State Machine Entry
Version 1.2
September 1987 P25-01700-02

Changes are made periodically to the information contained in this manual. These changes will be incorporated into subsequent editions.

Altera Corporation
3525 Monroe Street
Santa Clara, CA 95051
(408) 984-2800
TELEX: 888496

A+PLUS, SAM+PLUS, LogicMap, Turbo-Bit, MacroMuncher, SAM, BUSTER, EP310, EP320, EP600, EP610, EP900, EP910, EP1210, EP1800, EPB1400, EPS444, and EPS448 are trademarks of Altera Corporation. LogiCaps is a registered trademark of Altera. WordStar is a registered trademark of MicroPro Corporation. Fido is a trademark of Tom Jennings. MS-DOS is a trademark of Microsoft Corporation. FutureNet DASH is a trademark of FutureNet Corporation. PC-CAPS and PC-LOGS are trademarks of Personal CAD Systems, Inc. IBM Personal Computer is a registered trademark of International Business Machines Corporation.

Read This First...

The documentation for State Machine Entry (SMV) version 1.2 contains the following sections:

- State Machine Entry
- SMV Messages

Please insert them in your ***A+PLUS User Guide*** after the tab **II. Design Entry**.

At the back of the package, you will find a Warranty Card. Please fill out the Warranty Card, insert it into the Registration Envelope with the Altera Registration Card, and mail it to Altera. You will receive future update information for State Machine Entry software *only* if you mail this card.

Manual Updates

Altera documentation is updated with Change Pages, Section Reprints, and a **READ.ME** file.

Change Pages are issued for minor changes to the manual. New information is identified with vertical change bars in the margins next to the changed text. In addition, the date of issue is printed at the bottom of each page.

Section Reprints are issued if a section requires a substantial number of changes. The date of issue is indicated at the bottom of each page.

A **READ.ME File** is provided on the A+PLUS **INSTALL** diskette. This file contains information about recent changes to the State Machine Converter software that are not yet reflected in the manual.

Contents

State Machine Converter Messages

Illustrations

Tables

State Machine Entry

This package describes the process of creating designs with state machines. It includes a description of general requirements for entering State Machine Files and a functional description of Altera's State Machine software. In addition, you will find three sample sessions illustrating the steps used in designing with state machines; a list of design guidelines; and a reference section detailing Altera's State Machine File syntax.

Who Should Use State Machine Design Entry?

This design entry method is convenient for design engineers who are familiar with one or more of the basic methods for describing the logical operation of state machine designs, namely, state diagrams, operational flowcharts (e.g., algorithmic state machine (ASM) charts), truth tables, and Boolean expressions.

General Requirements

To enter your state machine design with A+PLUS, you need:

- A text editor that uses standard ASCII character conventions. (If your word processor has both document and non-document modes, use only the non-document mode.)
- A file with the extension .SMF.

Refer to *Installation* in the ***A+PLUS User Guide*** for instructions on how to install the State Machine Converter.

Functional Description

Altera's State Machine software enables you to enter and edit a state machine design with a standard text editor. Multiple state machine designs may be entered in a single State Machine File (SMF), which has a flexible syntax that allows you to define states and state transitions in several different formats. The SMF format also allows you to enter netlist statements containing primitives, Boolean equations, and truth tables; and to specify an asynchronous clear input and the state outputs for your design.

The State Machine software performs automatic flipflop selection. It also supports an optional automatic part selection feature, which instructs the Altera Design Processor to select the Altera EPLD that is best suited to your design requirements.

After you have entered your design into the SMF format, the A+PLUS State Machine Converter (SMV) translates it into an Altera Design File (ADF) when the Altera Design Processor (ADP) is invoked. You may also run the State Machine Converter (SMV) directly from DOS (stand-alone mode). (Refer to *A+PLUS and ADP Reference* in the ***A+PLUS Reference Guide*** for instructions.)

If an error is encountered during the SMF-to-ADF conversion, the SMV displays a message specifying the cause of the error and terminates design processing. The ADP then processes the ADF into a standard JEDEC file. Finally, the LogicMap II program uses the JEDEC file to program an Altera EPLD.

