Project Group 'Racing Car IT'

Statemachine Graphical Editor

User documentation

Rishab Dhar

University of Paderborn

Summer Semester 2012

Advisers:

Prof. Dr. Marco Platzner, Tobias Beisel, Sebastian Meisner, Lars Schäfers

Contents

1 Statemachine Editor

1.1 Introduction

The Statemachine Editor is a graphical tool to model the tasks that the user wants to execute on the microcontroller (AT90CAN128). The tool is delivered as an Eclipse plugin. The user needs to have Eclipse IDE installed and the Eclipse.zip to use this tool. The statemachine tool is an extension of the Mealy-More machine. It is assumed that the user well versed with the semantics of Mealy-More machines to work use this tool productively.

This document proceeds with describing the various parts/elements of the editor and then moves on to the installation process, and usage of the Statmachine Graphical Editor. To demonstrate how this editor can be used for generating C code, a simple statemachine example is provided.

1.2 Brief Overview of the Editor

The editor allows the user to model a statemachine, which runs at a specified base rate (in seconds or milliseconds), with one of the 4 supported states and transitions between any of these states. In addition, the editor also supports inclusion of global code in GlobalCode boxes. The notation of the basic statmachine elements – *InitialState, NormalState, FinalState, and SuperState, Transition, and GlobalCode* is shown in Figure 1. As can be seen in the figure, there is another statemachine element, the Region, which is contained in SuperState. Only a SuperState can have a Region. The states, regions, and global code boxes are labeled with State/Region/GlobalCode names, while the transitions are labeled with one of the possible triggers, followed optionally by a list of actions – the set of instructions (such as lighting up an LED) that the statemachine performs – to be executed. The semantics of transition, action, and the 4 types of states are discussed in the following sections.

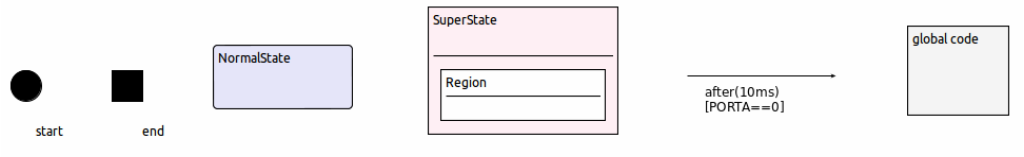

Figure 1: Basic elements of Statemachine Editor

1.2.1 Actions

An action consists of a sequence of C language statements that are executed by the statemachine. The NormalState, SuperState, and FinalState can have 4 different types of actions – ENTRY, EXIT, DURING, and ALWAYS. The ENTER action is performed when the statemachine enters that state; the EXIT action is performed when the statemachine exits that state; the DURING action is executed when the statemachine is in that state, and ALWAYS action as the name suggests is always executed. The separator / is used to separate the C statements from the action type. Below are a few examples on actions,

ENTER/PORTA++
EXIT/PORTA = 0
ALWAYS/ increment_portb()

1.2.2 States

As mentioned before, there are 4 types of states, with the SuperState having a special statemachine element called Region. Each region can have a complete statemachine. So it is possible to create nested statemachines by using regions. Each of these regions and states must have a name that is a valid C identifier. Also, each region and each of the 4 states has a parent, which decides the context of that region or state. Within a given context each state/region must also have a unique name.

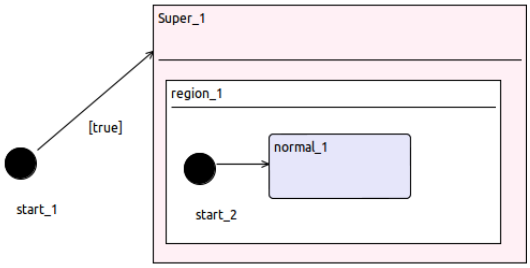

Figure 2: Example on State and Region parent

In Figure 2 the parent of start_1 and Super_1 is the State Machine, so these states are in State Machine context. The parent of the region_1 is Super_1, so it is in State Machine ⟶ Super_1 context. Finally the parent and context of start_2 and normal_1 are region_1 and State Machine ⟶ Super_1 ⟶ region_1 respectively.

