Operators
  - Edge Detection Operators:
    - crossing()
      - crossing(expression)
    - falling()
      - falling(expression)
    - rising()
      - rising(expression)

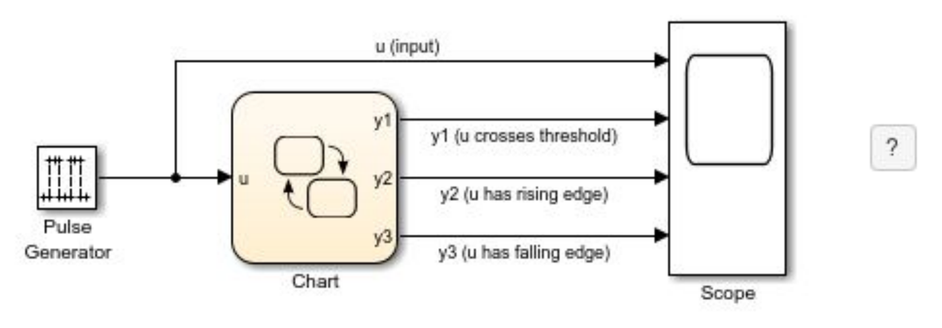


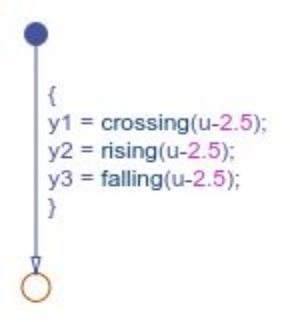


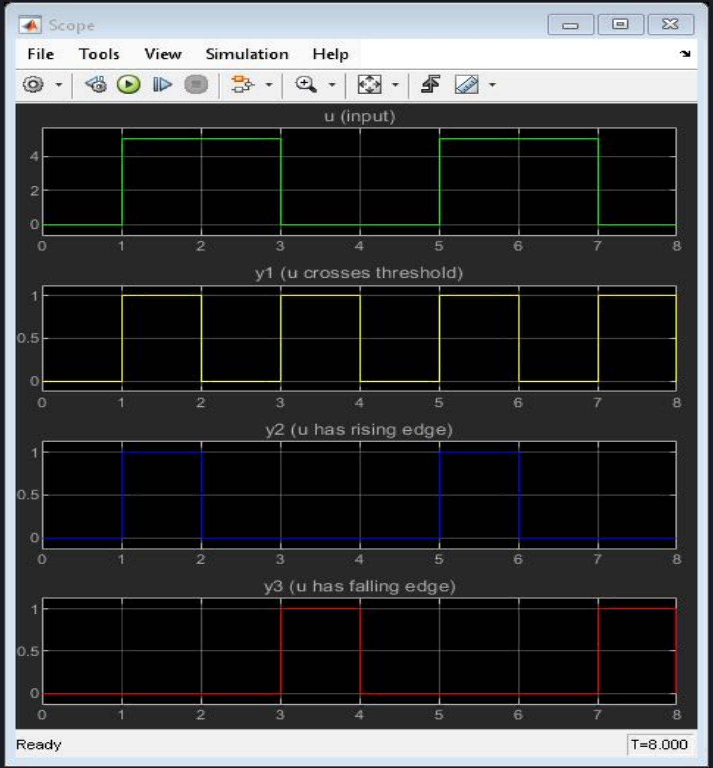
Stateflow Design

State Machines

- Operators





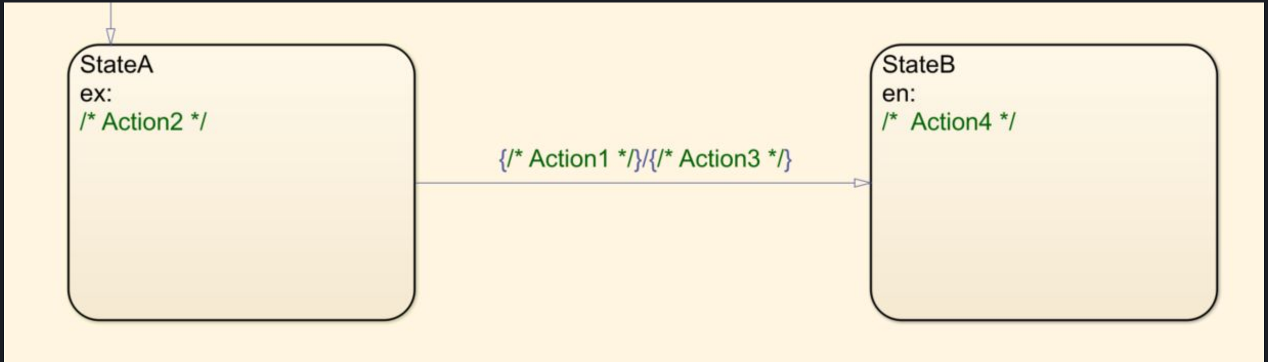


Model-Based Development Program

Stateflow Design

State Machines

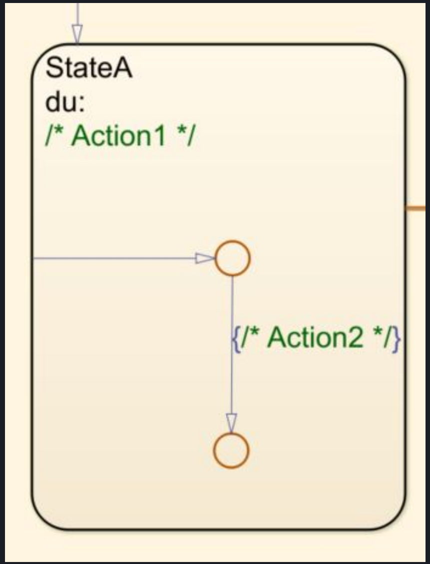
- Action Execution Order
  - State Transitions:
    - Condition Action
    - Exit Action of the state that is exited
    - Transition action
    - Entre action of the state that is entered



Stateflow Design

State Machines

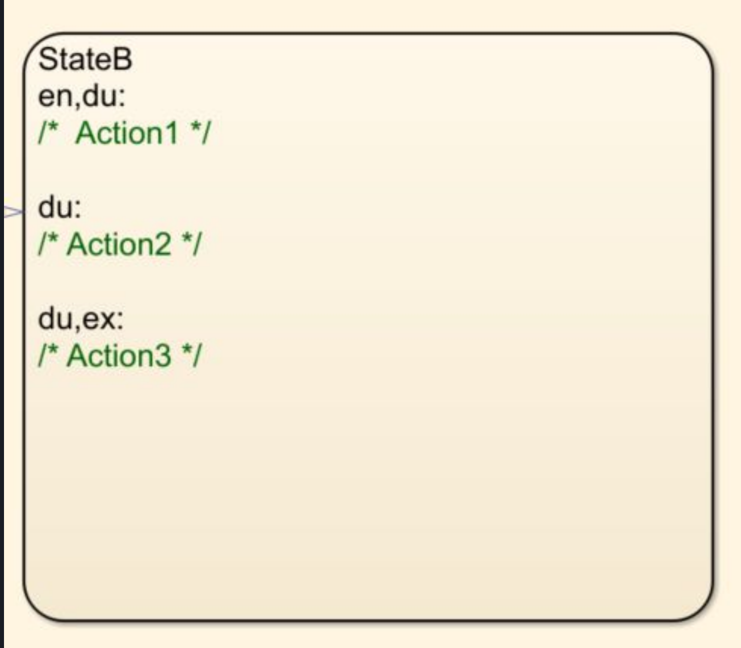
- Action Execution Order
  - State label action , flow graph
    - Label action
    - Flow graph action



Stateflow Design

State Machines

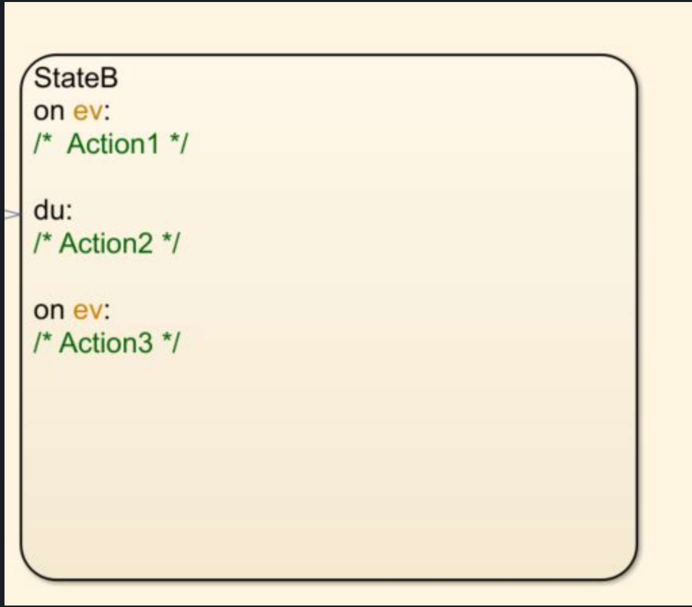
- Action Execution Order
  - State labels with same action
    - The actions are executed in appearance order



Stateflow Design

State Machines

- Action Execution Order
  - during vs. on event
    - The actions are executed in appearance order



Stateflow Design

Flow charts

- what is a flow chart?
  - A flow chart is a graphical construct that models logic patterns by using connective junctions and transitions.
  - the junctions provide decision branches between alternate transition paths.
  - you can use flow charts to represent decision and iterative loop logic.

- When to use flow chart?
  - use flow charts to represent combinatorial logic in graphical functions or between states in a chart
  - a best practice is to encapsulate flow charts in graphical functions to create modular, reusable decision and loop logic that you can call anywhere in a chart.
  - complex state logic in state actions; except exit

Stateflow Design

Flow charts

- Basic Rules
  - Conditions are drawn horizontally, with the exception of loop patterns
    - The horizontal conditional path shall always have priority “1”.
  - condition actions are drawn vertically with the exception of loop patterns.
  - Right-to-left horizontal transitions must be blank label or comment label only.
  - junctions shall have an unconditional path.
  - Only one default transition
  - Only one terminating point
  - No transition actions

Stateflow Design

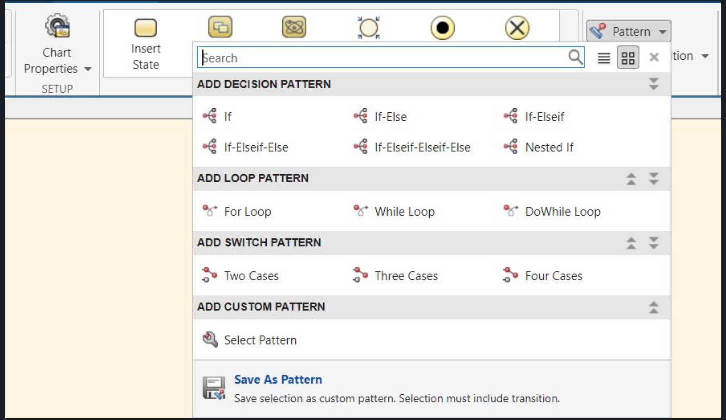
Flow charts

- Flow Patterns
    - To properly design decision flows
    - standardized patterns exist
    - easily understandable and maintainable flow charts can be created.
    - complex logic can be reduced to the basic patterns
    - a good practice is to separate consecutive patterns.

Stateflow Design

Flow charts

- Pattern Generator
  - A generator for design patterns that can be found in the chart menu.

