Lab Tutorial and Exercise 5

Systems Modelling with Stateflow/Simulink

This tutorial covers a fraction of the first two stages of the V-cycle described in the first lecture.

The aim of the tutorial is:

- To get insight into the usefulness of developing an executable specification, allowing questions such as the following ones to be answered:
  - Is the specification complete?
  - Is this behaviour really what I (and/or the person who ordered the system) want the system to do?
- To get hands-on experience with the graphical modelling of state machines, being similar to programming, but performed on a higher abstraction level. We will use the Simulink add-on Stateflow for these purposes.
- To illustrate how continuous time dynamics models can be simulated jointly with the logic of state machines.
- To illustrate that the logic part of a Stateflow diagram can be event- or time-triggered.

Stateflow is an interactive graphical design tool that works with Simulink to model and simulate event-driven systems, also called reactive systems. Event-driven systems transition from one operating mode to another in response to events and conditions. The corresponding systems are called discrete-event dynamic systems, as opposed to discrete-time dynamic systems – the term used for sampled systems. Discrete-event dynamic systems are often used to model logic for controlling a physical device such as a fan, motor, or pump. Event-driven systems can be modelled as finite state machines.

Finite state machines represent operating modes as states. For example, a house fan can have states such as High, Medium, Low and Off. To construct finite state machines, Stateflow provides graphical objects that you can drag and drop from a design palette to create state-transition charts in which a series of transitions directs a flow of logic from one state to another.

In this tutorial, you will learn how to develop a simple model including a Stateflow chart in the Simulink environment. It is common for embedded systems to include lots of logic (modes of operation) together with control systems code. Using Stateflow and Simulink, it is possible to model these two aspects more conveniently (logic is more cumbersome to describe by only using Simulink).
The tutorial will illustrate the process of creating, changing and debugging your model. Try to answer the questions posed as they will give you more insight into the environment. First quickly skim the complete tutorial – then go to work!

1 The Specification

The specification roughly corresponds to the behaviour of an electronically heated car seat.

The heating of the seat is controlled by a button which has three positions; {neutral, half, full}, see Figure 1. The button is always in the neutral position if not pushed because of two springs. The left part of the button is marked with an H. If this part is pushed, this means that 50% heating should be applied to the seat. The right hand part of the button is marked with an F. If this part is pushed, the seat should be fully heated (100%).

![Figure 1. Button controlling the car seat.](image)

If 50% heating is applied to the seat, pushing H once more should stop the heating, whereas pushing F should result in 100% heating.

If 100% heating is applied to the seat, pushing F once more should stop the heating, whereas pushing H should result in 50% heating.

Question: Consider the above specification – do you consider it to be complete? We will come back to this question later during the tutorial.

2 Introduction to Stateflow

What is a Stateflow chart? Figure 2 shows an example which models, as a finite state machine, the logic required to shift gears in an automatic transmission system of a car.

Notice the following details in this flow chart:

- Each gear and shift position is represented by a state
- Some states are exclusive (only one can be active at a time) while others are parallel (can be active concurrently).
- Transitions can be triggered by events and conditions.
- States can execute actions while they are active.

You will learn more about these features later in this tutorial as you build your own Stateflow chart.
2.1 How to build a Stateflow chart

To develop a state machine model using Stateflow you first have to create a Simulink model. Using the Simulink browser you create a Stateflow chart by selecting Stateflow and Chart, from the Simulink library browser.

According to the Mathworks, developing a Stateflow chart can be divided into seven steps, as illustrated in Figure 3.

From a systems point of view, it is clear that before you do this, you have to know

- what functions you want to describe
• what functions are more suitable to describe using ordinary Simulink blocks, and which parts are better described using Stateflow.

Given that you have also decided to use Stateflow to describe a portion of the system functionality, you also have to consider when this functionality should execute, e.g. in a periodic fashion or triggered by some event.

Now, assuming you have an initial idea of the functions to describe using Stateflow (in this case, the main portion of the specification and behaviour logic described in section 1), you can proceed according to the procedure described in Figure 1, including the following steps:

1. Defining the interface to Simulink. In this step, you define how the Stateflow chart is connected to the other parts of the Simulink model. In particular, you define the inputs required from Simulink, and the outputs that the chart should produce. You can then connect these inputs and outputs to the appropriate signals in the Simulink model.

2. Defining the states, types if states and their structuring. It is usually possible to develop solutions with a different number of states. Stateflow allows you to define hierarchical states – this is a powerful technique in which certain states are grouped together and where events “common” to all the grouped states only have to be defined once. In addition, states can be defined to be sequential – so called OR states, or to run in parallel – AND states.

