Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-44001

Spec Title: FINITE STATE MACHINE IN PSOC(R) DESIGNER (TM) - AN44001

Sunset Owner: PATRICK KANE (PKX)

Replaced by: NONE

Finite State Machine in PSoC® Designer™

AN44001

Author: David Cooper
Associated Project: Yes
Associated Part Family: CY8C20xxx, CY8C21xxx, CY8C24x23A, CY8C24794, CY8C27xxx, CY8C29xxx
Software Version: PSoC Designer™ 5.0
Associated Application Notes: AN2261

OBSOLETE

Application Note Abstract

The finite state machine is a powerful construct, suitable for implementation of the algorithms required for today's embedded designs. The PSoC Designer™ System Level Design State Machine transfer function is a well architected, efficient, and verified state machine engine that allows you to easily create a flexible state machine. You can define states and transition events in a graphical environment and the tool generates the code. This application note provides a brief overview of the state machine fundamentals and discusses how to use the PSoC Designer™ System Level Design tool to create a state machine and integrate it into your design. The example project demonstrates many of the ideas presented.

Introduction

The finite state machine has been at the core of numerous embedded designs for many years. It is one of the popular building blocks used to create robust software designs. System behavior is concisely expressed as a logical organization of states, transition events, and associated system behavior based on these states and transitions. A key advantage of a system designed around a state machine is the ability to modify the design. This includes adding or deleting states or transition conditions without the need to rework the underlying code structure.

There are many ways to implement a state machine to solve a design problem, from building state driven hardware to creating your own table driven or switch statement based software engine. PSoC Designer™ System Level Design does this part of the work. You only need to focus on the system behavior that must be implemented by the state machine, express this using a graphical tool, and click the Build button.

State Machine Fundamentals

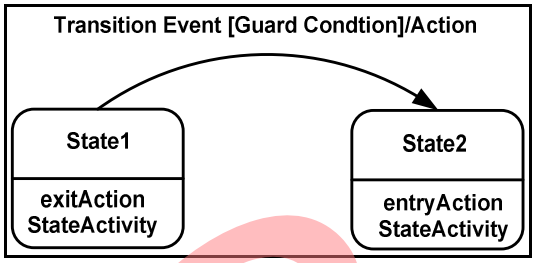
A finite state machine may be characterized by the required states, transitions, and activities derived from a problem definition. The state machine architecture is effective where distinguishable states are defined. Some basic definitions follow:

- **State.** Defines a set of condition coordinates from all possible coordinate values for a system. A state is a condition the system is in, which can be uniquely defined.
- **Transition.** Indicates a change of state. A transition may be triggered by an external event or an event generated by logic evaluation in your design.
- **Activity.** An ongoing task or function that must be executed; in this sense, it is associated with a state.
- **Action.** A task that is performed at a given time. An action may be seen as an "activity" that happens only once at a specific time.

The concept of a state implies retained information about the past behavior of a system. Transition conditions, actions, and activities allow the state machine to respond to system events and affect future events within the system.

Figure 1 illustrates the basic components traditionally associated with a state machine. A state machine has a number of states. Each of these states has associated entry or exit actions and activities that occur when in a particular state. Another notion central to the state machine construct is that of the transitions between states. A state transition is triggered by an event, which may additionally have a guard condition that must be true for the transition to occur. There may also be one or more actions that execute when the transition occurs.

Figure 1. Basic Finite State Machine Diagram



There is a subtle difference between an activity and an action. An action may be considered as an activity tied to a transition. The action only occurs on the change of state, whereas an activity is ongoing when in a given state.

The actions encompassing the state machine behavior are divided into the following categories:

- **Entry Actions.** Action is executed when entering the state.
- **Exit Actions.** Action is executed when exiting the state.
- **Transition Actions.** Action is executed when a specific state transition occurs.
- **Input Actions.** Action execution is based on the current state and the values of system inputs.