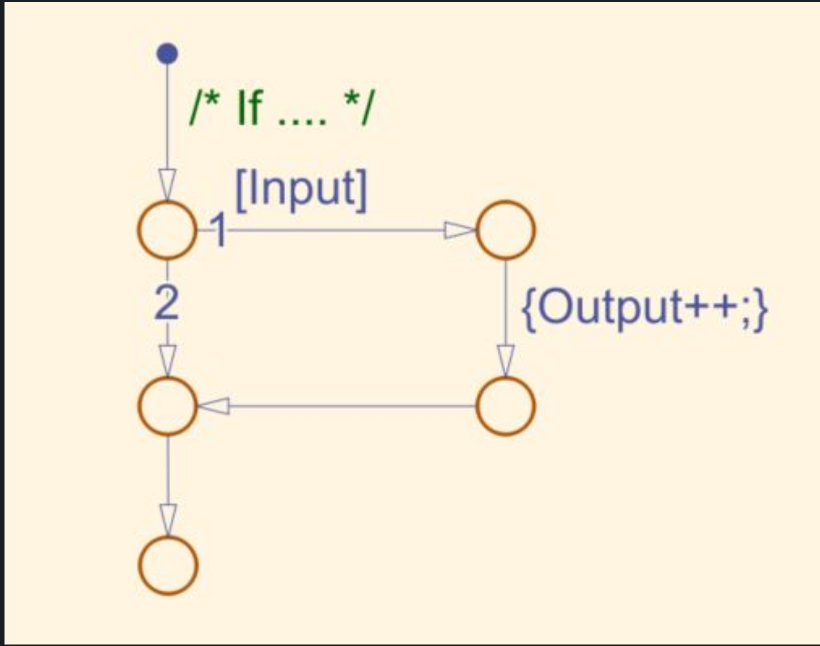


Stateflow Design

Flow charts

- Exercises
  - if-end

```
if (Input)
{
    Output++;
}
```

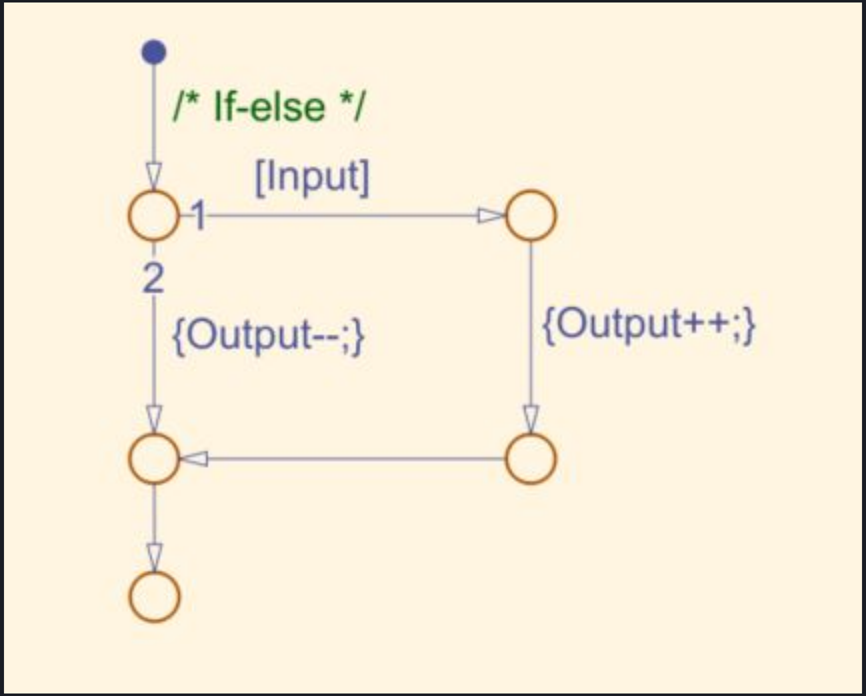


Stateflow Design

Flow charts

- Exercises
  - if-else

```
if (Input)
{
      Output++;
}
else
{
      Output--;
}
```

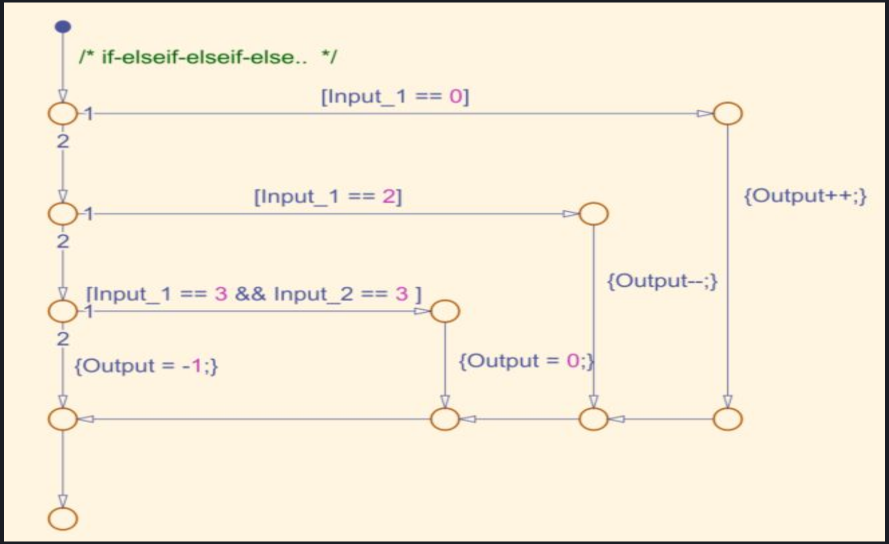


Stateflow Design

Flow charts

- Exercises
  - if-elseif-elseif-else..

```
if (Input_1 == 0)
{
    Output++;
}
else if (Input_1 == 2)
{
    Output--;
}
else if (Input_1 == 3 && Input_2 == 3)
{
    Output = 0;
}
else
{
    Output = -1;
}
```



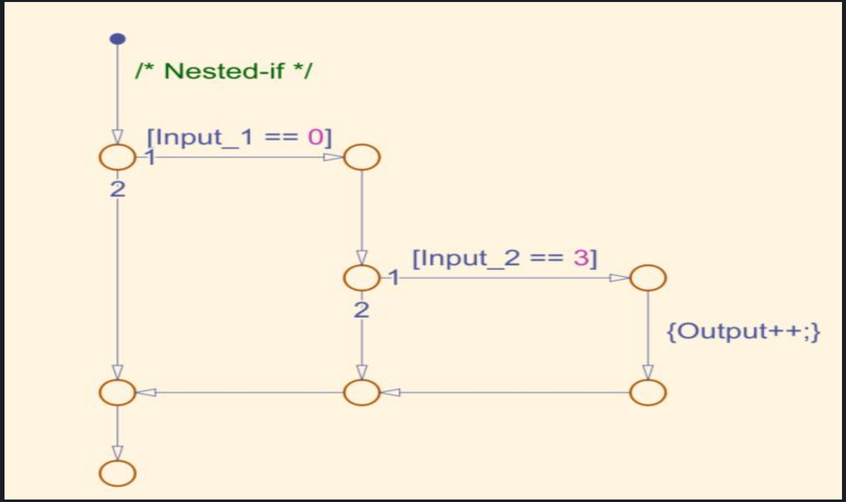
Stateflow Design

Flow charts

- Exercises
    - Nested-if

```
if (Input_1 == 0)
{

    if(Input_2 == 3)
    {
        Output++;
    }
}
```

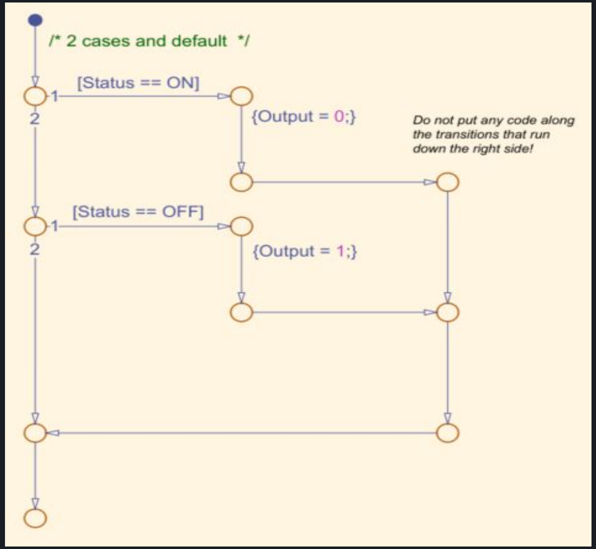


Stateflow Design

Flow charts

- Exercises
  - switch case

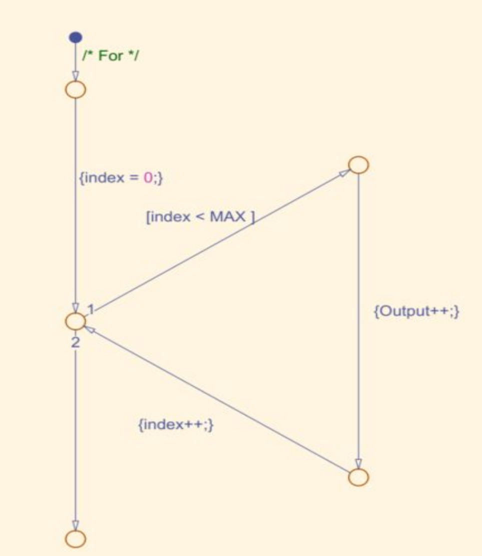
```
switch (Status)
{
    case ON:
    {
        Output = 0;
        break;
    }
    case OFF:
    {
        Output = 1;
        break;
    }
    default
    {
        break;
    }
}
```



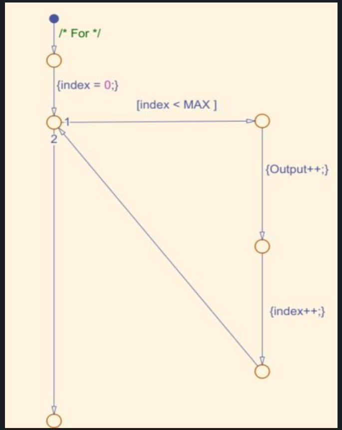
Stateflow Design

Flow charts

- Exercises
  - for loop




```
for (index = 0; index < MAX; index++)
{
    Output++;
}
```