3. Defining state actions and variables. Actions can be defined to take place when entering a state, when exiting a state, or while a state is active – “during” a state (actually they can also be defined for transitions, but we will skip that part here). Variables can be defined for states, and just as in programming, they can be temporary or persistent.

4. Defining transitions between states. Transitions create paths for the logic flow of a system from one state to another. When a transition is taken from state A to state B, state A becomes inactive and state B becomes active. Transitions have direction and are represented in Stateflow charts by lines with arrowheads. You must add a transition for each direction of between two states. Guarding a transition means specifying a condition, action or event that allows the transition to be taken from one state to another.

5. Triggering a Stateflow chart. Simulink can wake up a Stateflow chart by sampling the chart at a specified or inherited rate, by using a signal as a trigger, or by using one Stateflow chart to drive the activity of another.

6. Simulating the chart. This is not so different from ordinary simulation in Simulink. The main difference is that you should set up the use of the Stateflow debugger which is a very useful tool. It is similar to an ordinary debugger in that it allows you to set breakpoints and to step through the execution of the Stateflow chart.

7. Debugging the chart. Here you execute the chart with the debugger.
3 Step-by-Step Tutorial

Note! If you have problems – or if you want to read more – then you have the following help available:

- the Stateflow tool help functionality: Open a Stateflow chart and click on Help, Stateflow Help
- the Stateflow web documentation:
  

3.1 Creating a Simulink model and introducing a Stateflow chart

First create a new Simulink model.

Then create a simple model of the push button, for example as illustrated in Figure 4:

![Figure 4. Simple model of the pushbutton](Image)

As a first step, we will concentrate on the inputs to the chart and the operation of the chart. You will later add an output signal that controls the actual heating of the seat.

Save you Simulink model with an appropriate name and give the name “SeatLogic” to the chart by clicking on “Chart” (the default name of the Stateflow chart) and entering the new name.

Ensure that your Simulink directory is the right one by typing “pwd” or “cd” (the result should be the directory where your model is stored).

3.2 Defining the Interface to Simulink

Inputs and outputs are data elements in a Stateflow chart that interact with the parent Simulink model. To define inputs and outputs for your Stateflow chart, follow these steps:
Double-click on the SeatLogic chart to open the Stateflow chart. The Stateflow Editor opens on your desktop, as shown in Figure 5.

There are several ways to add inputs and output signals to your chart. To do this, we will now use the Add menu shown above.

- Add a data element to hold the inputs from the button from Simulink by following these steps:
  - Select Data > Input from Simulink from the Add menu.
  - The Data properties dialog box opens on your desktop with the General tab selected.
  - You only need to change the name from “data” to Fbutton.
  - Leave all the other fields in the General tab at their default values, as they should be sufficient.

Note: Ports are assigned to inputs and outputs in the order they are created. Because Fbutton is the first input you created, it is assigned to input port 1.

- Select the Value Attributes tab and select the Watch in debugger check box.

- Enabling Watch in debugger allows you to examine the values of the variable during breakpoints in simulation. You will try this later in section 3.9 of this tutorial.
• Click OK to apply the changes and close the dialog box.

Repeat the procedure for the other input required: *Hbutton*

Look back at the Simulink model by clicking the up-arrow in the Stateflow Editor toolbar: You should be able to see that two inputs have been created. Connect these two to the respective button signals.

**Tip:** There are several ways to add data objects to Stateflow charts. You have now used the Stateflow Editor, which allows you to add data elements to the Stateflow chart that is open and has focus. However, to add data objects not just to a chart, but anywhere in the design hierarchy, you can use a tool called the Model Explorer. This tool also lets you view and modify the data objects that you have already added to Stateflow.

You can start the Model Explorer from the **Tools** menu by pressing **Explore**.

### 3.3 Defining the States for Modelling Each Mode of Operation

You have now entered phase 2 of the basic workflow for building a Stateflow chart: defining the states for modelling each mode of operation.

Basic considerations include deciding how many states (modes) you need. States model modes of operation in a system. To determine the number and type of states required for your SeatLogic chart, you must identify each mode in which the system can operate. For example, the system can be idle, at 50% heating or 100% heating. You will find that there are different solutions to the same problem, requiring a different number of states.

Depending on the incoming events, the system should take different actions. Often, a table or grid is helpful for analyzing each mode and determining dependencies between modes.

To define a state, go to the Stateflow Editor for SeatLogic. Notice the object palette on the left side of the editor window. This palette displays a set of tools for drawing graphical Stateflow chart objects, including states.