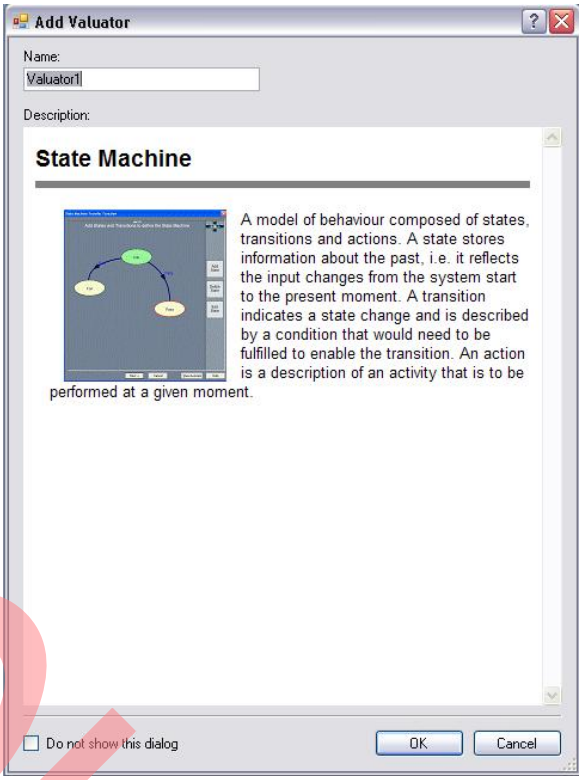
State machine models are characterized by the action type the model relies on. The PSoC System Level Design StateMachine transfer function supports the familiar Moore machine model. This model relies on entry actions where the output is determined by the current state value. By using the state machine state and transition information in other transfer function logic, activities and actions may be qualified by the current state and any other value in the design. Transitions are triggered by external events or are based on logic evaluations performed by other pieces of the design.

PSoC Designer System Level Design State Machine

In PSoC Designer System Level Design, a finite state machine is included in a design by placing a Valuator on the design desktop and selecting a StateMachine transfer function. A valuator creates variables within the design that have associated logic (the transfer function). This logic determines the values of variables at any given time. The StateMachine transfer function is one of the available logic constructs that may be associated with a valuator.

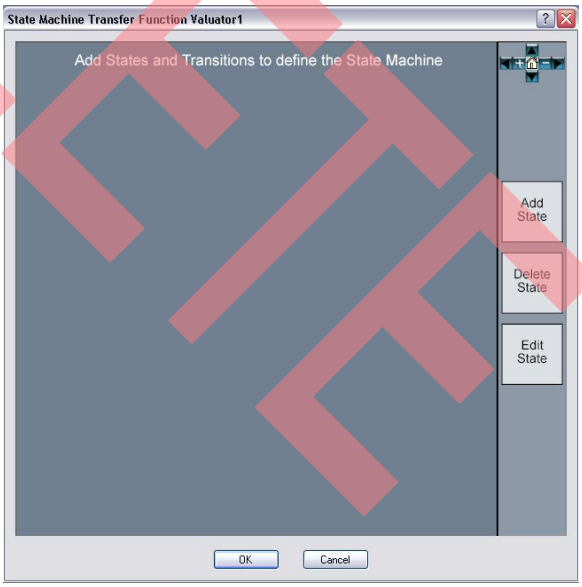
To add a StateMachine transfer function valuator to a project, select the Valuator tab in the Driver Catalog and expand the Transfer Function Valuators category. Double click the StateMachine label or select the label using the left mouse button and hold the button while you drag the label onto the design space. Figure 2 shows the Add Valuator dialog box that appears when a state machine object is placed on the design desktop.

Figure 2. StateMachine Transfer Function



Click the OK button to view the State Machine Transfer Function definition window (Figure 3).

Figure 3. State Machine Transfer Function Window



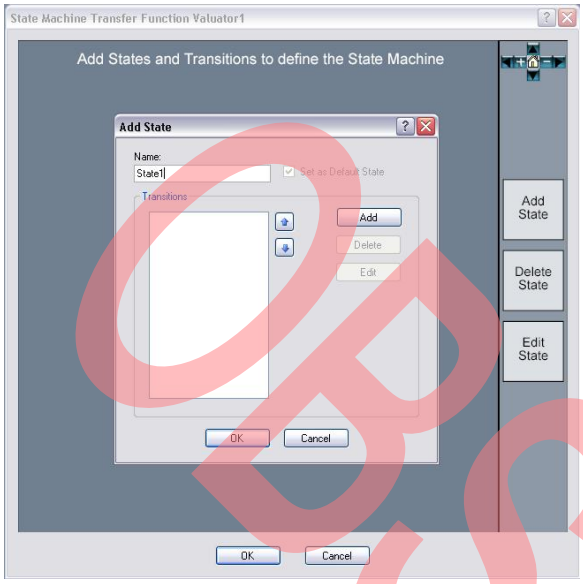
The three controls on the right of this window are used to define and work with the states that make up your state machine.