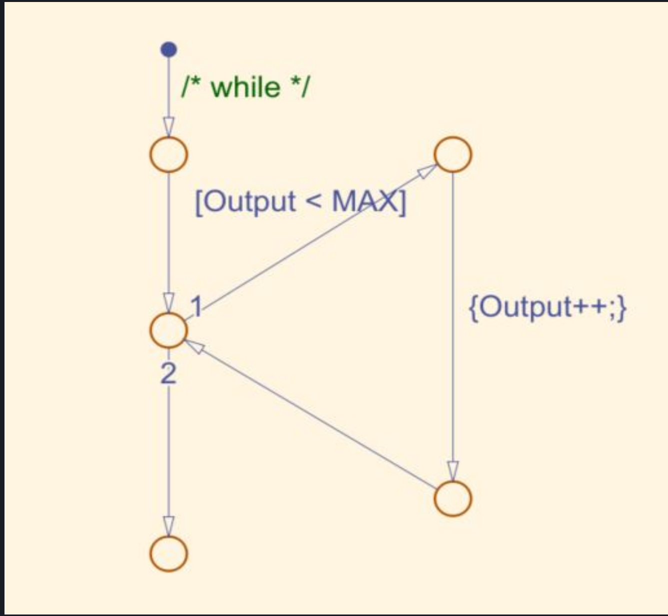


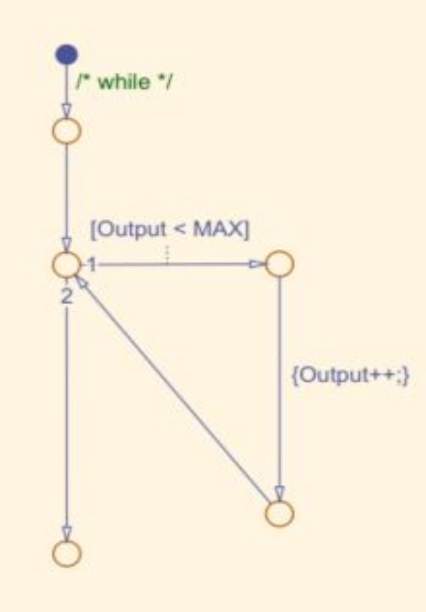
Stateflow Design

Flow charts

- Exercises
  - while loop

```
while (Output < MAX)
{
    Output++;
}
```



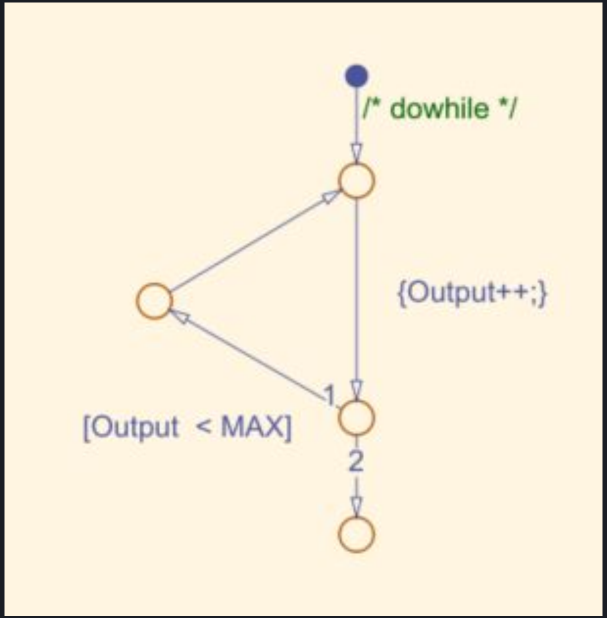


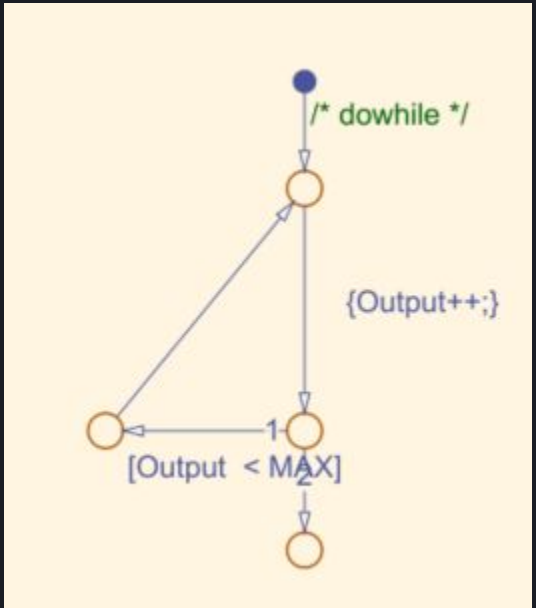
Stateflow Design

Flow charts

- Exercises
  - do-while loop

```
do
{
    Output++;
}
while (Output < MAX)
```



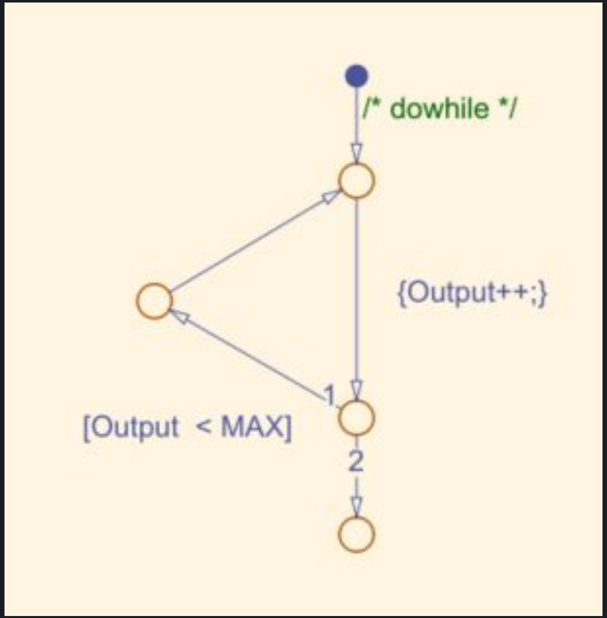


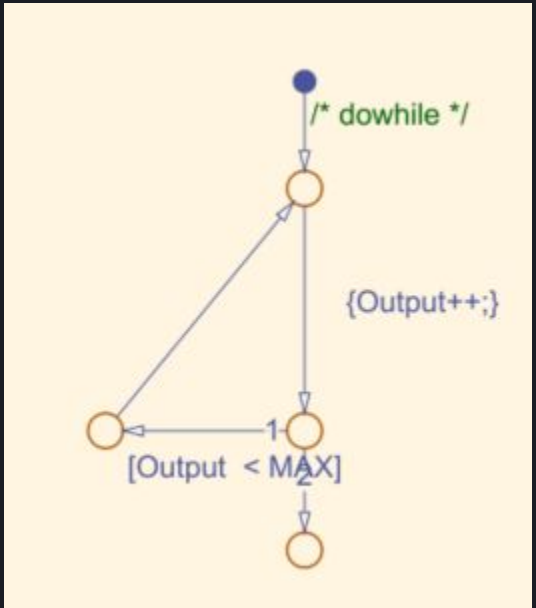
Stateflow Design

Flow charts

- Exercises
  - do-while loop

```
do
{
      Output++;
}
while (Output < MAX)
```

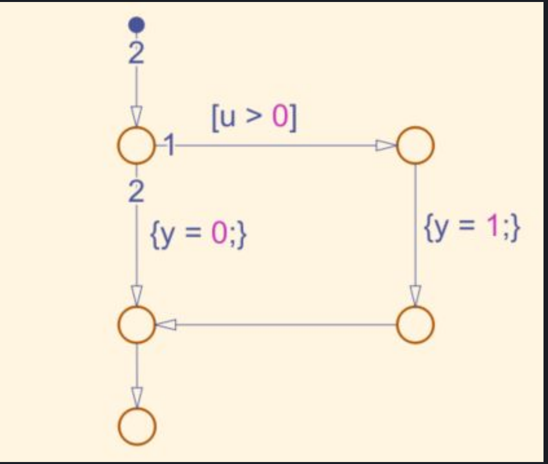




Stateflow Design

Exercise: Accelerator Pedal Position Sensor

- A flow graph is a graphical construct that models logic patterns by using connective junctions and transitions.
- The junctions provide decision branches between alternate transition paths.
- Here is an example of a flow graph that models simple if-else logic:
- it is required to implement a decision logic to determine
    - APP_bSensor1Failure
    - APP_bSensor2Failure
    - APP_bCoherencyFailure



Stateflow Design

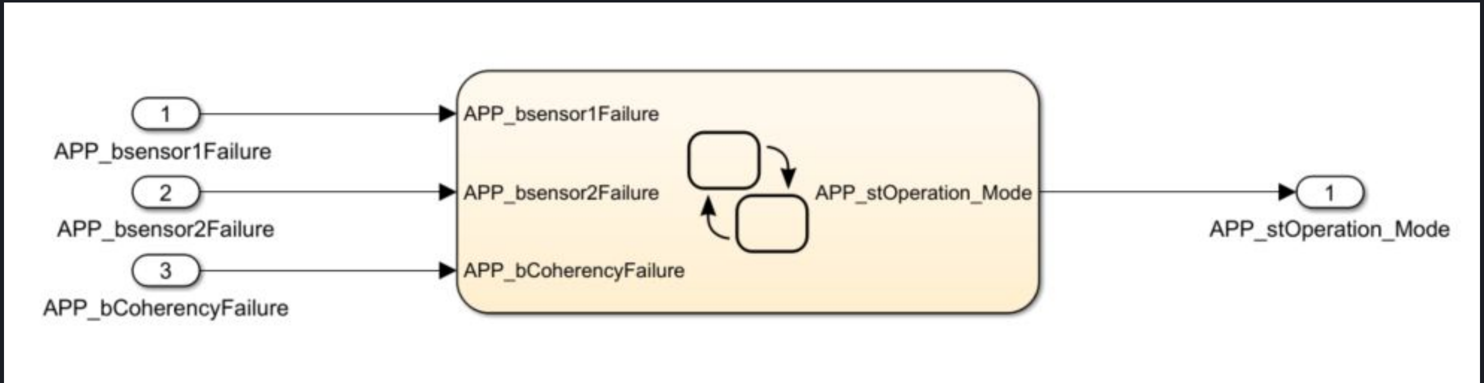
Exercise: Accelerator Pedal Position Sensor

- The reading of the accelerator pedal position is safety critical; the system can not operate normally in case there is a failure in thai reading. in one design, the system has four modes of operation
  - Normal Mode
    - No electrical failures and exit and coherency check is OK.
  - Downgraded mode Sensor 1 only
    - there is an electrical failure in sensor 2 and hence only sensor 1 will be used
  - Downgraded mode Sensor 2 only
    - there is an electrical failure in sensor 1 and hence only sensor 2 will be used
  - Failure mode
    - electrical failure in both sensors or there is a failure in coherency check

Stateflow Design

Exercise: Accelerator Pedal Position Sensor

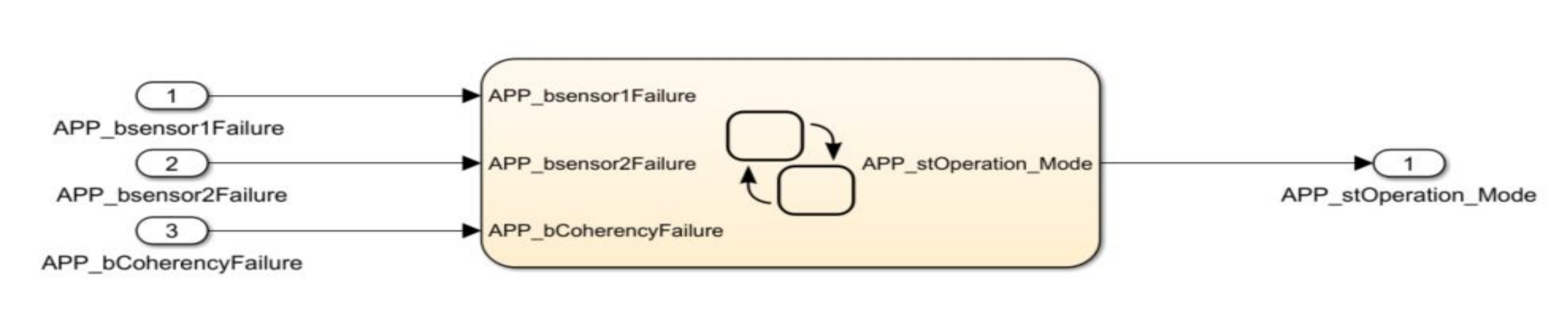
- it is required to model the modes of operation
  - Normal mode
  - Downgraded mode sen 1
  - Downgraded mode sen 2
  - Failure mode



Stateflow Design

Exercise: Accelerator Pedal Position Sensor

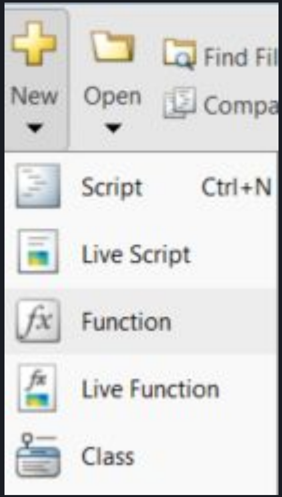
- Defining Inputs and Outputs



Stateflow Design

Exercise: Accelerator Pedal Position Sensor

- In the MATLAB Command Window, select Home > New > Class
- define enumerated values in an enumeration section