Figure SM-1 shows a block diagram of this process.

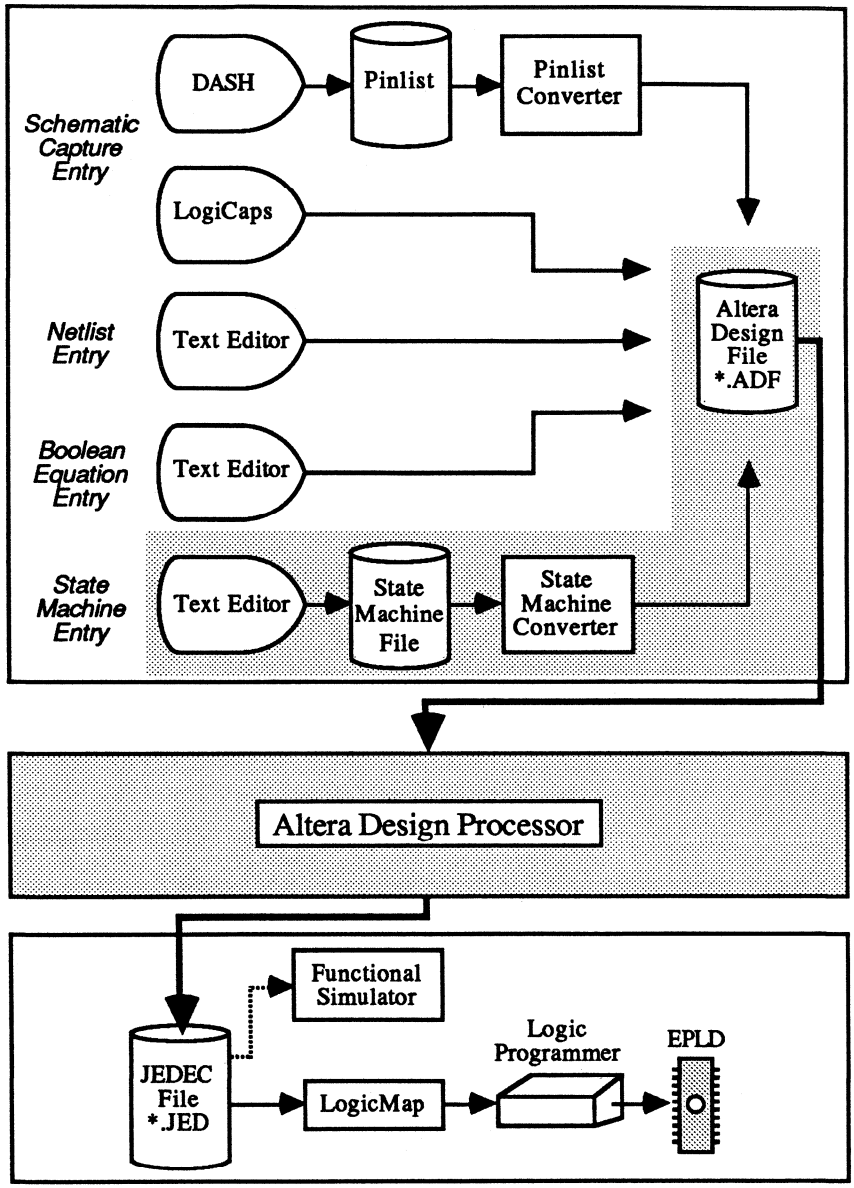


Figure SM-1. State Machine Entry

Design Steps

Every module of a logic system can be defined by its inputs, outputs, and transitory states. After determining the inputs and outputs, you can create your design with a state machine by following the steps listed below. Depending on your own preferences, you may perform these steps in a different sequence or choose to skip a step. The following pages contain three sample sessions that illustrate each of the suggested design steps.