OBSOLETE

Adding States

Click the Add State button to view the dialog.

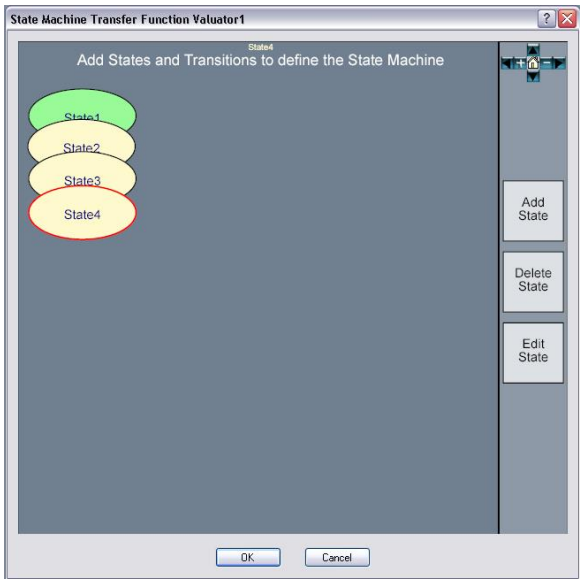
Figure 4. Add State Dialog



The state name is specified in the Name entry box. Check the “Set as Default State” check box if you want the state machine in this state when the system is initialized. The first state defined automatically becomes the default state. If this check box is selected for any other state, that state becomes the default.

Click the OK button to return to the State Machine Transfer Function window with the newly defined state displayed in the diagram area. Figure 5 shows a state machine diagram with four newly added states.

Figure 5. State Machine Diagram—Newly Added States



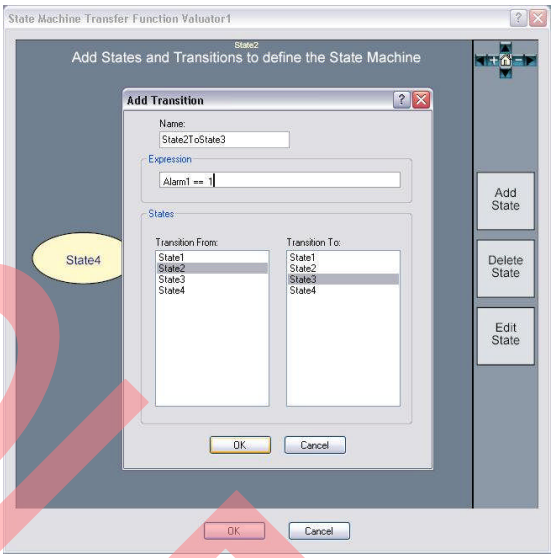
To rearrange the placement of states in the window, click on a state with the left mouse button. Hold the button while you drag the state to the required location.

Adding State Transitions

Both the Add State and Edit State dialog boxes include a Transitions section that lists the transitions defined for that state (Figure 4 shows the section with no transitions defined). The Transitions section includes Add, Delete, and Edit buttons to work with state machine transitions.

Clicking the Add button brings up the Add Transition dialog box (Figure 6).

Figure 6. Add Transition Dialog



Enter a name for the transition. Enter a logic expression to specify the event or guard conditions to control when the state transition occurs.

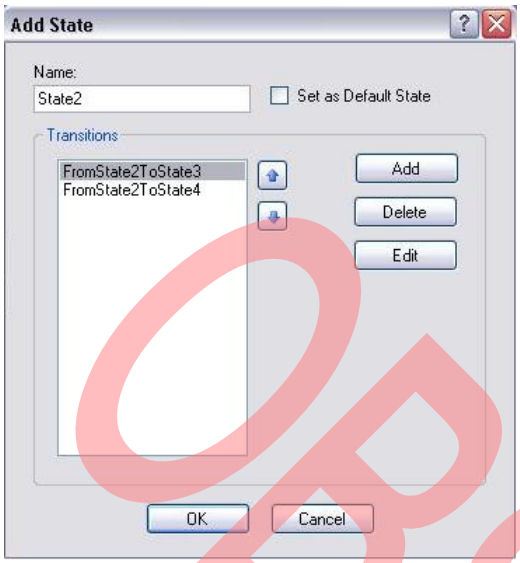
The flexibility in defining transition expression logic makes the state machine a versatile part of your PSoC System Level Design project. An expression can incorporate any value in the design. This includes other valuators and the input and output driver values.