1.2.3 Transitions

Each transition has a source and a destination state. The InitialState should not be a target of any transition and not be a source of more than one transition. While the FinalState cannot be a source of any transition. Each transition can have three type of triggers (except for transition from InitialState, which cannot have any triggers) : **event**, **waitType**, and **guard condition**. These triggers can be optionally followed by an *action*. Also, a transition can have a priority, which can be useful in case there are multiple transitions originating from a state. The event name should be a valid C identifier and should be a valid C expression that evaluates to true or false. The general syntax for labeling a transition is shown below.

event :waitType(rate) [condition] / action

An event triggered transition takes place when that event occurs, and the statemachine is in the source state of that transition. Examples of events include a button being pressed or an interrupt. There are three possible types of waitTypes – *before*, *after*, and *wait*. Each waitType specifies when the transition should take place. The waitType should also include time information, such as wait(10ms) or after(1s), where ms or s are abbreviations for milliseconds and seconds. A condition triggered transition takes place when that condition evaluates to true.

A few examples of valid and invalid transition labels are shown in Table 1.

Valid Transition Labels	**Invalid Transition Labels**
turn_off : before(10ms) /PORTA=0	turn_off before(10ms) PORTA = 0
[PORTA == 0xff] / PORTA = 0	PORTA == 0xff / PORTA
increment : after(10ms) [PORTA != 0] wait(10ms)	2increment : after(10ms) [PORTA != 0] wait(10)

Table 1: Examples of Transition labels

1.3 Installing Statemachine editor plugin

The graphical editor works out of the box once you install the statemachine plugin from the EclipsePlugins.zip archive. A step-by-step description for installing the plugin is given below.

1. Select in the Eclipse drop down menu (*Help* $\longrightarrow$ *Install New Software*)
2. You can add the repository of the Statecharts plugin on the Install window by (*Add* $\longrightarrow$ *Archive*). A file explorer is opened, in which navigate to the directory containing the EclipsePlugins.zip archive and import it like in Figure 3. This archive contains two UPBracing Softwares, of which only

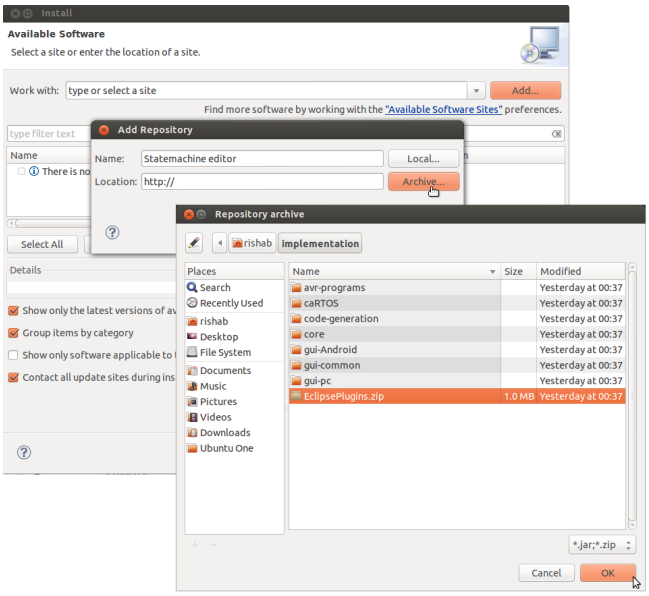

Figure 3: Importing EclipsePlugins.zip folder

the UPBracingStatemachineEditor is of interest to us. You might want to install the UPBracing AVR Timer–Configuration too if you want to work on timers (refer to UPB-TimerConfiguration for more on it), but they are not at all required for working on statemachines.

If you have performed the above steps successfully, it means you have successfully set up the graphical editor. You can now move on to the next section to get started with working on the editor.

