

Finite State Machines for MSP430

*Michael Lueders, Stefan Schauer**MSP430 Product Applications*

ABSTRACT

Many MSP430-based systems are likely candidates for implementation as finite state machines. A system that must sequence a series of actions or that must handle inputs differently depending on the mode it is in is often best implemented as a finite state machine. Typical examples are control-logic-oriented applications such as metering, monitoring, and control applications.

This application report describes a simple approach to implement finite state machines for MSP430-based systems with an easy-to-use Microsoft Excel® based code generation tool.

Contents

1	Introduction	1
2	Code Generation Tool	3
3	Run Demo	5
4	State Machine Development Steps	5
5	Summary	5

List of Figures

1	State Diagram of Finite State Machines	2
2	Demonstration Application	2
3	State Diagram of the Demonstration Application	3
4	FSMGenerator State Table	4

List of Tables

1 Introduction

1.1 Why Use Finite State Machines

Many MSP430-based systems are used in control-logic-oriented applications such as metering, monitoring, and control applications. One feature that these applications have in common is that the system must sense external events. Based on these events and the current state of the system, some external actions are controlled. The software of such a system is based mainly on conditional jumps and bit manipulations.

There are many ways to describe the behavior of such an MSP430-based system in software. Most popular is a large switch case construct in the main function, from which every sub function is called. Another, more elegant, way is the so-called finite state machine concept. Simply said, the application using this concept is seen from an event-driven point of view: if an event occurs, the system reacts to that change performing some action(s); the system may also change its state.

1.2 Definition of Finite State Machines

A finite state machine is an abstract computational model based on automata theory. It consists of a finite number of states that embody information about the current situation in the system. The state machine is always in one of the finite states. Events are outside stimuli to the state machine. Triggered by events, each state has transitions to other states. A transition does not have to move the state machine to another state; it can also loop back to the same state. During a transition, some actions (called transition actions) might be performed. In the target application, the actions are user-written C functions.

A state machine is very often represented by a state diagram (see [Figure 1](#)).

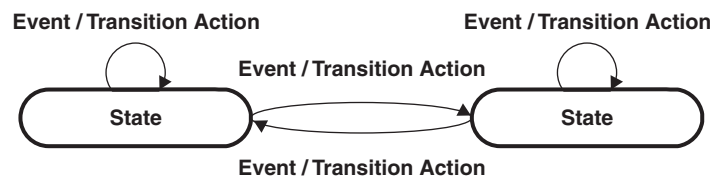


Figure 1. State Diagram of Finite State Machines

1.3 Demonstration Application

This application report describes a simple approach to implement a state machine for a MSP430-based system. In [Figure 2](#), this approach is demonstrated by a simple game.

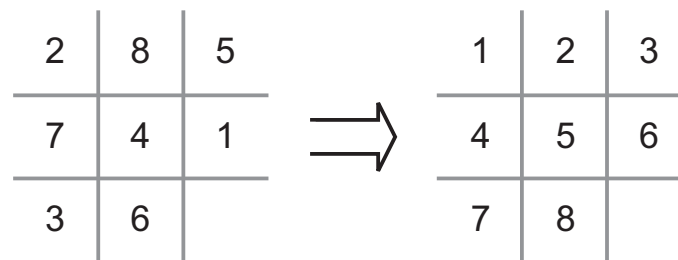


Figure 2. Demonstration Application

The game consists of 3x3 squares. In eight of these squares, the numbers one to eight are randomly placed. The aim of the game is to put these numbers in the correct order by pushing them one after another either left, right, up, or down.

From a state machine point of view, nine states can be defined, each representing one of the nine possible empty fields (in the following called "Empty Field 1" to "Empty Field 9"). Additionally there are two more states defined, one to represent the game initialization ("Init Game") and another one to represent the end of the game in case of winning ("Stop Game").

Four events can be defined representing the four possible directions to push the numbers (in the following called "Left", "Right", "Up", and "Down") and two more events for "Win" when all of the numbers are in the correct order and "Button" for a pressed button.

Altogether, this results in eleven states and six events as shown in [Figure 3](#).