Expressions are created in the same way as the expressions in other PSoC System Level Design transfer functions. An expression is defined using common logic operators (==, !=, <, >, and others). It may be a compound conditional using ‘&&’ (and) and ‘||’ (or) to combine values in creating the condition that governs the state transition.

In the States section, specify the ‘from’ and ‘to’ states for the transition. Each defined state is displayed in the “Transition From” and “Transition To” lists. Click on a state name to select that state as the originating or destination state for the transition. Click the OK button to close the Add Transition dialog and return to the Add State or Edit State dialog. The transition added appears in the Transitions area as shown in Figure 7.

OBSOLETE

Figure 7. Edit State Dialog with Defined Transitions

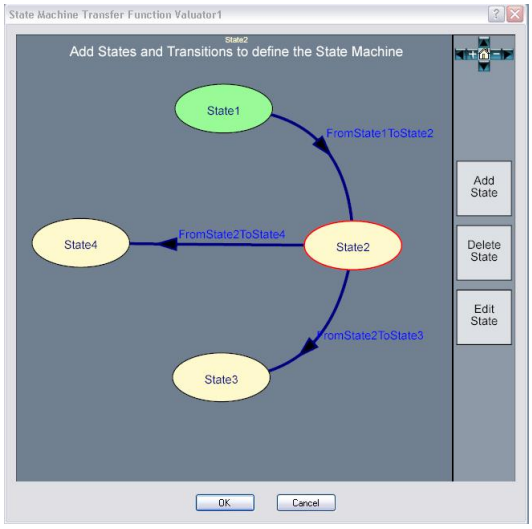


To add transitions between any two states, use the Add button in the Add State or Edit State dialog associated with the state. Note that only those transitions that originate in the state you are currently working in are displayed in the Transitions box. Transitions that originate in any other state are added to the design but do not appear in the state diagram until you return to the State Machine Transfer Function window.

After defining the state transitions, click the OK button to return to the State Machine Transfer Function window. The new state transitions are shown in the state machine diagram.

To move the transition lines in the diagram, click the arrow using the left mouse button. Hold the button while you drag the line to a desired location. Figure 8 illustrates a state machine diagram with the state and transition lines rearranged.

Figure 8. State Machine Diagram With Transitions



Transition Expression Evaluation

When a state has multiple exit transitions, the order in which the transition conditions are considered is relevant. The expressions specifying the transitions from a given state are evaluated in the order the transitions are listed in the Transitions section. The up and down arrow buttons next to the transitions list are used to change the order in which the transition expressions are evaluated.

Modifying States and Transitions

Despite a rigorous design phase, you may need to make changes to your project as development progresses. Modifying an existing state or transition in your state machine design is straightforward.

For an existing state, open the Edit State dialog in one of three ways: use Edit State selection in the State Machine Transfer Function window to open the active state (the active state is outlined in red); right click on a state and select Edit from the displayed menu; or double click on a state to open the Edit State dialog. In this dialog, rename the state and add, delete, or edit the associated transitions.

It is easy to work with the existing state transitions in your design. When you mouse over the arrow handle on a transition line, the associated expression logic is displayed at the top of the State Machine Transfer Function window. Clicking on the arrow handle opens the Edit Transition dialog. This allows you to rename, modify the logic expression, or reassign the “From” or “To” states for the transition. You can also open the Edit Transition dialog in the following ways: right click on the transition arrow and select Edit from the displayed menu; in the Edit State dialog, select the transition in the Transitions list and click the Edit button.

Integrating State Machine with Other Design Logic

The state machine can reference input and output driver and valuator values in the expressions that define the state machine behavior. Other project logic can use the state machine state and transition values to define behavior. This means that there is flexibility in interacting with other parts of your system.

State Transition Events

The methods in which system values are used to create expression conditions to trigger a state transition are discussed in the following sections.

Direct Event Transitions

A transition is directly triggered by an event as conveyed by an input driver value. The transition equation tests the input value to determine when the transition must occur.