• Left click on the state tool icon: ![State tool icon]

• Move your cursor into the drawing area. The cursor changes to a rectangle, the graphical representation of a state. Click in the upper-left corner of the drawing area to place the state.

• The new state appears with a blinking text cursor in its upper-left corner.

• At the text cursor, type *Idle* to name the state.

• Move the cursor to the lower right corner of the rectangle so it changes to this symbol: ![Corner symbol]

• Drag the lower-right corner to enlarge the state.

• Now your task is to think first (for a little while) on the number of states that you will need
Then you add the additional states to the chart. One possible solution is shown in section 3.4.

### 3.4 Defining State Actions and Variables

States perform actions at different phases of their execution cycle from the time they become active to the time they become inactive. Three basic state actions are:

<table>
<thead>
<tr>
<th>Type of Action</th>
<th>When is the action executed?</th>
<th>How often is the action executed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>When the state is entered (becomes active)</td>
<td>Once</td>
</tr>
<tr>
<td>During</td>
<td>While the state is active and no valid transition to another state is available</td>
<td>At every time step</td>
</tr>
<tr>
<td>Exit</td>
<td>Before a transition is taken to another state</td>
<td>Once</td>
</tr>
</tbody>
</table>

| Table 1. Actions used in Stateflow |

For example, you can use entry actions to initialize data, during actions to update data, and exit actions to configure data for the next transition. (There are other types of state actions, but they involve concepts that go beyond the scope of this guide. For more information, see Using actions in Stateflow in the online Stateflow User’s guide documentation.)

In this tutorial we will not define variables in states. However, we will define actions.

You should now add an output from your state chart to the rest of the Simulink model, where the output is a signal that could be used for actuation.

- Select **Data > Output** from the **Add** menu.
- Call this output HeatSignal, e.g. enter in the **Name** field.

Having defined this output, you can complement your Simulink diagram by attaching a scope to your output. Your Simulink model should now look as shown in Figure 7.
Next, you will add entry actions to the states that you have defined. The syntax for entry actions is:

\[ entry:\{one \ or \ more \ actions\}; \]
\[ en:\{one \ or \ more \ actions\}; \]

To write an entry action for the \textit{Idle} state, follow these steps:

- Click inside the \textit{Idle} state after the last letter of its name label to get a blinking text cursor.
- Press the Enter key and type \textit{entry: HeatSignal = 0;}
- Add entry actions to the other states as well.

To ensure that you are on the right track, you should now have a Stateflow chart that looks something like Figure 8:

\textbf{Figure 8. Stateflow chart after adding entry actions.}
3.5 **Defining transitions between states**

Transitions create paths for the logic flow of a system from one state to another. When a transition is taken from state A to state B, state A becomes inactive and state B becomes active. Transitions have directions and are represented in a Stateflow chart by lines with arrowheads. You must add a transition for each direction of flow between two states.

You should consider the following design questions when defining transitions between states:

- How does my state machine transition from one operating mode to another?
- Where should I place default state transitions?
- How should I guard each transition from one state to another?

Good design practice in Stateflow requires that you specify default transitions for exclusive (OR) states at each level of hierarchy. Default transitions indicate which exclusive (OR) state is to be active when there is ambiguity between two or more exclusive (OR) states at the same level in the Stateflow hierarchy.

Guarding a transition means specifying a condition, action or event that allows the transition to be taken from one state to another.

This for example applies to your chart – where you have to define the idle state as the default one.

In placing such a *default transition*, follow these steps:

- In the Stateflow Editor, left click the default transition icon in the object palette:

  ![Default Transition Icon](default_transition.png)

- Move your cursor into the drawing area.
- The cursor changes to a diagonal arrow.
- Place the cursor at the left edge of the *Idle* state.
- When the arrow becomes orthogonal to the edge, release the mouse button.
- The default transition attaches to the *Idle* state. It appears as a directed line with an arrow at its head and a closed tail.

To insert transitions between states, consider for example inserting a transition from the *Idle* state to the *Half* state.

- Move the cursor over the edge of *Idle* (closest to *Half*), until the cursor shape changes to crosshairs.
- Hold down the left mouse button, drag the cursor to the closest side of *Half*, and release the mouse.

You should now see a transition pointing from *Idle* to *Half*.
3.6 Setting conditions on transitions

Conditions are expressions enclosed in square brackets that evaluate to true or false. When the condition is true, the transition is taken to the destination state; when the condition is false, the transition is not taken and the state of the origin remains active.

An example condition, assuming an input variable temp, is the following

\[ \text{temp} \geq 120 \]

In your case, the transitions should be taken based on the status of the button. If these are considered as Boolean signals, you can for example use the condition

\[ Hbutton \]

As a condition to transition from the Idle to the Half state.