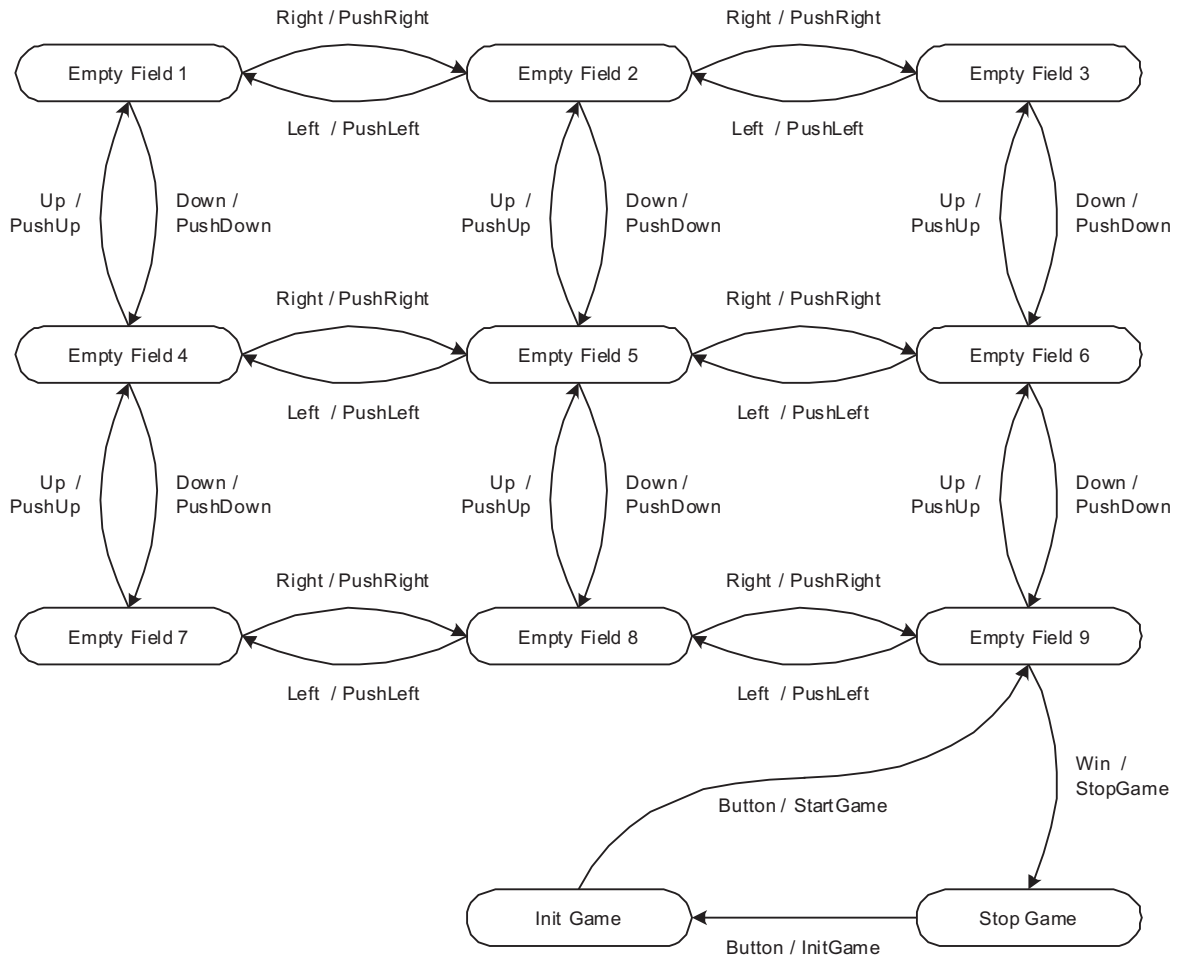


Figure 3. State Diagram of the Demonstration Application

To simplify the illustration, not all transitions are shown in this state diagram. The transitions to the "Init Game" state for a pressed button and the transitions from one state looping back to the same state for otherwise undefined transitions are not included.

2 Code Generation Tool

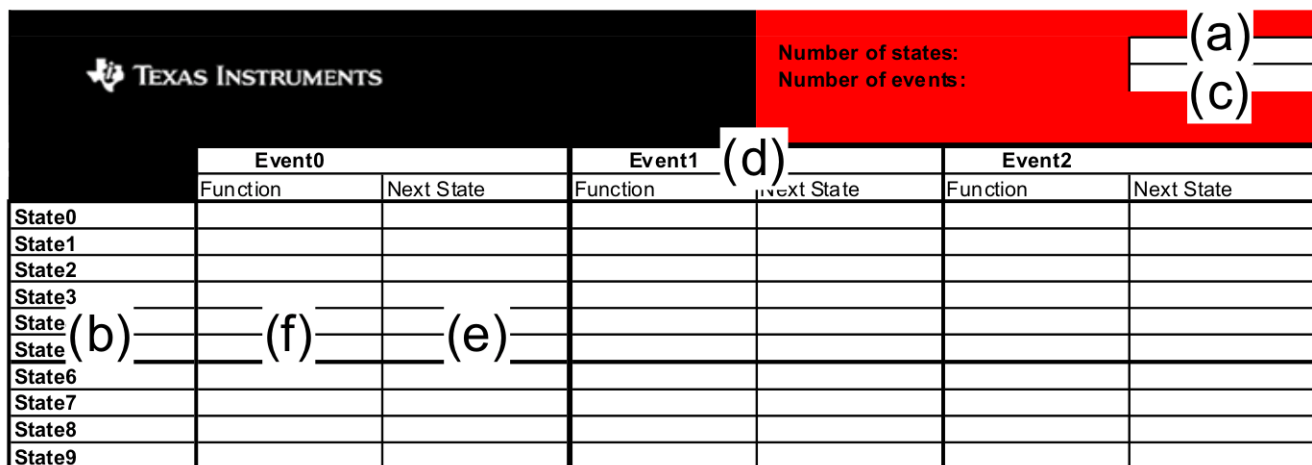
To make the use of finite state machines easier in MSP430-based systems, a Microsoft Excel based code-generation tool, called FSMGenerator, is included with this application report. To define a finite state machine, the FSMGenerator uses a state table. This table contains fields to specify the states, events, transitions, and transition actions and is, therefore, an absolutely equivalent representation of state diagrams. Based on a state table, code files can be generated automatically. These code files can be easily integrated in an application.