(DigitalAlarm__Input == DigitalAlarmInput__ON)

OBSOLETE

Compound Condition Transitions

The transition expression can reference multiple values and use any number of logical operators to evaluate the transition condition. If the trigger conditions for a transition are overly complex, it might be appropriate to encapsulate the logic for the transition in a separate valuator. This allows you to determine the transition condition using any of the transfer functions available: PriorityEncoder, TableLookup, or even another StateMachine. The transition expression in your state machine is as follows:

((PreConditonValuaor1 == 1)

&& (PreConditionValuaor2 == 1))

This state transition happens when the logic associated with both referenced valuators yields a value of 1.

Time Based Transitions

State transitions are based on elapsed time. An Interval Generator input driver is used directly to create a periodic timer “tick” (select Pulse mode) to clock a state machine through the defined states. The transition expression consists of the following statement:

(IntervalTick1 == IntervalTick1__Triggered)

This expression causes a state transition each time the configured time interval elapses.

A more versatile timer is created using a PriorityEncoder transfer function. It implements a counter which increments or decrements at the rate determined by an Interval Generator driver. The transition expression then triggers a state change by testing the timer value. The transition expression has the following form if the timer decrements:

(DelayTimerCount == 0)

The transition expression has the following form if the timer increments:

(DelayTimerCount == elapsetimeLimit)

The logic in the PriorityEncoder resets the value either when the count expires or a new timeout value is calculated using other system values. Refer to application note AN2365 for more ideas on creating timers in PSoC Designer System Level Design.

Automatic Transitions

Another transition condition to consider is the value ‘1’. In this condition, the state machine is in the originating state for one loop and automatically transitions to a new state in the next loop iteration. This transition type provides a known delay between states that have dependant activities. It is also used to trigger a sequence of actions tied to the transitions that result from this type of logic.

Tying System Activities and Actions to State Machine Behavior

The usefulness of the state machine stems from its ability to control the activities and actions of a system. This is based on the history contained in the state information in conjunction with the current system input values. The StateMachine transfer function state and transition information combined with other transfer function logic is used to implement the types of actions and activities discussed earlier.

Exposed Variables

Both the current state and current transition (if applicable) values associated with a StateMachine transfer function valuator are exposed to the rest of the system. The state variable name has the form:

statemachinename_state

The transition variable name has the form:

statemachinename_transition

The set of labels for the values each variable can hold are presented in the Expression Assistant box in these forms: statemachinename_state__statename and statemachinename_transition__transitionname. These values are used in the transfer function logic of other design components to trigger actions or control activities based on state machine behavior.

Activities

The PSoC Designer System Level Design state machine model does not support the ability to define an activity that is automatically associated with a state. The classic notion of an activity that must be performed when a state machine is in a particular state is easily implemented by conditioning the output value of a valuator using the state machine state value.

The state value can be an input to a TableLookup transfer function or used in the expressions of PriorityEncoder or StatusEncoder logic. For example, a PWM output driver transfer function can be written so that the PWM is active only when the state machine is in a particular state. If a PriorityEncoder transfer function is used, the “if” expression in the transfer function logic tests the state machine valuator state variable to control the output driver value:

(statemachinename_state ==
statemachinename_state__DrivePWMState)

The “then” part of the expression pair sets the desired output value.

OBSOLETE

Actions

According to the working definition, an action occurs at one moment in time. In PSoC Designer System Level Design, this means on a single iteration of the loop. A transition action is tied to a state transition and an entry action to a new state. An input action is triggered when the state machine is in a given state and a particular system input takes on a specified value. These concepts are implemented in the transfer function logic of the dependant valuator or output driver using the state machine transition and state values in the transfer function logic.

A valuator or output driver transfer function directly implements a transition action by testing the transition value in the "if" expression:

(statemachinename_transition == statemachinename_transition__transitionname)

This expression evaluates true for a single iteration of the loop when the state transition occurs. In PSoC Designer System Level Design, the state machine state value has the new state value on this loop.

The "then" expression of the transfer function must implement the required action.

Implementing an input action requires using the state machine state value and the value of the relevant system input. The "if" expression of the transfer function follows:

((statemachinename_state == statemachinename_state__statename) &&

(RelevantInput == RelevantInput__On))

This condition executes the "then" expression only if the state machine is in a particular state and the specified input condition exists.