To add conditions,

- Click the transition you want to work with, e.g. from Idle to Half.
- The transition appears highlighted and displays a question mark (?).
- Click next to the question mark to display a blinking text cursor.
- Type the expression corresponding to the condition.

You may need to reposition the condition for readability. Click outside the condition, then left click and drag the condition expression to its new location.

Repeat the steps given in sections 3.5 and 3.6 until you have added all the transitions needed in the model. After doing this, you should have Stateflow chart looking something like Figure 9.

*Figure 9. Stateflow chart with transitions, including conditions.*
3.7 Triggering a Stateflow chart

Simulink can wake up a Stateflow chart by

- Sampling the chart at a specified or inherited rate
- Using a signal as a trigger
- Using one Stateflow chart to drive the activity of another

We will start by using periodic sampling, assuming for example a microprocessor implementation that polls the button. Alternatively, you will later try using event-triggers, i.e. mimicking interrupts.

Start the Model Explorer and point to SeatLogic. Choose “Discrete” as update method and choose an appropriate sampling time. In this case, the system will be polling a human. What do suggest as a suitable sampling time?

If the rate is too slow, the system could miss button pushes, if the rate is too high, this will consume CPU time, and moreover, might cause one push to be interpreted as several.

![Figure 10. Setting the sampling rate for a periodically executed Stateflow chart.](image)

3.8 Setting up the simulation of the chart

You are now very close to be able to simulate your model.

When you simulate a Simulink model, you can animate Stateflow charts to highlight states and transitions as they execute. Animation provides visual verification that your chart behaves as expected. Animation is enabled by default, but you need to set the speed. To configure animation for simulation session, follow these steps:

- Make sure animation has been enabled for your chart, as follows:
  - In the Stateflow Editor, select Open Simulation Target from the Tools menu.
  - The Stateflow Target Builder dialog box opens on your desktop, as shown in Figure 11.
Figure 11. The Stateflow Target Builder.

Note: This dialog box is used to configure Stateflow for building targets. A target is a program that executes a Stateflow chart or a Simulink model that contains a Stateflow chart. Stateflow builds a simulation target (sfun) that lets you simulate your Stateflow application in Simulink. For more information, see Building Targets in the online Stateflow User’s Guide documentation.

- Click the Coder Options button in the middle of the dialog box.
- The Stateflow sfun Coder Options Dialog opens on your desktop, as shown in figure 12.

Figure 12. The Stateflow sfun Coder Options dialog box.

- Make sure that the Enable debugging/animation checkbox is checked.
- Close both dialog boxes.

- Set the speed of animation, as follows:
  - From the Stateflow Editor, open the Stateflow debugger by selecting Debug from the Tools menu, or by clicking the Debug icon.
  - The Stateflow debugger opens on your desktop, as shown in Figure 13.
  - Set the delay to 1 second so the animation will proceed at the slowest speed.

Note: You can change the speed of animation at any time during simulation.
The debugger is a very important window. It is further described in the following section.

3.9 Debugging the chart

As you can see from the debugger window, it is possible for you to set breakpoints on for example Chart Entry and State Entry.

- Set breakpoints for both Chart and State Entries.

With the start and step button you can perform debugging, similar to what you do in a programming environment.

At the bottom of the Debug window, you can watch the value of variables – and in the Stateflow chart, you will see how the entering and transitions between the states is animated.

Figure 14 shows a snap-shot from the debugging. It will be useful for you to also have access to the Simulink model during debugging – because you will want to push the button to test your state machine!
3.10 Concluding the polled solution to the seat heating logic

Some considerations for you concluding this solution are as follows;

- If you run the simulation without breakpoints, how does your solution behave when you have a fast sampling rate?
- How does your solution behave when you have a very slow sampling rate?

When running your model (with or without the debugger) it is possible that you will find that the behaviour is not quite as you had expected. The behaviour will depend on the sampling period, how the inputs (buttons) are modelled and the durations of the pushes.

Hopefully, you will see by now how useful this modelling and simulation technique is. It allows you to understand the problem better, as well as studying conceptual solutions (time- vs. event triggering from the button).

As a further part of this tutorial, you will now change your Stateflow chart so that it is instead an event-triggered solution.

3.11 Changing to an event-triggered solution

There are several ways this can be achieved. We will here illustrate one approach.

For the new solution, we suggest you store your previous model and create a new one.

A basic step in setting up the event-triggered solution is to define

1. The Stateflow chart to be event-triggered.
2. Input events for the chart.
3. A suitable interface and connection between your new chart and Simulink.
4. Define suitable transitions in the charts based on the events.