1.4 Getting started with the Editor

Once you have successfully set up the statemachine editor, you need to create an empty Eclipse project (containing *.statemachine* and *.statemachine_ diagram* files) before you can start modeling any statemachine. The .statemachine_diagram contains the graphical version of the statemachine, whereas the other file contains the tree of the statemachine that you draw.

A step-by-step guide for creating an Eclipse project containing these files is given below :

1. Create an empty simple Eclipse project (*New* $\longrightarrow$ *Project*) and give it some arbitrary name as shown in Figure 4.

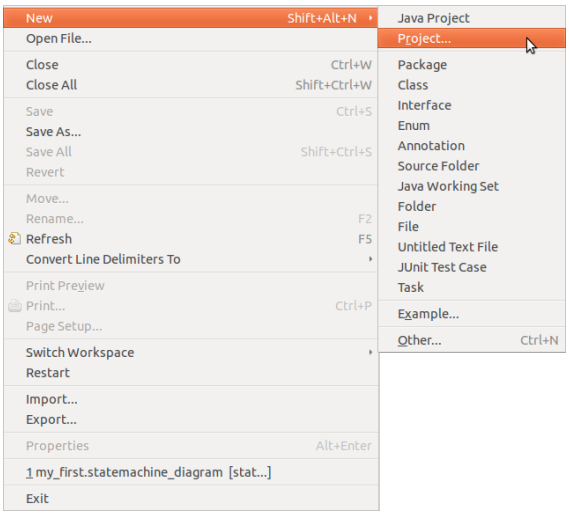

Figure 4: Creating an empty Eclipse project

2. Right click on this newly created project and open the wizard for creating Statemachine Diagram (*New* $\longrightarrow$ *Other* $\longrightarrow$ *Statemachine Diagram*(select) $\longrightarrow$ *Next* (Do not click finish or the diagram would be called "default")) as shown in Figure 5.

3. Now name the statemachine files (*Select the empty project that you created before* $\longrightarrow$ *Give an appropriate name to the .statemachine_ diagram file* $\longrightarrow$ *click Next* $\longrightarrow$ *click Finish letting the .statemachine file name remain the same*) as shown in Figure 6. You can model your tasks in either one of the two files. The .statemachine_diagram file is the one we are concerned with, although you can create the model in the other file too. Once you have finished modeling, you can delete the .statemachine_diagram file, as

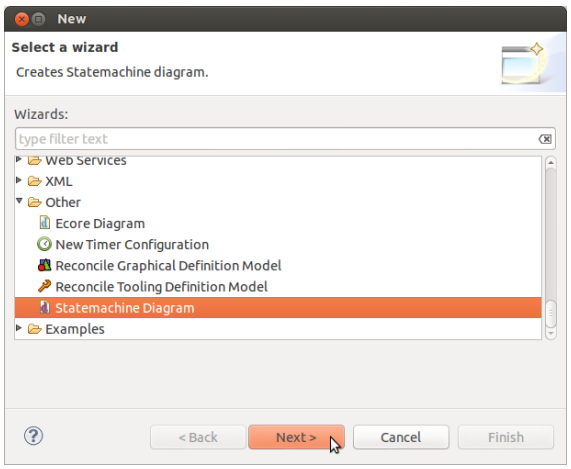

Figure 5: Statemachine Diagram wizard

it can be generated from .statemachine file (although the layout would be lost) ; however, the opposite is not true.