To define an entry action (triggered when a new state is entered) you must ensure the action occurs before any activity associated with the destination state. To do this, the activity condition must include both the state and transition values. The "if" expression must test whether the state machine is in the required state and the transition into this state did not just occur (the state machine has been in the state for at least one loop iteration). The "if" expression follows:

((statemachinename_state == statemachinename_state__statename) && (statemachinename_transition != statemachinename_transition__transitionname))

This condition is true on the second loop iteration after the state transition occurs and remains true until the state machine transitions to a new state.

A transition action, entry action, and an activity associated with a state are uniquely qualified using the state value, the transition value, and a LoopDelay transfer function. The LoopDelay transfer function acting on the state transition variable is equal to the value of the transition variable on the previous loop.

The three "if" expression conditions are:

Transition Action

(statemachinename_transition == statemachinename_transition__transitionname)

Entry Action

(statemachinenameTransitionDelayed == statemachinename_transition__transitionname)

State Activity

((statemachinename_state == statemachinename_state__statename) && (statemachinename_transition != statemachinename_transition__transitionname)

&& (statemachinenameTransitionDelayed != statemachinename_transition__transitionname))

This logic results in triggering the transition action first, the entry action occurring on the next loop iteration, and the activity condition becoming true after both actions are completed.

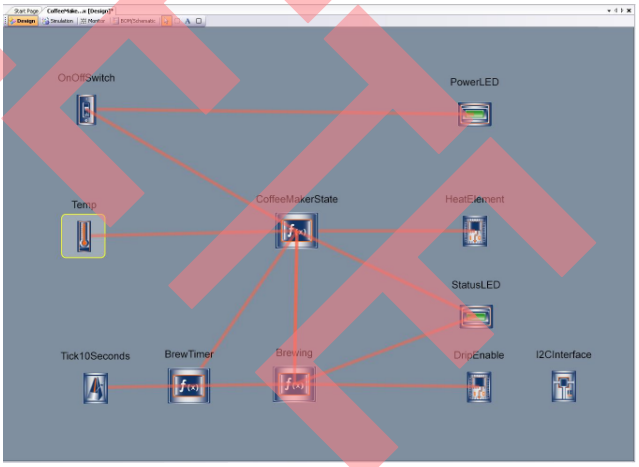
Example Project

The example project associated with this application note is the Coffee Machine State Machine design included in the System Level Design Catalog on the PSoC Designer System Level Design Start Page tab. The Coffee Maker project is in the System Control category.

Design Overview

The PSoC Designer System Level Design desktop view of this design is shown in Figure 9.

Figure 9. Example Project—A Simple Coffee Maker



The system inputs consist of an On/Off switch and a temperature sensor to monitor the temperature of the water or coffee.

OBSOLETE

The system controls four outputs. Two LEDs are used for power and status indication. The power LED indicates the position of the On/Off switch. The status LED indicates when the coffee is ready to serve. The LED blinks when the coffee is brewing and is on solid when brewing is complete. The status LED is off when the coffee maker is off or preheating the water.

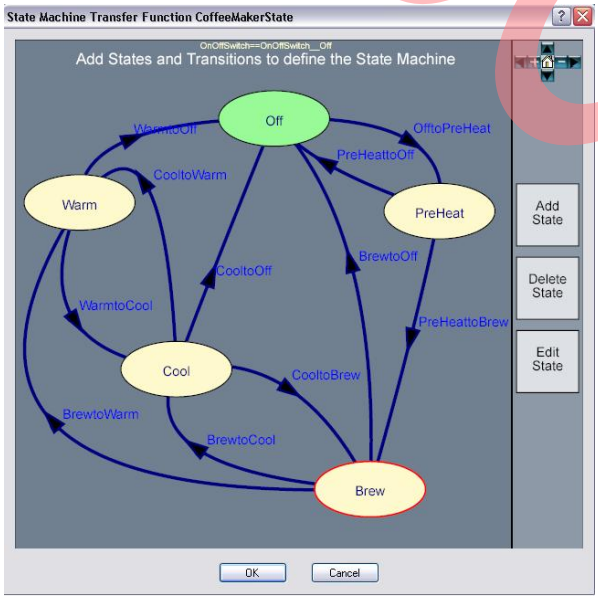
Two digital outputs, in the form of relay drivers, represent control of a heating element and an actuator to turn the water on or off.