The first step is accomplished from the Model Explorer by defining the update method as Inherited, shown in Figure 15.

You also need to ensure that the check box “Execute (Enter) Chart at Initialization” in the Model Explorer is enabled.

![Figure 15. Defining update method for an event-triggered system](image)

The second step “Input events” is also performed in the Model Explorer. Two new events have to be added. To give them the same names as the two “old” data inputs, you have to delete these first. Then create two event inputs with Scope set to “Input from Simulink” and Trigger to “Rising”. This is shown in Figure 16.

![Figure 16. Creating an event in the Model Explorer.](image)

The idea behind this is shown in the Simulink diagram in Figure 17.

Although you have defined two external events (triggers) for the chart, only one trigger will be visible in the chart block in the Simulink model.

Notice that the two input signals are fed into a Mux block where they are joined into an array to a single output. When you connect the Mux to the trigger port, the index of the signals in the array are associated with the like-numbered ports.
You will also have to adjust the Stateflow chart so it uses events instead of conditions, as shown in Figure 18.

**Figure 17. Event-triggered Stateflow chart in Simulink.**

**Figure 18. Adjusted Stateflow chart**

**Hint:** You have to modify all the transition conditions. Previously they were based on conditions.

A transition is characterized by a label. The label can consist of an event, a condition (and more things not covered here).

The question mark (?) character is the default transition label. Transition labels have the following general format:
event[condition]/condition_action)/transition_action

Previously you used the condition expressions, e.g. \([Hbutton]\). Now you will instead use the event inputs for the transitions. This means that it suffices to label the transitions with the appropriate names of events – as shown in the previous figure.

3.12 Debugging the event-triggered solution

**Hint:** To facilitate the debugging of this solution it is useful to set the simulation time to infinity. You do this through the **Simulation** menu of your Simulink model (also available in the Stateflow chart):

In **Simulation, Configuration Parameters** – set **Stop time** to infinity: \(inf\).

Does your new solution behave better compared to the polled one?

Is the requirement specification complete?

- No, it probably isn't. If you forget to turn off the heater when turning off the motor, it should probably be turned off automatically after a certain time to prevent emptying the battery of power. This is a possible extension you may want to try.

- Another suggestion for extension that you may want to try is to build a polled solution that actually works as it should (compared to the one showed in this tutorial). This however means you have to build a state machine with at least three additional states compared to the solution in this tutorial, and it is considerably more complex using a lot more transitions. Can you find the trick used for building it?

Other suggestions for further – but optional – work are provided in the following section.

4 Further work

1. Change the update method from discrete to inherited.

2. Modify the Stateflow diagram such that the Fbutton functions as a data input (shown in figure 7) and Hbutton as an event (figure 17).
   a. The transition from Half to Idle should occur when Hbutton = 0.
   b. The transitions “Full to Half” and “Idle to Half” should occur when Hbutton=1.

**Hint:** Use a clock signal for the Hbutton trigger.

5 Further reading and hints for more work

Online Stateflow tutorial


There are also several demos provided with Matlab. One of them is **sf_car** – type this at the command line to start it.
Where to go next. You have completed a basic workflow for building a Stateflow chart, but there is more to learn. To gain further experience with Stateflow, explore these resources:

- **Stateflow documentation** – The MathWorks provides extensive documentation on how to work with Stateflow using the graphical user interface and the API. To access the Stateflow documentation, follow these steps:
  - In the Matlab window, select Help from the Start menu.
  - In the Help navigator, select the Contents pane and scroll down to the Stateflow node.
  - Select the Stateflow node and follow the links to documentation resources, including real-world example models.

- **Stateflow demos** – Stateflow provides a collection of demonstration models, which you can access as follows:
  - In the Matlab window, select Demos from the Start menu.
  - The Help browser opens, displaying information on how to get started using demos.
  - In the Help Navigator, expand the Simulink node to expose the Stateflow one.
  - Expand the Stateflow node to examine demonstration models that illustrate a variety of applications
6 Exercise

1. What are the first two stages of the V-cycle?
2. Why requirements are very important? Give at least two reasons.
3. What is the difference between requirements and specifications?
4. Why there is no condition on one of the transition arrows in figure 18?
5. Name three different kinds of events that you can add in Stateflow.
6. Name the four different kinds of triggers used for different events in Stateflow.
7. What is the difference between different transitions that is implemented in part 4 of this tutorial?
8. Name the three basic state actions for a state in Stateflow.
9. Is it possible to use function call trigger for Hbutton and rising edge trigger for Fbutton?
10. How can Stateflow help you in your project work?