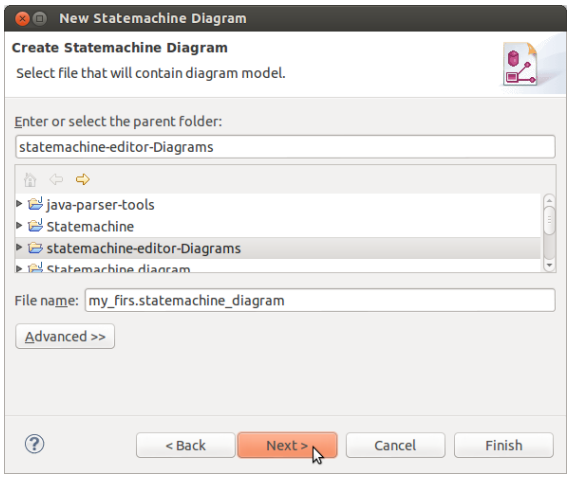

Figure 6: Naming Statemachine diagram file

1.5 Modeling Statemachine elements

This part guides the user on how to model various statemachine artifacts (from *Canvas* or *Palette* – the 4 types of states , transitions, and regions, and includes instructions on how to resize the various artifacts, set some advanced properties, and use multiline textfields where possible. To go any further into performing any of these tasks, you first need to bring Canvas to the view by opening .statmachine_diagram file.

1.5.1 Modeling from Canvas view

Not all elements can be modeled from the canvas, and some elements can be frustrating to model. For instance, the Region cannot be modeled from the canvas, and modeling a transition can be frustrating task compared to modeling other elements. Modeling the states or the GlobalCode from the canvas as simple as placing the mouse on it and selecting one of the diamond like icons (as shown in Figure 7) from the container that shows up in a few seconds.

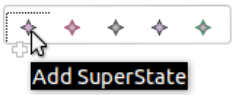

Figure 7: Pop up in the Canvas

To model a transition (the normal way), you already need to have the two states between which you want to have a transition. Once you have the states, draw the transition between the two states (*Place your mouse over any of the two states to bring the two arrows to front* Figure 8 $\longrightarrow$ *Select one of them* (depending on whether that state is the source or the destination of the transition) $\longrightarrow$ *Drag your mouse to the other states*).

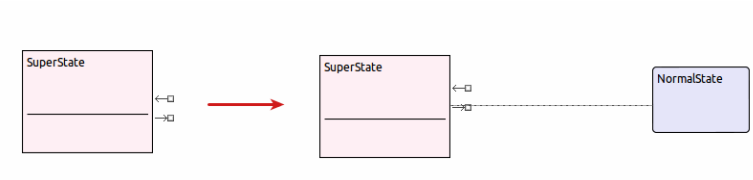

Figure 8: Transitions from canvas

In case the source and the destination of the transition are the same, bring the the two arrows to front just like in any other transition, select the incoming arrow and drag the arrow slightly on the border of the state, and finally release mouse button as in Figure 9.

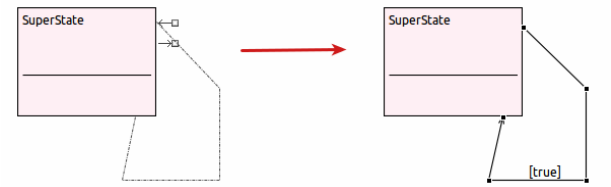

Figure 9: Self transitions from canvas

Another possible way to model transitions from the canvas is to first hover your mouse over either the source or the destination state and while selecting one of the arrows (depending on whether the state is source or destination) drag your mouse over the canvas and finally release the button there as in Figure 10.

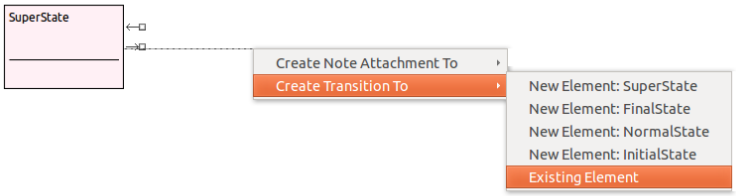

Figure 10: Transitions from menu in the canvas

A menu would pop up, where you can either decide to create a new state as the source/destination of your transition or select an existing state as the source/destination (see Figure 11). This technique can be particularly useful when the statemachine gets really big, and it gets hard to locate the source/destination state on the canvas to manually model the transition.