The operation of the coffee maker is simple. When the unit is turned on, the water begins to heat. When the temperature reaches a predefined brewing temperature of 91ºC, the water drip is turned on and brewing begins. While the coffee is brewing, the water temperature is maintained between 89ºC and 93ºC. When the predefined brewing time (2 minutes) elapses, the water is turned off; the coffee maker maintains the temperature of the coffee between 83ºC and 87ºC. The coffee maker may be turned off at any time.

The State Machine

The bulk of the control logic for this design is implemented in the finite state machine—CoffeeMakerState. The State Machine Transfer Function definition window for the valuator is shown in Figure 10.

Figure 10. Coffee Maker State Machine Diagram



The five states of the state machine are as follows.

Off. The OnOffSwitch is off.

PreHeat. Initial state entered when the OnOffSwitch is turned on. The exit transition from this state is triggered when the water temperature reaches 91ºC.

Brew. Coffee is brewing. There are two transitions from this state: a transition to the Cool state if the Temp input driver value exceeds 93ºC and a transition to the Warm state when the BrewingTimer value reaches 0 (coffee is ready).

Warm. The system is in this state when the coffee is ready and the temperature is below 87ºC. The state machine transitions to the Cool state if the Temp input driver value goes above 87ºC.

Cool. In this state, the HeatElement output driver value is off (Low). The state machine transitions to this state if the Temp input driver value rises out of a target temperature range. When the temperature falls below the lower limit of the defined range, the state machine transitions back to the previous state. That is, if the coffee is brewing, the state machine transitions between the Brew and Cool states. When brewing is complete, the transitions are between the Warm and Cool states.

Transitions

The logic of the state machine is implemented in the expressions that control the state transitions. The transitions from or to the Off state are directly triggered by the state of the OnOffSwitch. The state machine transitions to the PreHeat state when the switch input driver value in on. Each state has an exit transition to the Off state that occurs if the switch value changes to off.

The Temp input driver value is also directly used to trigger transitions. The expression logic in the PreHeattoHeat transition and the cycling transitions to or from the Cool state make a greater than or less than comparison to determine when a transition must occur. The Temp input driver value is used for this comparison.

The exit condition from Brew to Warm is time based. A timer is implemented using an Interval Generator driver and a PriorityEncoder (BrewTimer) that decrements a counter at a specified rate. The BrewtoWarm transition expression triggers the transition when the BrewTimer value reaches zero.

Actions and Activities

The exposed state machine state and transition values are used by other design objects to affect the control of system outputs. The HeatElement output driver is directly controlled by the state machine state value. A TableLookup transfer function sets the output driver value to High or Low based on the current state machine state. This means that the output driver value responds to the current state and to changes in the Temp input driver value.

The Brewing valuator is an intermediate logic value that controls a number of design activities. The PriorityEncoder transfer function logic combines the notions of a transition action and an input action to control the water drip function. The Brewing value is set true by the PreHeattoBrew transition. The value is set false based on the state machine state value—when the state machine is not in either the Brew or Cool state. The Brewing value is used in other transfer functions and transition expressions. Each of these expressions requires the compound logic described earlier, if the intermediate value is not available.

OBSOLETE

Summary

The finite state machine is a powerful construct that can readily be applied to the solution of embedded system designs involving explicit states and decision based actions. The PSoC Designer System Level Design StateMachine transfer function provides a straight forward interface to define a fully flexible state machine. Using the exposed state and transition values in the transfer function logic of other design components you may implement nearly any state machine model. When you build a project, you get reliably generated code without having to spend hours creating a state machine infrastructure.

About the Author

Name: David Cooper

Title: Software Engineer

Background: David brings over 20 years of embedded systems experience to the PSoC team. He has done embedded system design and software development for sense and control, communications, and networking applications.

Contact: cvo@cypress.com

Document History

Document Title: Finite State Machine in PSoC® Designer™ - AN44001

Document Number: 001-44001

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2216369	PKX	03/17/2008	New Spec.
*A	3211917	PKX	04/18/2011	Updated to PSoC Designer
*B	4296627	PKX	03/12/2014	Obsolete AN

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip," PSoC Designer, and System Level Design are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/

© Cypress Semiconductor Corporation, 2008-2011. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

OBSOLETE