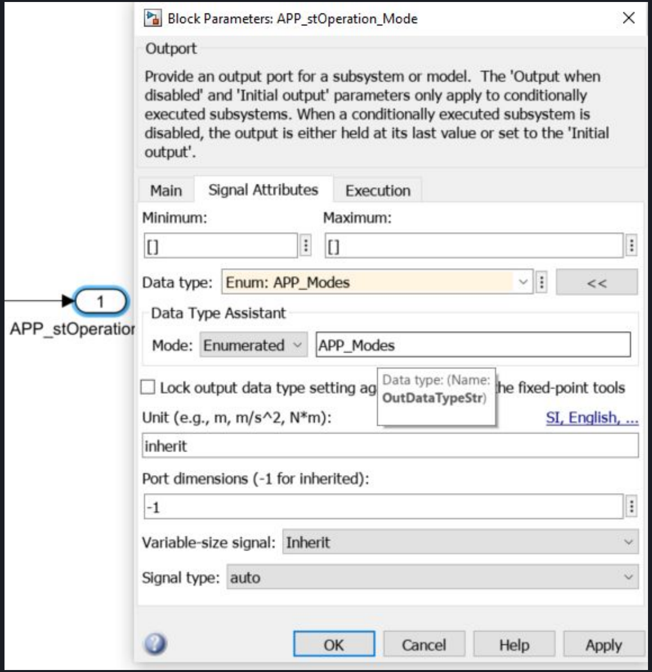
APP_Modes.m

```
classdef (Enumeration) APP_Modes < Simulink.IntEnumType
    enumeration
        Normal(0)
        Downgraded_Sen_1(1)
        Downgraded_Sen_2(2)
        Failure(3)
    end
end
```

Stateflow Design

Exercise: Accelerator Pedal Position Sensor

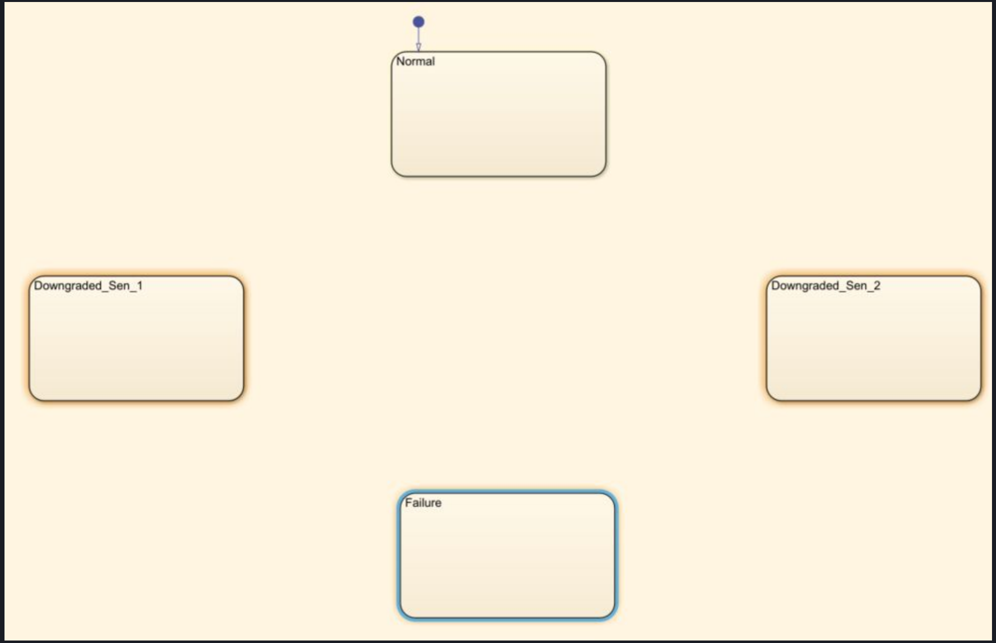
- In the MATLAB Command Window, select Home > New > Class
- define enumerated values in an enumeration section
- Add Enumerated data to a chart



Stateflow Design

Exercise: Accelerator Pedal Position Sensor

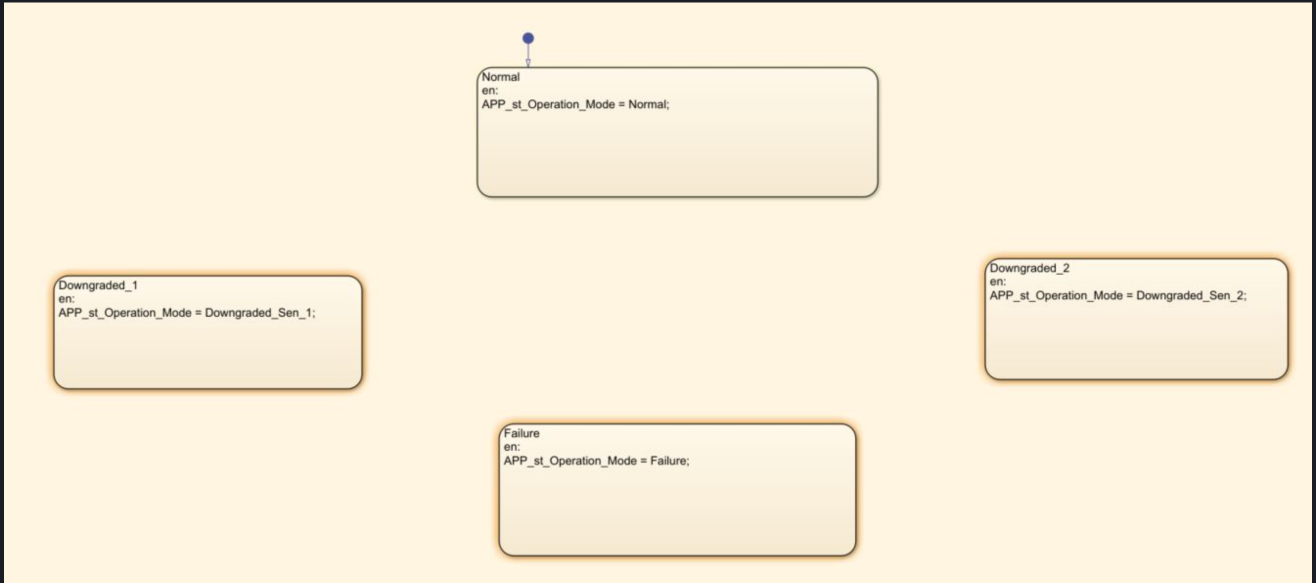
- Defining States



Stateflow Design

Exercise: Accelerator Pedal Position Sensor

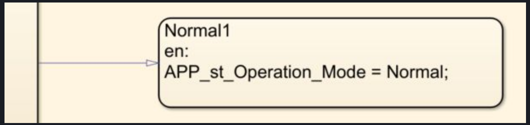
- Defining State Actions

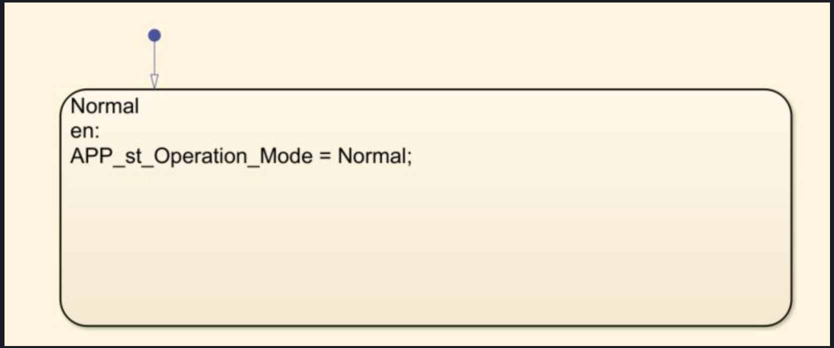


Stateflow Design

Exercise: Accelerator Pedal Position Sensor

- A transition is a line with an arrowhead that linked on graphical object to another.
- Defining default transition
  - Default transition specifies which exclusive (OR) state to enter
  - A common programming mistake is to create multiple exclusive (OR) without a default transition
    - in the absence of the default transition, there is no indication of which state becomes active by default

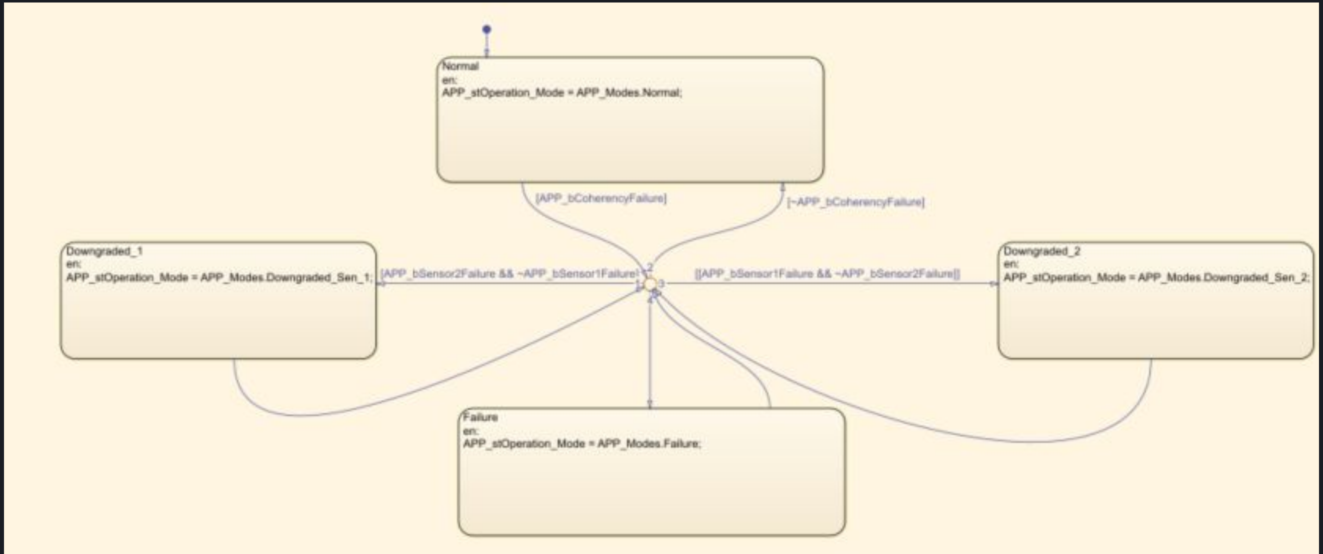




Stateflow Design

Exercise: Accelerator Pedal Position Sensor

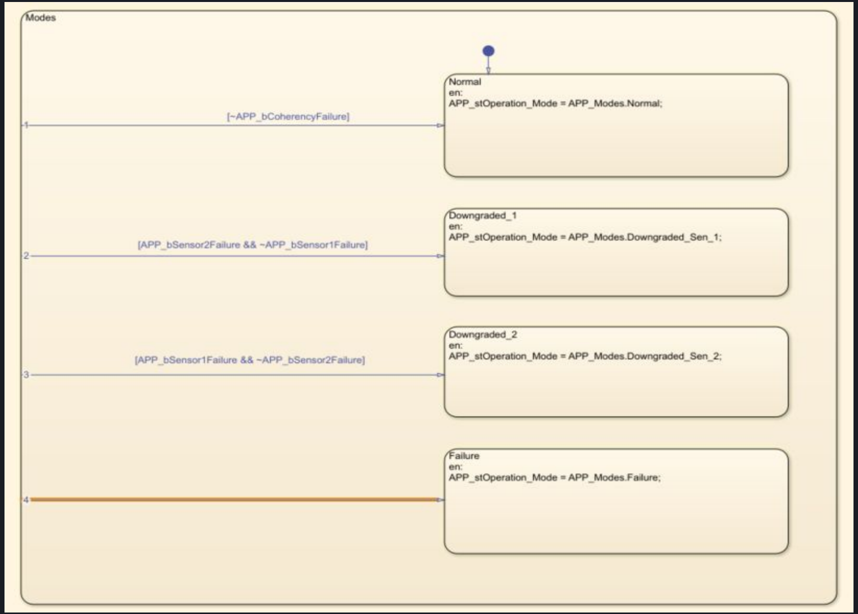
- Implement transitions between states to satisfy system requirements