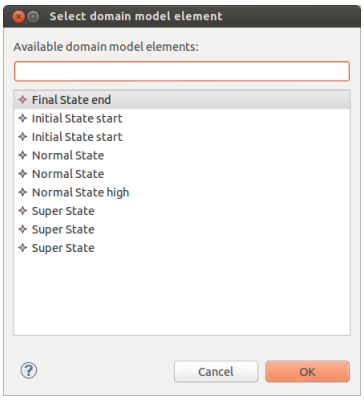

Figure 11: Select exiting state as source/destination of transition

1.5.2 Modeling from the Palette

All statemachine elements can be modeled from the Palette Figure 12, and it is recommended that you use the Palette because it not only helps your to model all statemachine elements , but also helps you to place the elements at the precise position you wanted it in first place. To model from the palette, simply select the desired state, region, or global code and then move your mouse over to the canvas to model it, with the option of just clicking and releasing the mouse button on

the canvas to set the size of the element to default size (for resizing please read resizing section) or clicking and dragging your mouse with the element selected to set element size as desired.

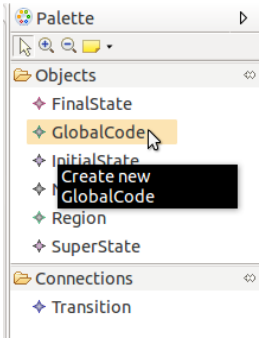

Figure 12: Palette

You can model transitions from the Palette in two ways. The second way is quite similar to the way from the canvas. In the first way, you need to first select Transition under Connections from the palette, and then while selecting the source state (in case of SuperState you need to select from the top half of the partition) drag your mouse to the interior of the target state where you finally release the select button. If you want to model a self transition, just select the state and then release the button. In the second way, you just need to select either the source or the target state and then with the state selected drag you mouse anywhere in the canvas, producing a pop-up menu quite similar to the one in Figure 10. You can then model the transition just like you would model it from the canvas.

1.5.3 Resizing States, Regions, and GlobalCode boxes

The InitialState and FinalState are by default of optimum size, so there is no need to resize them (though you can resize them in the same way as described for other statemachine elements). However, in case of NormalState, SuperState, Region, and GlobalCode the size is really small when you first model them (shown in Figure 13), either from palette or from canvas, so they need to be resized.

To resize them, you need to select the element, then move the pointer to the edge or the corner from where you want to resize, and finally with the element still selected drag the mouse in the desired direction. However, in case of Region, if you haven't resized the SuperState already, it is better to resize it first (at the same time adjusting the position of region by selecting and moving it in the

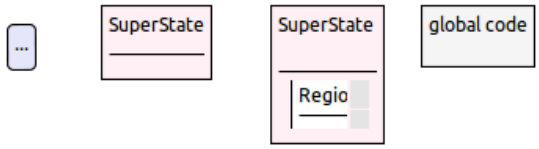

Figure 13: Default size of NormalState, SuperState, and GlobalCode

SuperState) to the point there are no more scrollbars as you see in figure below and finally resize the region in the same way as the others are done. You can see the resized versions of the elements in Figure 14.

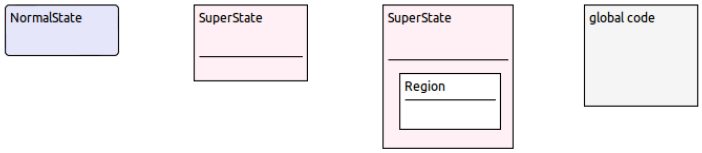

Figure 14: Resized NormalState, SuperState, and GlobalCode

1.5.4 Setting property values of statemachine elements

There are two possible ways to set the property value of a statemachine element – from the *Canvas* or from the *Properties view*. Setting property values from the canvas can quite tedious, so using properties view is recommended in that case. Also, using properties view has many added advantages such as being able to set advanced properties, for eg parent of the statemachine element or statemachine base rate, and being able to set a property value in case that property has been deleted from the canvas. You need to have properties view open for that (*Window* $\longrightarrow$ *Show View* $\longrightarrow$ *Other..* $\longrightarrow$ *Select Properties (under General) from the Menu that pops up*) or *Right click statemachine element* $\longrightarrow$ *Show properties view*).