The suggested design steps are:

1. Define the design inputs and outputs.
2. Draw a state diagram or an ASM chart.
3. Define state variables and assign state values.
4. Create a truth table or a state table (optional).
5. Enter the design into the SMF format with your text editor.
6. Save the design in a file with the filename extension **.SMF**.
7. Submit the design file to the ADP.
8. Program the EPLD.

Sample Sessions

This section contains three sample sessions that demonstrate the suggested sequence of steps for entering designs with state machines. For additional information on state machine design entry, refer to *Design Guidelines* and *State Machine File Format*.

Example 1

This sample circuit, called EXAMPLE1, controls the operation of a coin-activated beverage dispenser. Figure SM-2 shows a state diagram for EXAMPLE1. The mechanism has three inputs and two outputs. EXAMPLE1 advances from one state (State 1) to another state (State 2), to another state (State 3), and then begins again. A transitory fourth state (State 4) is also possible if the machine becomes jammed.

Step 1 – Define Inputs and Outputs:

First, you must define the inputs and outputs of your design. In this case, you have three inputs (**COINDROP**, **CUPFULL**, and **RESET**) and two outputs (**DROPCUP** and **POURDRNK**). You also need to define output dependencies on inputs and the cycle sequence needed to perform a specified function.

Step 2 – Draw a State Diagram or ASM Chart:

Next, you draw a state diagram or an algorithmic state machine (ASM) chart. The state machine diagram and chart both serve the same purpose of defining your algorithm. Each method is described below.

The State Diagram

A state diagram uses circles and arrows to represent the design. Circles represent states; arrows represent transitions between states. The diagram shows the sequence of states and outputs that occur in response to the inputs, i.e., the status of the outputs for each state and the effect of the inputs on the transitions. In State 1, the machine is in standby condition and both inputs are low. When **COINDROP** goes high, the machine goes to State 2 and then to State 3. State 4 occurs if both **DROPCUP** and **POURDRNK** are asserted. In this case, a Clear is initiated to force the unit to return to State 1. (Refer to Figure SM-2.)

When you develop a state diagram, you should label the states and transitions. In complex state machines, it is actually better to assign symbolic names rather than numbers to the states, in case future design modifications make it necessary to change the transitions or the state register values.

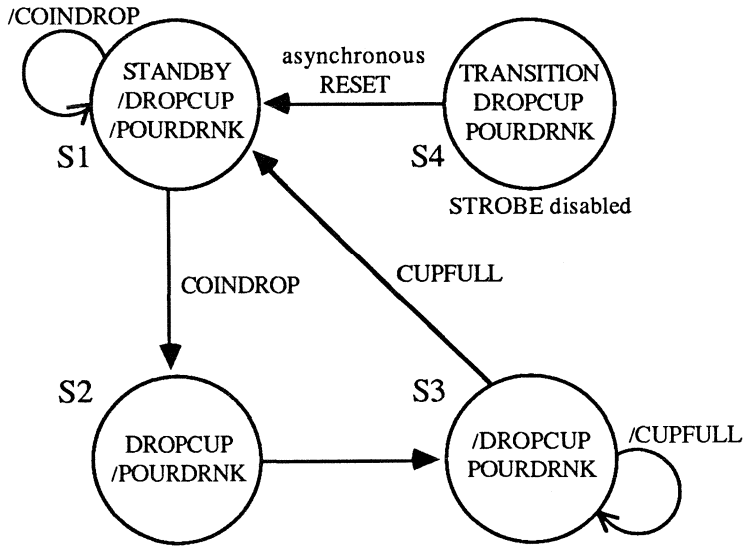


Figure SM-2. State Diagram for EXAMPLE1

The Algorithmic State Machine (ASM) Chart

The standard symbols used in an ASM chart are: the state box, which represents a single state; the decision box, which represents a single decision; and the conditional output box, which describes other outputs that are dependent on one or more inputs. State names are represented within circles. Figure SM-3 shows an ASM chart for EXAMPLE1.