Stateflow Design

Exercise: Accelerator Pedal Position Sensor

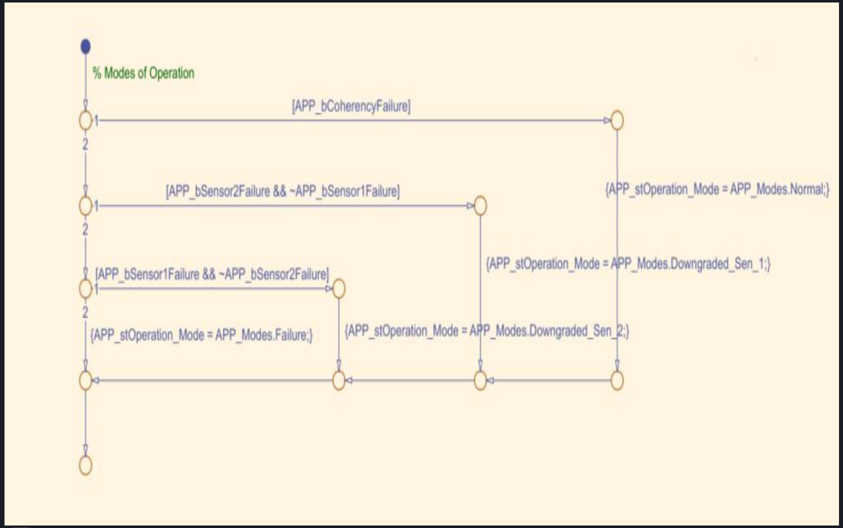
- Implement transitions between states to satisfy system requirements

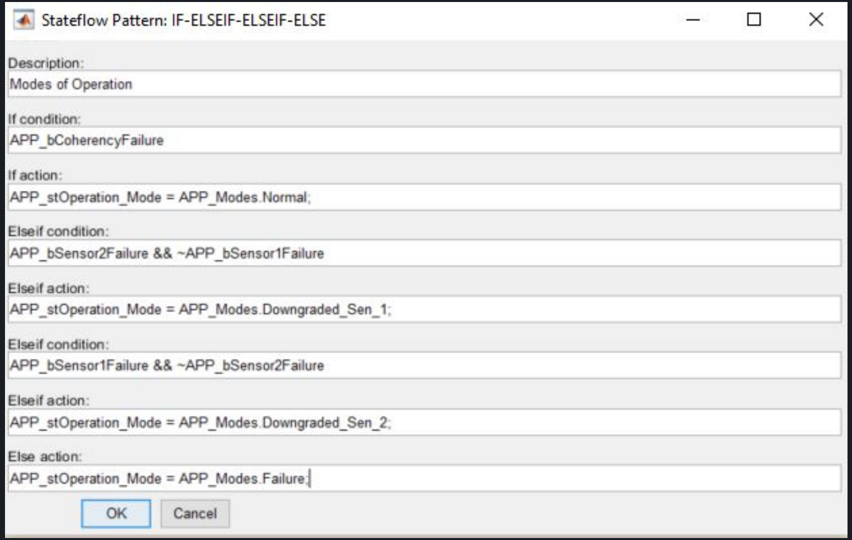


Stateflow Design

Exercise: Accelerator Pedal Position Sensor

- Although the example deals with “modes of operation”, it does not require”model logic” as the calculation does not depend on history of previous states/ modes, it just depend on inputs variables. hence, the example can be designed using “flow-graphs” (stateless)

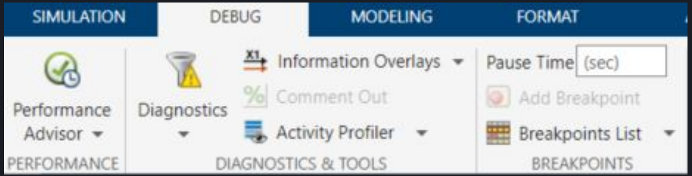


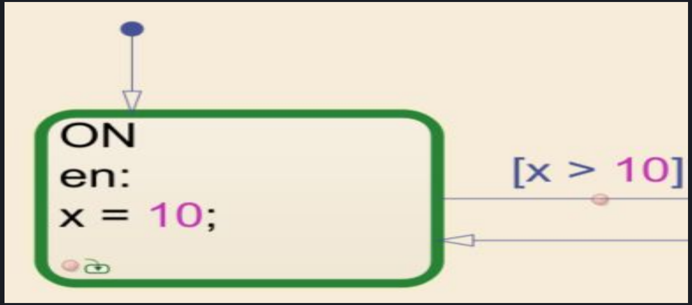


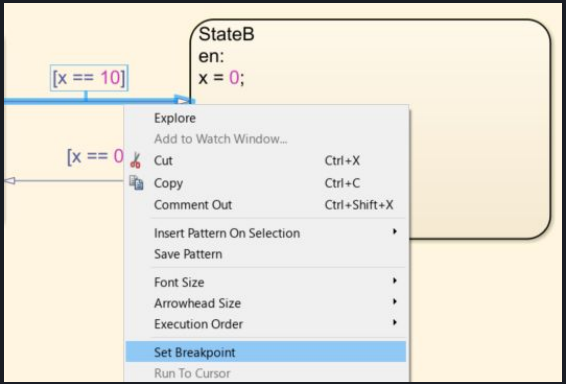
Stateflow Design

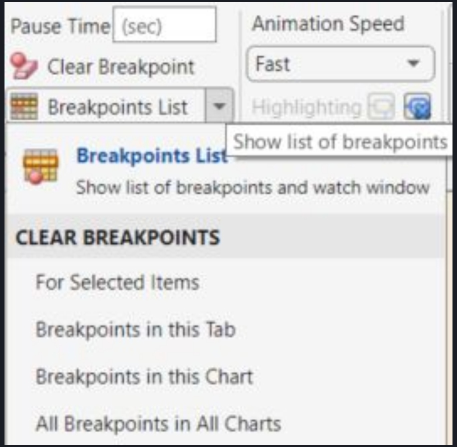
Running Simulations and Debug

- Stateflow supports breakpoints:
  - Select a Transition, a state, a Truth table or a Graphical Function
  - Set Breakpoints:
    - Right Click on the object
  - Clear Breakpoints
- Active object are highlighted in green









Stateflow Design

Running Simulations and Debug

- Standard debug steps are now available
  - Step Over
  - Step In
  - Step Out

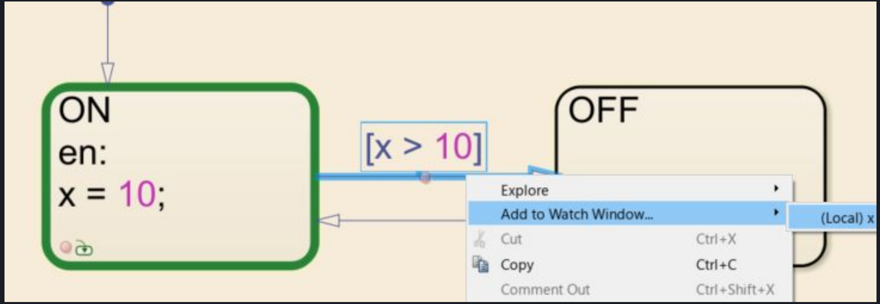
- An execution status badge appears in the graphical object where execution pauses.

Badge	Description
	Simulation is paused before entering a chart or in a state `entry` action.
	Simulation is paused in a state `during` action, graphical function, or truth table function.
	Simulation is paused in a state `exit` action.
	Simulation is paused before testing a transition.
	Simulation is paused before taking a valid transition.

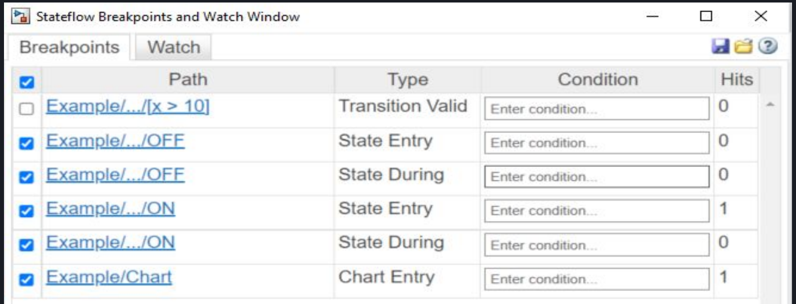
Stateflow Design

Running Simulations and Debug

- Watches



- Better Breakpoint handling:
  - Every Breakpoint can be activated and deactivated at will
  - Conditional Breakpoints easy to set up
  - Number of Breakpoint activation available



Stateflow Design

Graphical Functions

- What is a Graphical Function?
  - A graphical in a chart is a graphical element that helps you reuse control-flow logic and iterative loops.
  - This function is a program you write with flow charts using connective junctions and transitions.
  - you create a graphical function. fill it with a flow chart, and call the function in the actions of states and/or transitions.

- Why use a graphical function in stateflow chart?
  - Create modular, reusable logic that you can in your chart.
  - track simulation behavior visually during chart animation.

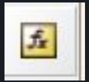
Stateflow Design

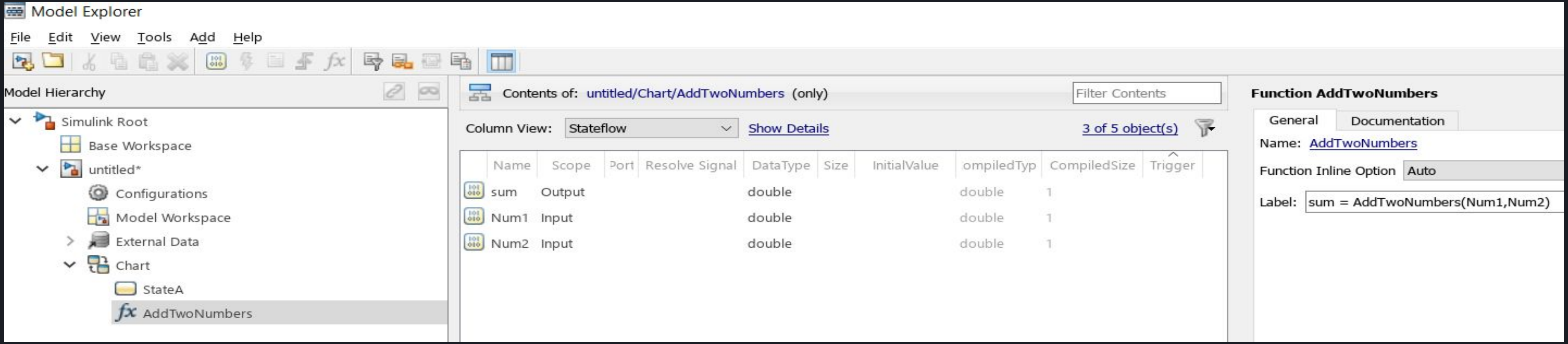
Graphical Functions

- Where to use a graphical function?
  - A graphical function can reside anywhere in a chart, state, or subchart.
  - The location of a function determines its scope, that is, the set of states and transitions that can call the function.
  - Follow these guidelines:
    - If you want to call the function only within one state or subchart and its substates,put your graphical function in that state or subchart. that function overrides any other functions of the same name in the parents and ancestors of that state of subchart.
    - if you want to call the function anywhere in that chart, put your graphical function at the chart level .
    - if you want to call the function from any chart in your model, put your graphical function at the chart level and enable exporting of chart-level graphical functions.