Canvas View

Below is a list of properties that can be set from the Canvas view and instructions on how to set them :

State, Region, and GlobalCode box names To set Initial or Final state name, simply select the default name (start or end) and type in your desired name. To

change the NormalState, SuperState, Region, or GlobalCode box names you first need to select their container and then select the default name to change it as in Figure 15.

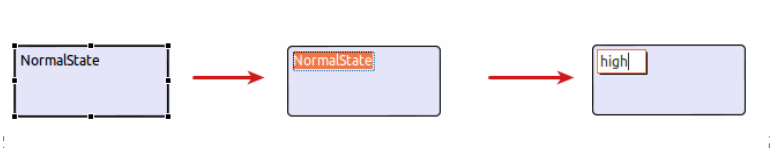

Figure 15: Rename Normal state

TransitionInfo The TransitionInfo can be set by selecting the default value of TransitionInfo which is [true] and then typing in the desired transition information as in Figure 16 .

Figure 16: Setting Transition Info

Actions and Code in GlobalCode boxes To add actions to the states or code to GlobalCode boxes, first select the state, and then with your mouse close to the bottom of the element (in case of SuperState slightly above the partitioning line) click again to bring the cursor to front as in Figure 17.

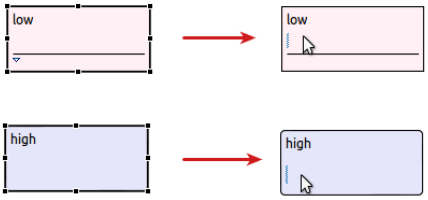

Figure 17: Add actions or code

Properties view

To set the property of any statemachine element, you first need to select it to bring the properties view for that element to front. The following section describes how various property values can be set from the properties view.

States and Region The properties view of NormalState is shown in Figure 18. You can set the actions, parent, and name of the state here (actions cannot be set for

InitialState, and for regions only name and parent can be set). For incoming and outgoing transitions it is not of much use because you cannot create a transition here, but just choose one of the existing transitions as an incoming or an outgoing one.

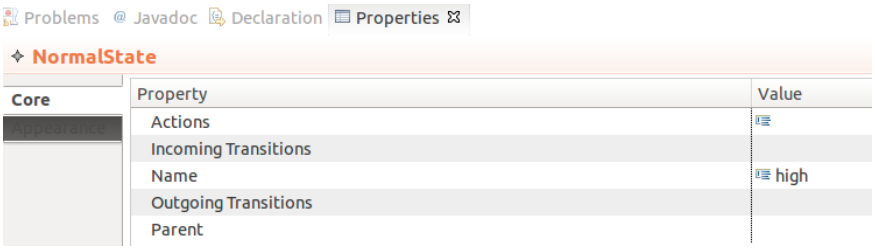

Figure 18: Set NormalState properties

Transition The properties view of a Transition in Figure 19 shows the various properties that can be set. Setting most of these properties is as trivial as typing in the value of the property. Source and destination values can be set by selecting the state from the scrollbox.

Figure 19: Transition properties view

GlobalCode boxes The properties view for GlobalCode boxes Figure 20 lets you set the additional "In Header File" property in comparison to the properties that you set directly from the canvas.

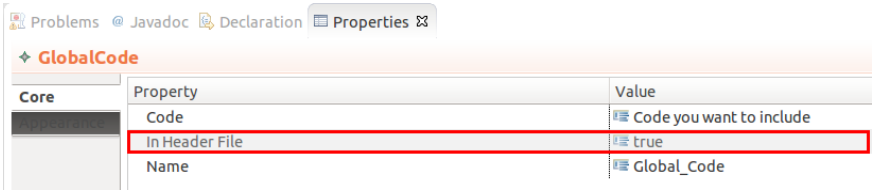

Figure 20: GlobalCode box properties view

Statemachine base rate The statemachine base rate can be set by clicking anywhere on the canvas and setting the value of "Base rate" property in the property view brought to front (see Figure 21).