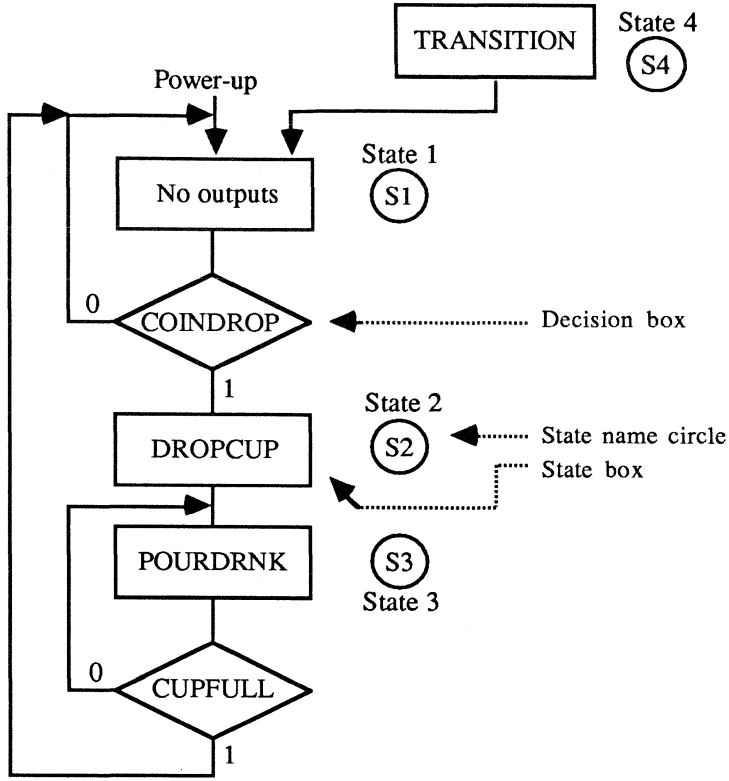


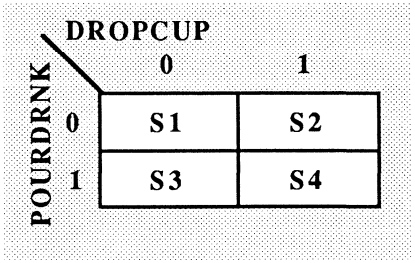
Figure SM-3. ASM Chart for EXAMPLE1

Step 3 – Define State Variables and Assign State Values:

Next, you assign the state variables. Each state requires a unique pattern of state variable values. (The feedbacks from the state registers are state variables.) In EXAMPLE1, the only outputs required are the circuit signals themselves. (In other designs, additional outputs may be specified for each state.)

The following are allowed: (1) state variables may be outputs of the circuit, or (2) states may have outputs, or (3) both.

Figure SM-4 shows the state variable assignments.



Explanation:

If both **DROPCUP** *and* **POURDRNK** *are not activated (i.e., 0 or not true), the current state is State 1. If* **DROPCUP** *is activated (i.e., true or 1) and* **POURDRNK** *is not activated, the current state is State 2. If* **DROPCUP** *is not activated and* **POURDRNK** *is activated, the current state is State 3. If both* **DROPCUP** *and* **POURDRNK** *are activated, the current state is State 4.*

Figure SM-4. State Variable Assignments for EXAMPLE1

Step 4 – Generate a Truth Table:

You may wish to generate a truth table with entries for all available states, combinations of inputs, and the resulting next states. Table SM-1 shows the truth table generated for the beverage dispenser controller circuit. (In a more complex design, the "Don't Care" entries in the Inputs Section allow greater flexibility in the logic reduction than explicit definitions of all combinations.)

Table SM-1. Truth Table for EXAMPLE1

INPUTS		PRESENT STATE		NEXT STATE	
COIN-DROP	CUP-FULL	DROP-CUP	POUR-DRNK	DROP-CUP	POUR-DRNK
0	X	0	0	0	0
1	X	0	0	1	0
X	X	1	0	0	1
X	0	0	1	0	1
X	1	0	1	0	0
X	X	1	1	0	0
X = Don't Care 1 = True 0 = False					

Step 5 – Convert the Information into an SMF:

After completing the first four steps, you can convert the information into a State Machine File (SMF). Examples 1 through 3 walk you through this process, but be sure to consult *Design Guidelines* and *State Machine File Format*. Note that many sections of the SMF must be identified by a keyword and a colon (e.g., **PART:**).