Stateflow Design

Graphical Functions

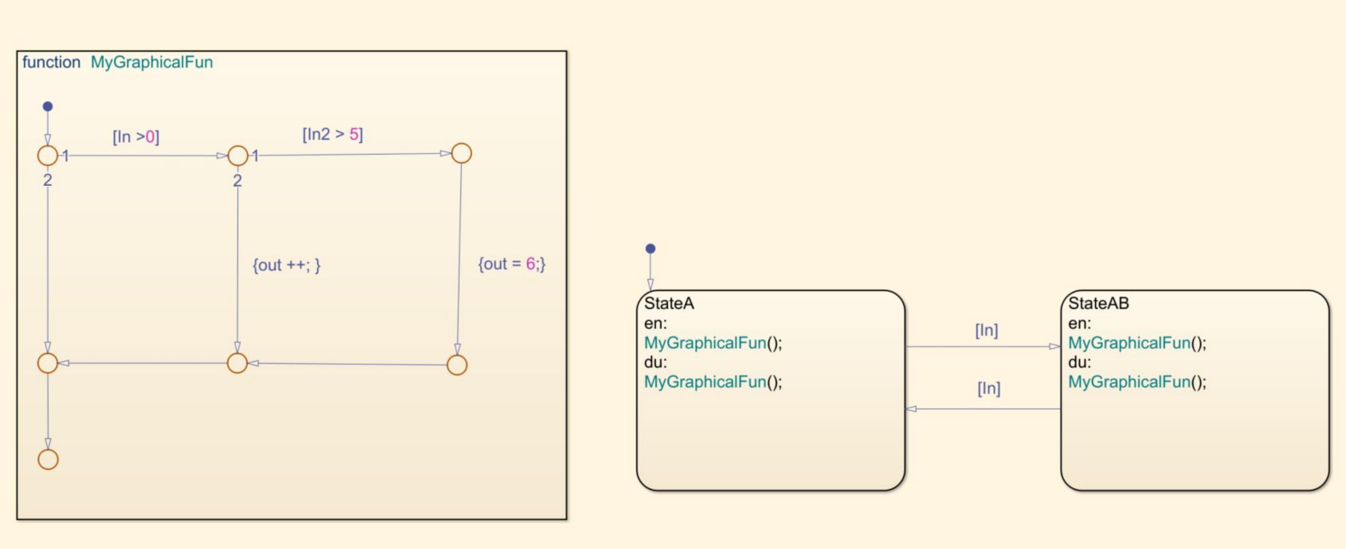
- Re-usable Graphical functions:
  - Stateflow allows to create so-called atomic boxes.
  - The handling is equal to atomic subcharts with the restriction that no state must be present in the chart
- Use the menu bar to add a graphical function 
- The interfaces and variables are added via the model explorer



Stateflow Design

Graphical Functions

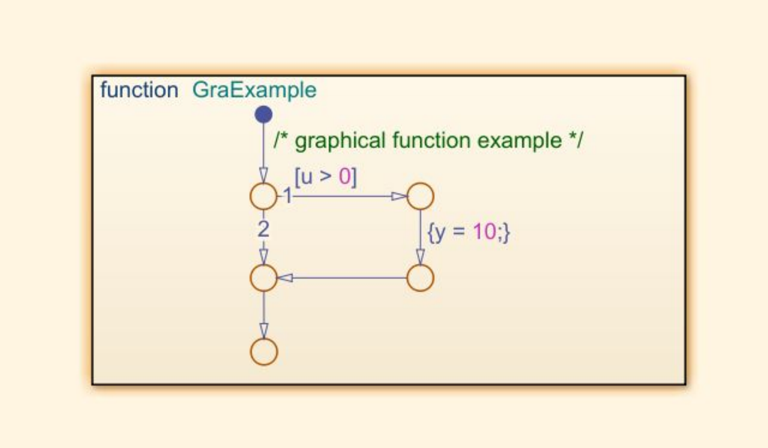
- Example



Stateflow Design

Graphical Functions

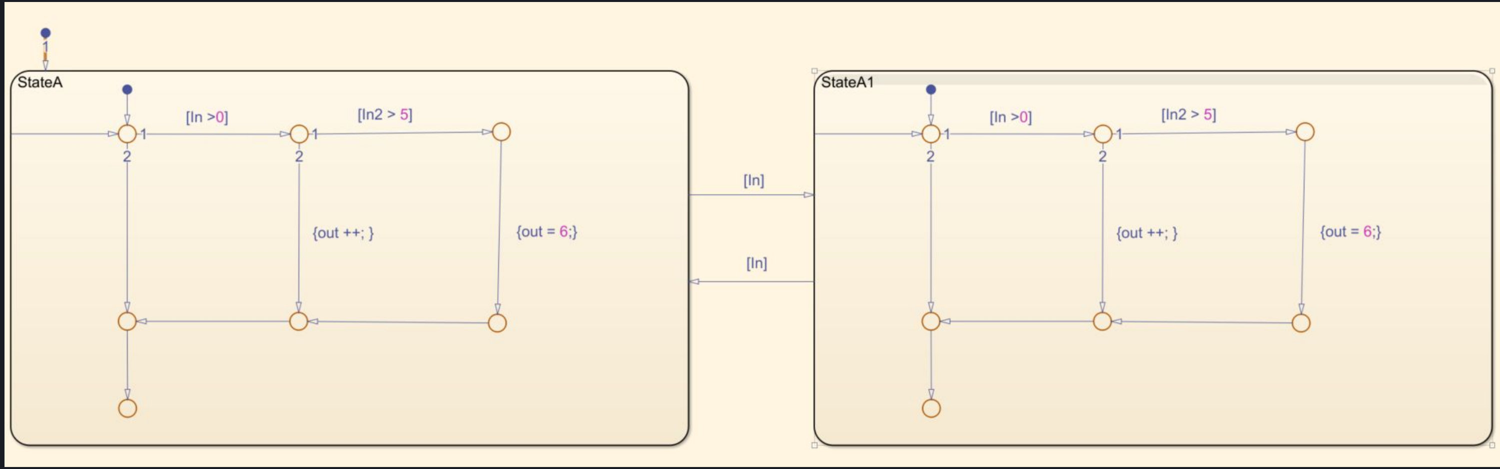
- Advantages
  - Enhance readability by ‘Subchart’ fro graphical functions
    - Ctrl+Shift+G



Stateflow Design

Graphical Functions

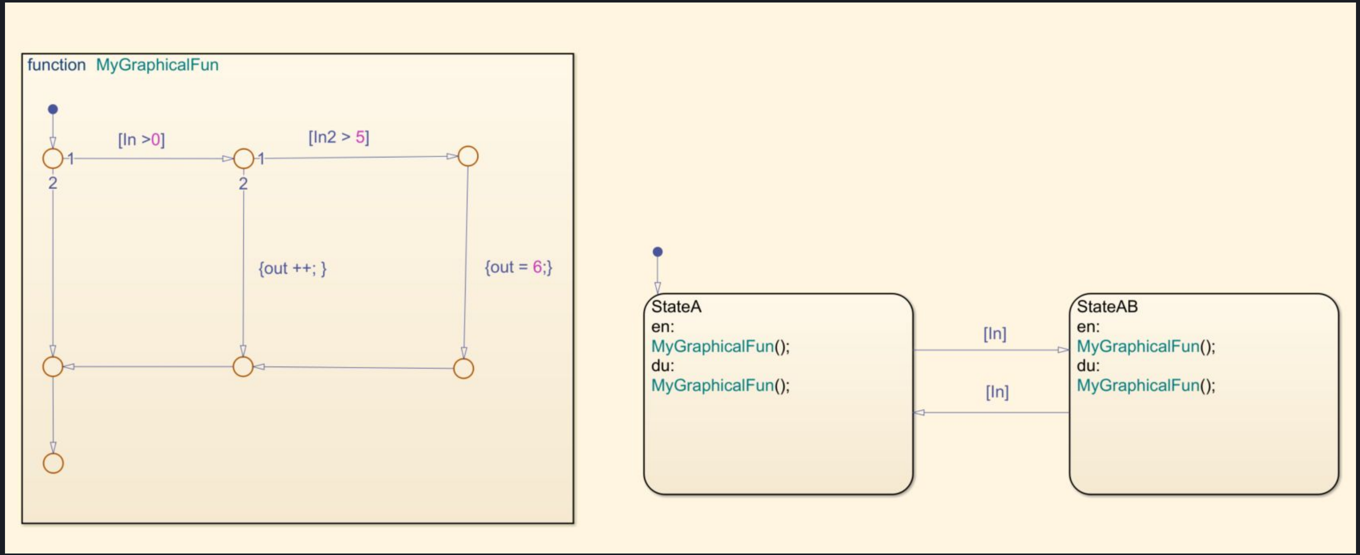
- Advantages
  - Grouping ‘Identical’ State Actions



Stateflow Design

Graphical Functions

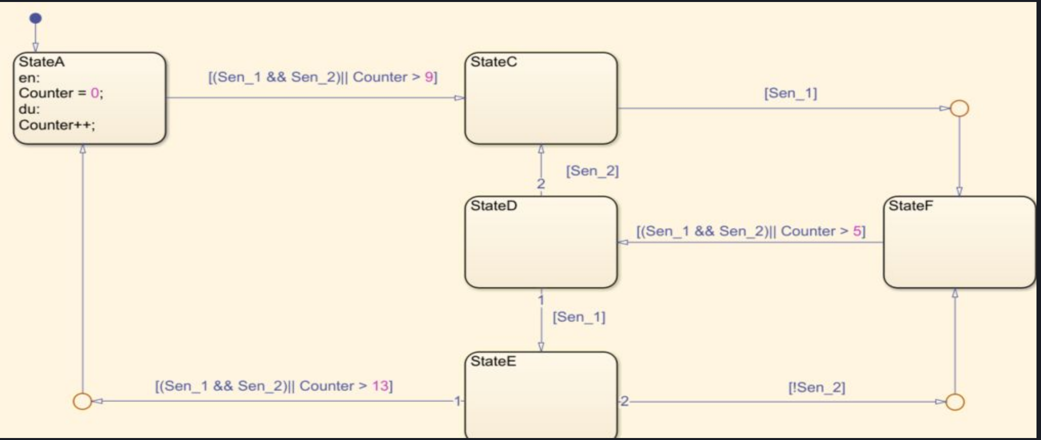
- Advantages
  - Grouping ‘Identical’ State Actions