In the following sections, defining the state diagram in Figure 3 using the FSMGenerator is explained.

2.1 How to Use the FSMGenerator

First, open the code generation tool (FSMGenerator.xls) with Microsoft Excel. For full functionality, macros must be enabled. Furthermore, it is recommended to make a backup copy of the original code-generation tool before starting to edit it.

The state table used by the FSMGenerator is shown in Figure 4.



	Event0		Event1 (d)		Event2	
	Function	Next State	Function	Next State	Function	Next State
State0						
State1						
State2						
State3						
State (b)	(f)	(e)				
State4						
State5						
State6						
State7						
State8						
State9						

Figure 4. FSMGenerator State Table

To define a finite state machine, the table in [Figure 4](#) is filled out in four steps:

1. Defining the states

The first step is to define all states in which the state machine may ever be placed. Specify the number of states in text box (a). The number of states is limited to 32. By default, the states are named "State0" to "State31" (b). To rename the states, overwrite their names. The top state is always the initial state.

2. Defining the events

The next step is to define all events that can ever occur. Proceed similarly to step 1: Specify the number of events in text box (c). The number of events is limited to 32 as well. By default, the events are named "Event0" to "Event31" (d). To rename the events, overwrite their names.

3. Defining the transitions

A transition is a change from one state to another state, triggered by an event. The next state can be selected from a dropdown list of every possible next state (e). Every transition must be defined. If a transition should not move the finite state machine to a next state, it must be defined to loop back to the same state.

4. Defining the actions

During a state change, a transition action might be performed. In this case, write the name of the transition action in the function field (f). If no transaction action performed, the field can be left blank.

The complete state table for the demonstration application can be found in the zip folder as part of this application report.

2.2 Generating C Code

On the basis of a state table defined with FSMGenerator, it is possible to automatically generate C code by pressing the "Generate Code" button. Three code files are generated and saved in the folder where FSMGenerator.xls is located. Any existing generated files are backed up and are not overwritten.

- fsm.c

This file contains the state table and the event functions that must be called if the corresponding event occurs. This file should not be modified by the user.

- fsm_transition.c

For every transition action defined in the state table, a function prototype is generated. This transition function must be filled with the required functionality.

- fsm.h

This file is the common header file of the finite state machine. It must be included in any C file that interacts with the finite state machine.

2.3 Integration in an Application

Once the code files are generated, they can be integrated in an application. Therefore, the generated code files need to be included either in a CCE project or in an IAR project.

To interact with the finite state machine:

- Call event functions

If an event occurs (triggered either by an interrupt or by polling) the corresponding event function must be called. Therefore, the transition function defined in the state table is called automatically.

- Add code into the transition functions

The automatically generated prototypes of the transition functions must be filled with the desired functionality. Note that the transition functions are performed during a state change; therefore, the time spent in every transition function should be kept as short as possible.

3 Run Demo

The complete software of the game described here can be found in the zip file that is available with this application report. It is designed to run on the MSP-EXP430F5438 experimenter board. Therefore, open either the CCE or the IAR project, compile it, and load the application on the experimenter board. The aim of the demonstration game is to bring the numbers on the display into the right order by balancing the board into the right direction so that a number can slide into the free field. The detection is done with the accelerometer on the board.

4 State Machine Development Steps

To apply the finite state machine concept to another application, follow these steps:

1. Review the problem. Write down all the entities involved.
2. Design a state diagram from the work done in step (1).
3. Review the state diagram. Make sure that it meets all the requirements and does not fail under any transitions.
4. Develop a state table to clarify the state diagram and correlate to the code.
5. Generate code with FSMGenerator and integrate it in the target application.

5 Summary

This application report shows how the behavior of software can be elegantly described with the help of the finite state machine concept. The finite state machine concept is particularly beneficial for control-logic-oriented applications such as metering, monitoring, and control applications where reliability, size, and deterministic execution are the main criteria. A significant advantage of using the finite state machine concept is that the stepwise procedure to define an application leads to fewer errors and easier debugging.

In this application report, a simple game is implemented as an example of a finite state machine. This concept can be also easily applied to other applications. Therefore, an easy-to-use Microsoft Excel based code generation tool is presented.

Revision History

Revision	Comments
SLAA402	Initial release
SLAA402A	No changes to document; associated code files updated (slaa402.zip)

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2009, Texas Instruments Incorporated