The Altera Design Processor, which processes the SMF, offers automatic part and flipflop selection. Therefore, when you specify the part name in the Part Section of the SMF, you may enter either a specific EPLD name (such as **EP600**) or **AUTO** for automatic part selection. The State Machine Converter assumes that a part with T flipflops is available and the ADP will automatically choose the part most likely to fit your design.

If you wish to use a part without T flipflops—EP310, EP320, or EP1210—do not enter **AUTO**; instead, specify the part name in the Part Section.

For complete details on the SMF syntax, refer to *State Machine File Format*.

The SMF information is entered in the following sequence:

1. Specify the header. This section is optional. However, if it is present, it must be the first section in the file. The header includes the following information:

```
Your Name
Your Company
9/30/87
1
A
EP310
Beverage Dispenser
```

2. Specify options:

```
OPTIONS:    SECURITY = OFF
```

3. Specify the part name:

```
PART: EP310
```

4. Specify the inputs (comments must be enclosed by percent (%) symbols):

```
INPUTS:
% acknowledge dropped coin %
    COINDROP
% enable fill the cup %
    CUPFULL
% clock %
    CLK
% clear %
    RESET
```

5. Specify the outputs:

```
OUTPUTS:
% enable drop the cup %
    DROPCUP
% enable pour the drink %
    POURDRNK
```

6. Specify the name of the state machine and its subsections. This section is identified by the keyword **MACHINE:**.

```
MACHINE: dispenser
```

State Machine names may contain up to 32 alphanumeric characters and underscores (_). No blanks may be embedded in state machine names. Refer also to Table SM-2 in *Design Guidelines*.

Specify the clock in the Clock subsection:

```
CLOCK:  CLK
```

The clock name in a state machine is the name of the node feeding the state machine register. In a synchronous clock, this node is typically the left-hand side of an **INP** statement. If a state machine design requires an asynchronous clock that is driven directly by a pin, use a **CLKB** Primitive in the Network Section. (Refer to *Appendix A* in the ***A+PLUS Reference Guide***.)

Specify the asynchronous clear input in the optional Clear subsection. The clear input resets all state variable values to zero.

```
CLEAR:  RESET
```

Next, you specify the state assignments in the required States subsection. Each state assignment must be unique. (Refer to *Transition Syntax* for other syntax choices.)

```
STATES: [ DROPCUP  POURDRNK ]
    S1      [ 0          0 ]
    S2      [ 1          0 ]
    S3      [ 0          1 ]
    S4      [ 1          1 ]
```

State variables (**DROPCUP**, **POURDRNK**) represent feedbacks and, optionally, output pin names; therefore, they may never appear in the Outputs subsections that follow Transitions subsections. The entries in an Outputs subsection represent the results of the evaluation of a Boolean expression. They may be used anywhere the left-hand side of an equation may be used, e.g., as a D, T, JK, or SR input to an I/O primitive.

Next, define the transitions and outputs. The syntax for conditional transitions is:

IF <expression> THEN <next state>

where each **IF-THEN** string represents an arrow in the state diagram. The Transitions from a state must always precede the state's Outputs subsection, which is optional. (Note: EXAMPLE1 does not use Outputs subsections.) Refer to *Transition Syntax* for other syntax choices.

```
S1:
    IF COINDROP THEN S2
    % No asynchronous state outputs %
S2:
    S3 % unconditional transition %
S3:
    IF CUPFULL THEN S1
S4:
    S1 %unconditional transition %
```

Transitions are evaluated in the order in which they are entered, i.e., the first transition has precedence over the second, the second over the third, etc. If no transitions are specified, the next state is the same as the current state, i.e., the machine will hold the current state.

7. Terminate the file:

END$

Figure SM-5 shows the completed SMF, including explanatory notes.