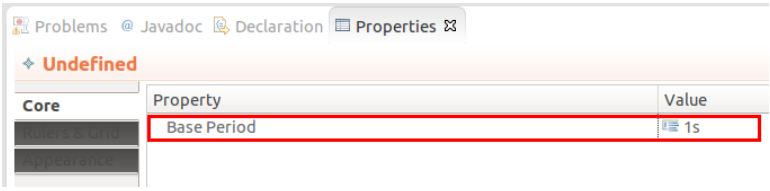

Figure 21: Setting statemachine base rate

1.5.5 Using multiline textfields for properties

The editor supports **multiline textfields** for *Actions* in the NormalState and the SuperState, for *conditions* on transitions, and for *Code* in GlobalCode. It is not possible to use multiline textfields from the properties view of an element. Therefore, to use multiline texfields on any element, you must first have that element's property field selected, in case you haven't, and then press **CTRL+ALT+ENTER** where you want to break to a newline (see Figure 22).

Figure 22: Multiline text fields for action in NormalState

1.6 A Simple Statemachine example

This example acquaints the user with the editor by walking through a simple statemachine example. This statemachine lights up the 8 leds on the DVK90CAN1 board. It is assumed that you have already installed the editor on eclipse.

First, create a folder called 0005-led containing in code-generation/tests/0065-statemachine directory. In this folder, create the two statemachine files – example.statemachine and example.statemachine_diagram and model the statemachine shown in Figure 23.

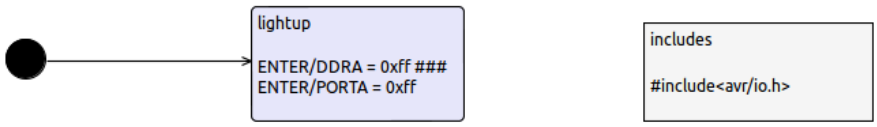

Figure 23: Example statemachine that lights LEDs

This statemachine has a statemachine base rate of 1s, and transitions immediately from InitialState to NormalState called lightup. Upon entry into the lightup state, all the 8 pins of PORTA are set as output and high, thereby lighting up all the 8 leds. For using the variables PORTA and DDRA, the requisite avr/io.h is included in the GlobalCode box called includes, with "In header file" property set to true.

Second, create config.rb file as shown in Listing 1 in 0005-led folder. This file is needed to invoke the statemachine code generator on the modeled statemachine.

Listing 1: config.rb

```
1 #DEPENDS ON:
2
3 $config.statemachines.load("exam_sm", "example.statemachine")
```

In Listing 1 the $config object is used to access the statmachines object, which is an instance of StatemachinesConfig class, and load the "example.statemachine" to the list of statemachines. This statemachine is named "exam_sm". The code generator will create the statemachines.h header file containing all the function declarations and statemachines.c file containing all the implementation of the functions in the header file.

Finally, create a main.c file as shown in Listing 2. In the main function, the statemachine is initialized by calling the exam_sm_init() function – the statemachine is called by using the syntax <statemachinename>_init(). The declaration of this function will be contained in the statemachines.h file (this file is generated when you call the generator for statemachine), so in order to use the initialization function you need to include this header file.

Listing 2: main.c

```
1 #include<statemachines.h>
2
3 int main() {
4     exam_sm_init();
5 }
```

Once all the steps have been performed, you are ready to generate code for your microcontroller. Simply go into the terminal and call the rake for your platform and build the example statemachine using build-statemachine-led. Refresh the 0005-led folder, and you will find two folders – bin and gen. The gen folder will contain all the files generated by the statemachine generator (you can have a

look at the statemachine.c and statemachine.h files here), while the Debug and Release folders in bin directory contain the hex files that you can run on your MCU and test the example.

References