Stateflow Design

Graphical Functions

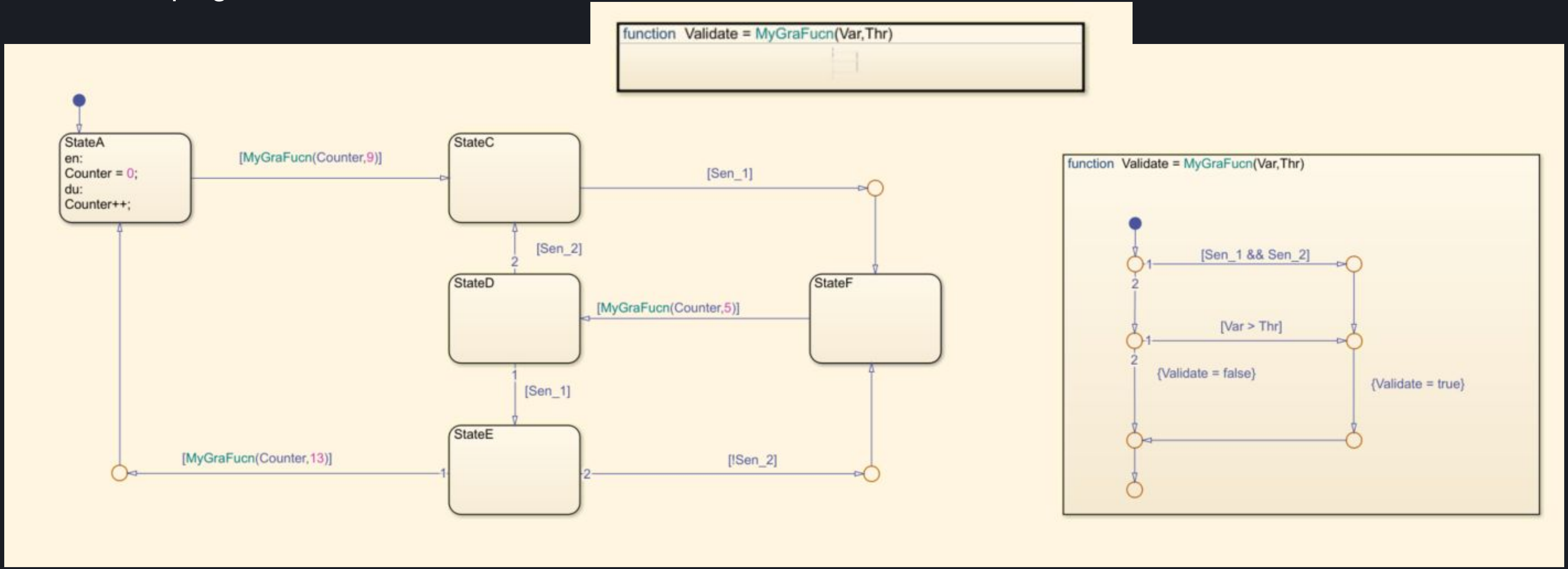
- Advantages
  - Grouping ‘Identical’ State Transitions



Stateflow Design

Graphical Functions

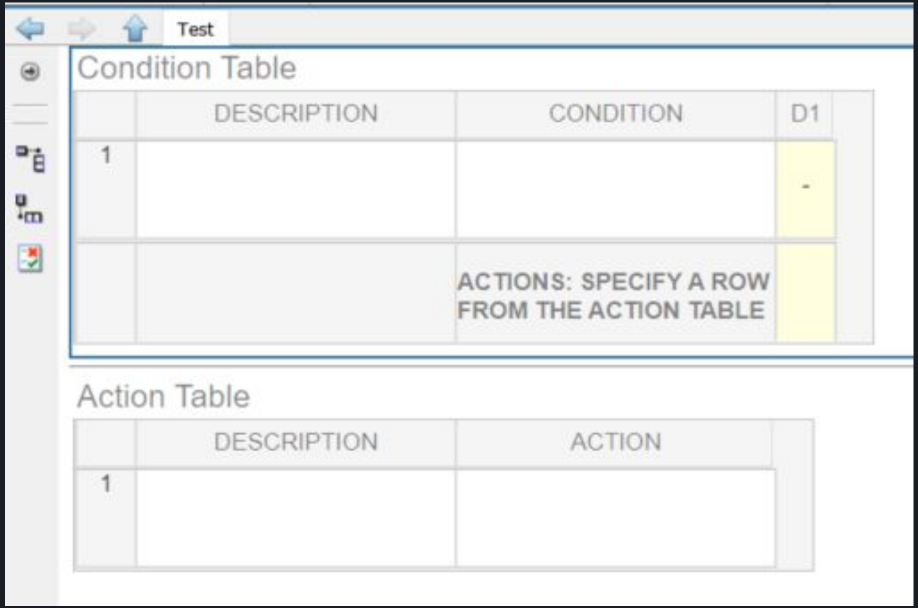
- Advantages
  - Grouping ‘Identical’ State Transitions



Stateflow Design

Truth Table

- Truth table functions implement combinational logic
- Stateflow truth tables contain
  - Condition
  - Decisions
  - Actions

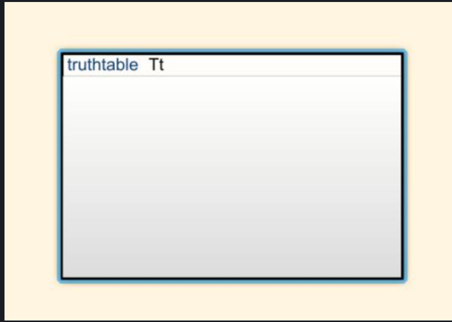


Stateflow Design

Truth Table

- Use the menu Bar to add a Truth table 

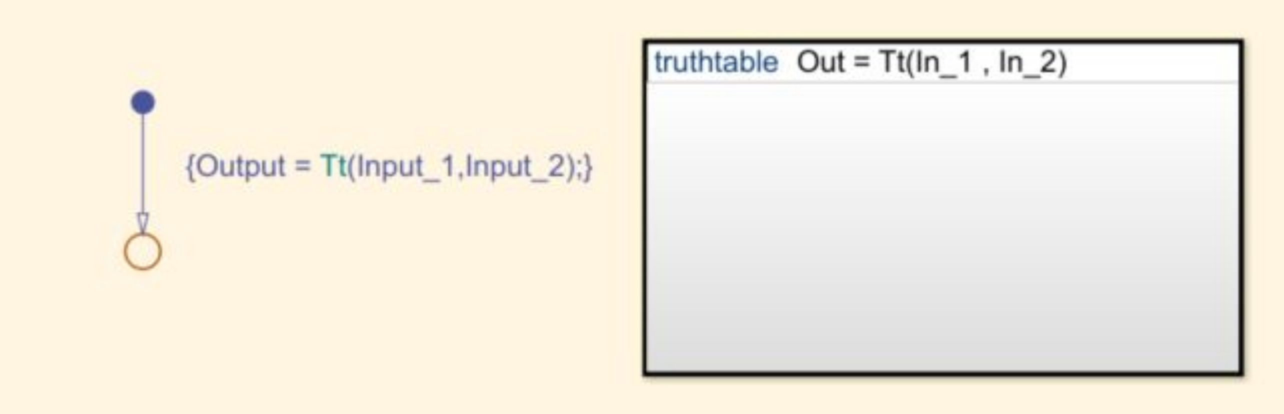
- The interfaces and variables are added via the model explorer



Stateflow Design

Truth Table

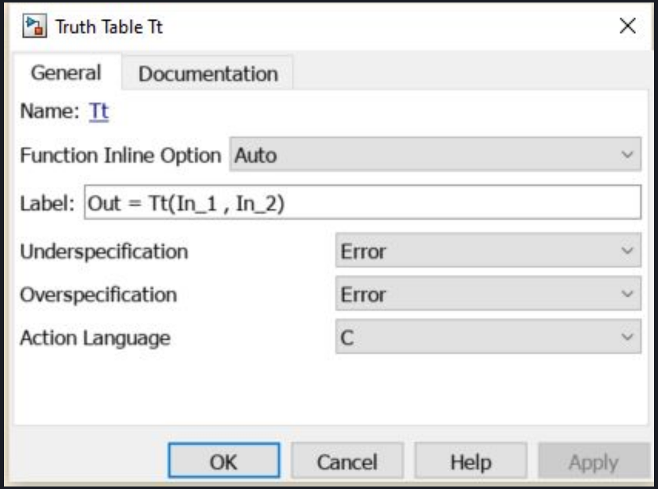
- The truth table is represented like a graphical function with the keyword ‘truthtable’ in the upper left corner
- The truth table ‘function’ can then be called from transitions and/or states
- In the example the truth table is called via the default transition

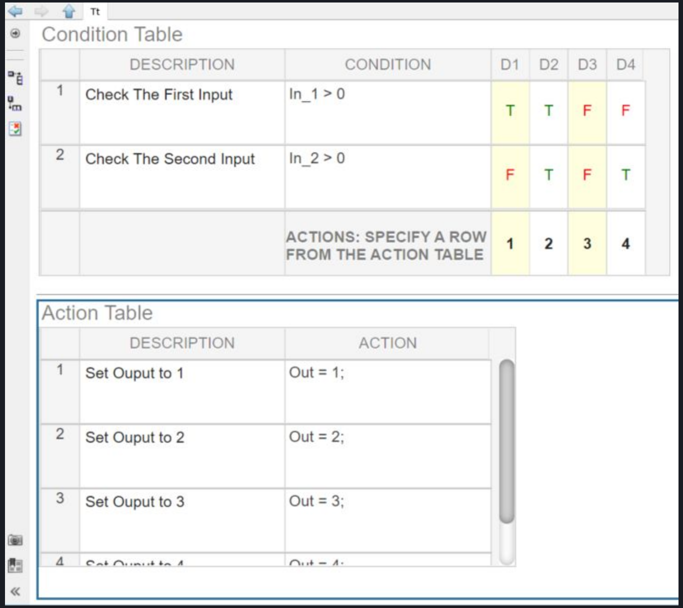


Stateflow Design

Truth Table

- Double click the truthtable block to access the actual truth table:
  - Set the underlying language to C
  - Fill the truth table

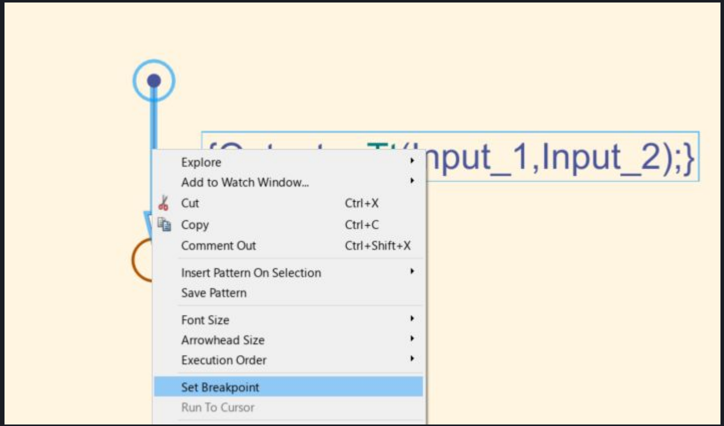




Stateflow Design

Truth Table

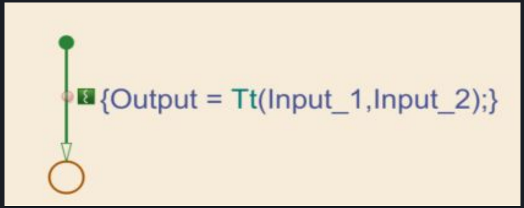
- Debug Truth tables from main chart
  - Set a breakpoint on the transition that triggers the truth table.
  - When the breakpoint becomes active step through the truth table with ‘step in’
  - The current action will be highlighted in the truth table and the underlying content.

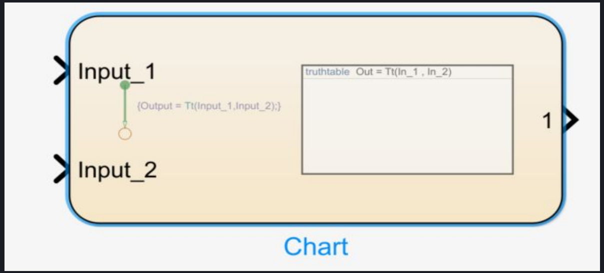


Stateflow Design

Truth Table

- Debug Truth tables from main chart
  - Set a breakpoint on the transition that triggers the truth table.
  - When the breakpoint becomes active step through the truth table with ‘step in’
  - The current action will be highlighted in the truth table and the underlying content.