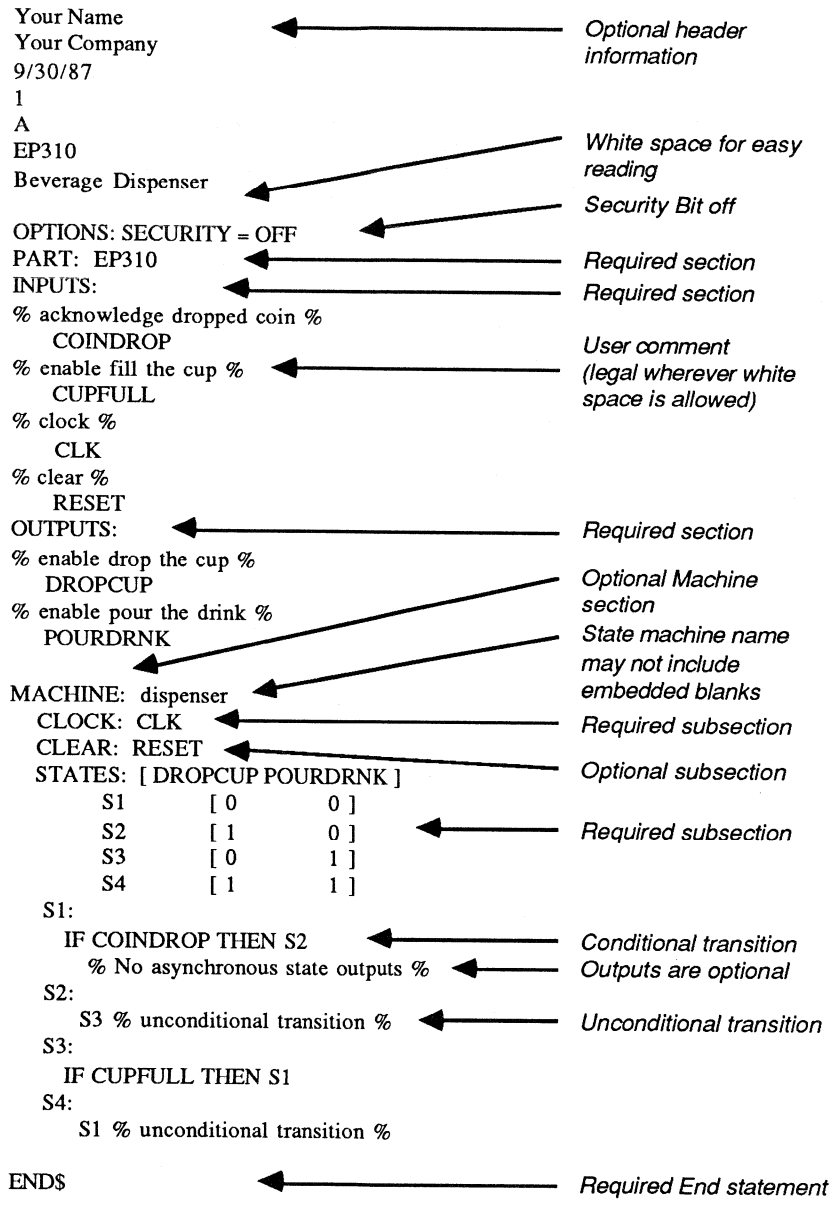


Figure SM-5. SMF for EXAMPLE1

Step 6 – Save the SMF:

Save the file under the legal DOS filename **EXAMPLE1.SMF** and return to DOS.

Step 7 – Submit the SMF to the ADP:

To submit the file **EXAMPLE1.SMF** to the ADP, you first display the APLUS Menu by typing from DOS:

APLUS <Enter>

Press **<F4>** to display the ADP Menu.

You are prompted to specify your form of input. Type:

S

(for State Machine File). Now answer the **<F4> File Name(s)** prompt by typing:

EXAMPLE1 <Enter>

You need not enter the filename extension; A+PLUS will automatically add the extension for you. Be sure to specify the correct pathname and directory.