TYPE	NAME	VALUE	PORT
	Input_1	0	1
	Input_2	0	2
	Output	0	1
	Tt		
	Out		
	In_1		
	In_2		

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Tt

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

Tt

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

- Debug Truth tables from main chart

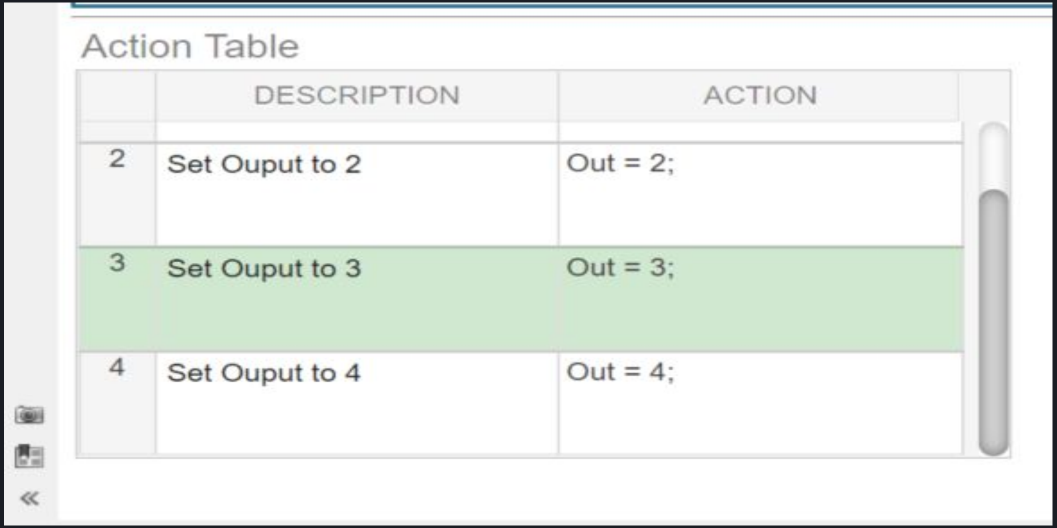
Action Table

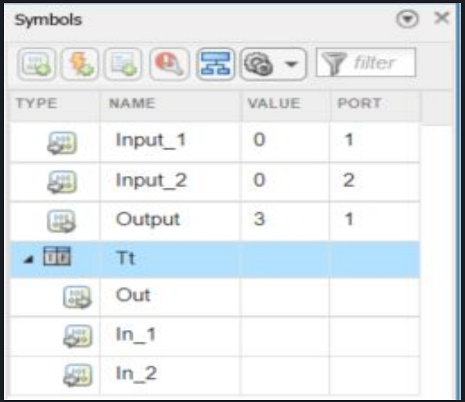
	DESCRIPTION	ACTION
2	Set Ouput to 2	Out = 2;
3	Set Ouput to 3	Out = 3;
4	Set Ouput to 4	Out = 4;

Stateflow Design

Truth Table

- Debug Truth tables from main chart

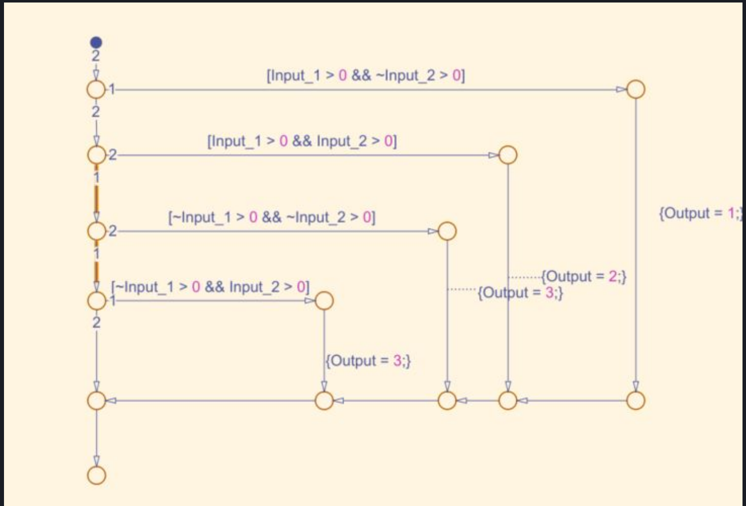




Stateflow Design

Truth Table

- Equivalent flow chart of the truth table



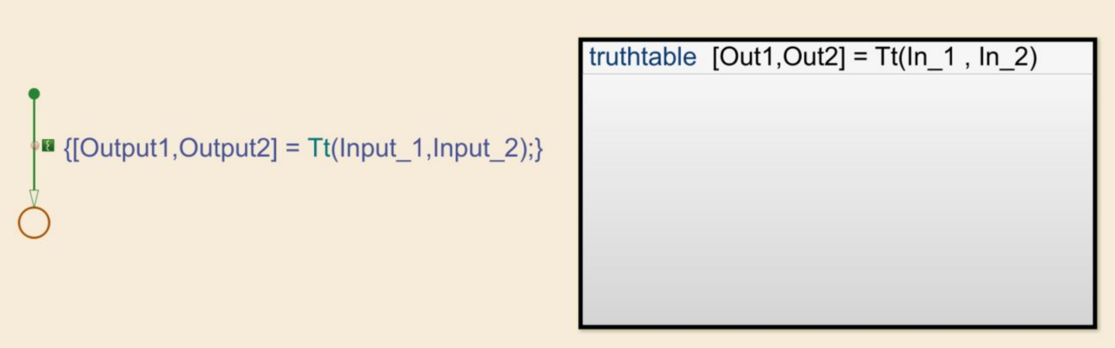
Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3	D4
1	Check The First Input	In_1 > 0	T	T	F	F
2	Check The Second Input	In_2 > 0	F	T	F	T
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3	4

Stateflow Design

Truth Table

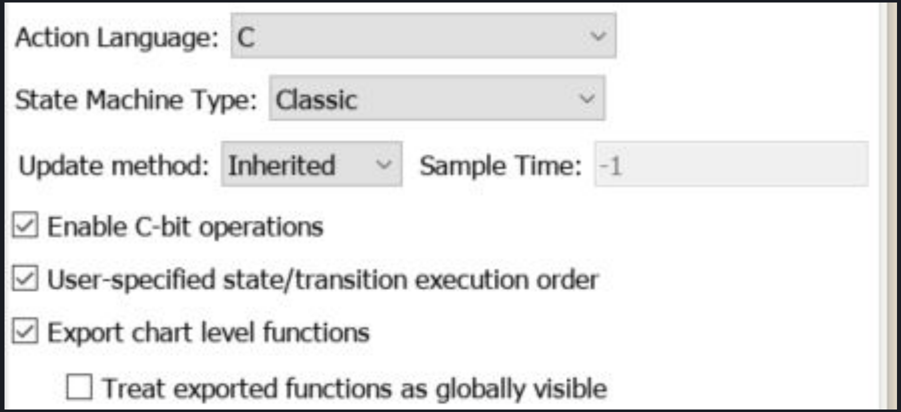
- Multiple return values in graphical functions/ truth table
  - Multiple return values are supported by using the Matlab langue bracket notation
  - Not standard C syntax, but supported in C charts



Stateflow Design

Global Stateflow objects

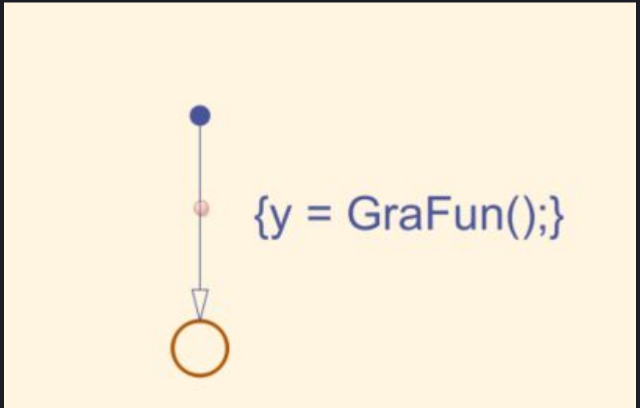
- Stateflow allows to generate model-wide functions
  - Only available if Graphical function / truth table reside in the chart top level
  - Functions can be called from other charts, allowing
    - Generation of generic services
    - Generation of model-wide utility functions

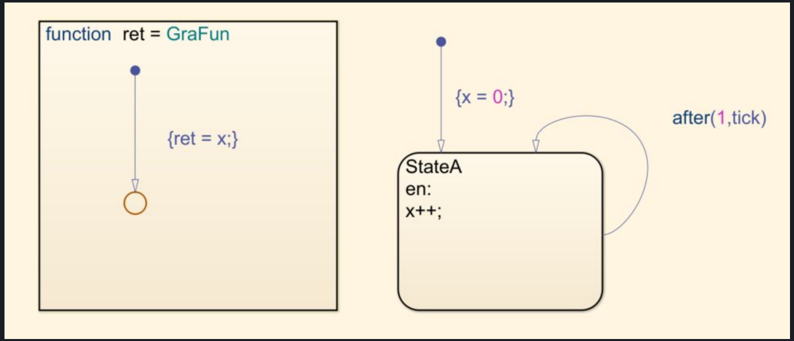


Stateflow Design

Global Stateflow objects

- Exercise

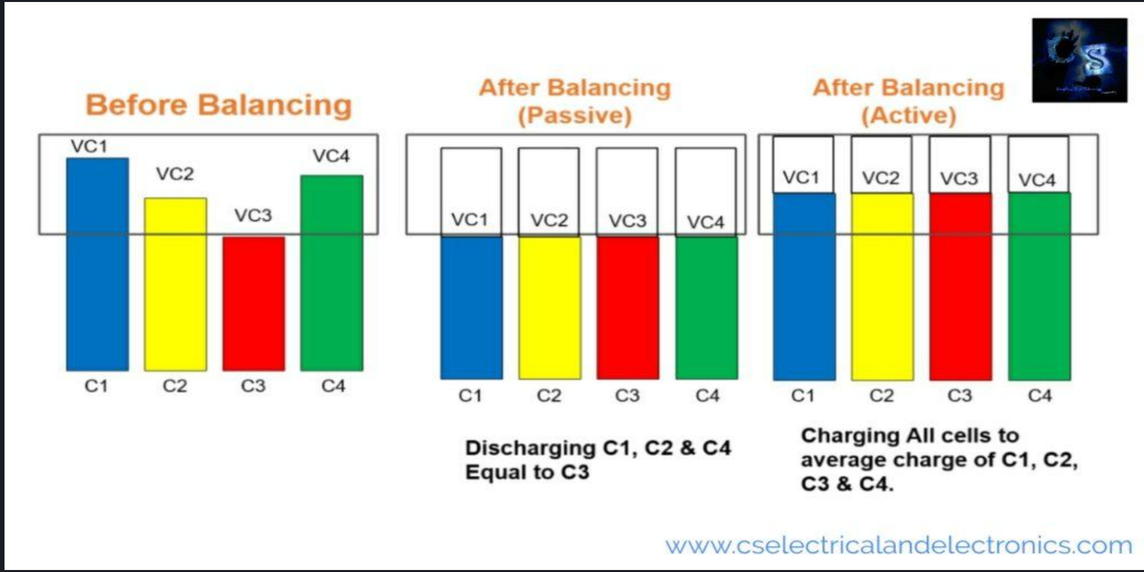




Stateflow Design

Final Project on Stateflow Design

- Cell Balancing



Thank You!