You are then prompted through the remaining ADP Menu functions:

For **<F5> Minimization**, press **<Enter>**. (**Y** [Yes] is the default).
For **<F6> Inversion Control**, press **<Enter>**. (**N** [No] is the default).
For **<F7> LEF Analysis**, press **<Enter>**. (**N** [No] is the default).

After you have answered the sequential prompts of the ADP Menu, you are asked:

Do you wish to run under the above conditions [Y/N]?

Enter **Y** or press **<F8> (Execute)** to execute the ADP. During design processing, the State Machine Converter, the ADP, and individual ADP

modules will display information messages that report current processing status. When the design cycle is completed, you are asked:

Would you like to implement another design [Y/N}?

Enter **N**. You are returned to the APLUS Menu. For detailed information on the Altera Design Processor and the ADP Menu functions, refer to *A+PLUS and ADP Reference* in the ***A+PLUS Reference Guide***.

Figure SM-6 shows the Altera Design File (ADF) for EXAMPLE1. The State Machine Converter always produces an ADF from the State Machine File (SMF) prior to design processing.

```
Your Name
Your Company
9/30/87
1
A
EP310
Beverage Dispenser
<SMV Version information>
OPTIONS:    SECURITY = OFF
PART:  EP310

INPUTS:
COINDROP, CUPFULL, CLK, RESET

OUTPUTS:
DROPCUP, POURDRNK

NETWORK:
COINDROP  =  INP(COINDROP)
CUPFULL  =  INP(CUPFULL)
CLK  =  INP(CLK)
RESET  = INP(RESET)
%
I/O's for State Machine "dispenser"
%
DROPCUP, DROPCUP  =  RORF(DROPCUP.d, CLK, RESET, GND, VCC)
POURDRNK, POURDRNK  =  RORF(POURDRNK.d, CLK, RESET, GND, VCC)

EQUATIONS:
%
Boolean Equations for State Machine "dispenser"
%
%
Current State Equations for "dispenser"
%
S1  =  DROPCUP ' * POURDRNK ';
S2  =  DROPCUP * POURDRNK ';
S3  =  DROPCUP ' * POURDRNK;
S$  =  DROPCUP * POURDRNK;
%
SV Defining Equations for State Machine "dispenser"
%
DROPCUP.d  =  S2.n;
POURDRNK.d  =  S3.n;
%
Next State Equations for State Machine "dispenser"
%
S2.n   =  (S1 * COINDROP);
S3.n   =  (S3) * (CUPFULL) '
       +  (S2);

END$
```

Figure SM-6. EXAMPLE1.ADF

Step 8 – Program the EPLD:

Finally, you submit your design to LogicMap II. While still in the APLUS Menu, press **<F5>** to select LogicMap II. If the Logic Programmer card is plugged in, the program will come up on the screen. If not, the following message is displayed:

Programmer self test failed

Device must NOT be in socket for this test to pass!

Enter:
C to continue without programming card
T to run diagnostics again
Q to return to operating system

When the LogicMap II System Level Window is displayed, you are asked to wait until the calibration process has been completed. Then the System Level HELP Window is opened.

Do not put the EPLD into the socket of the programming unit until you are prompted to do so.

Select **Program Device** with the box cursor and enter the filename **EXAMPLE1**. When you are prompted to:

Select Device for Programming

enter

EP310 <Enter>

LogicMap II automatically checks whether the EPLD is erased and ready for programming. After you have answered all the prompts, programming time is approximately five to ten seconds for this device. (For complete information on device programming, refer to the ***LogicMap II*** manual.)

This concludes the first sample session.

Example 2

Example 1 introduced you to the basic steps necessary to create and enter a design with a state machine. The second example, called EXAMPLE2, guides you through a similar sequence of steps for a more complex example. Be sure to familiarize yourself with the *Design Guidelines* and *State Machine File Format* for detailed information on the SMF format and syntax. Note that the SMF format is similar to the ADF format.

EXAMPLE2 consists of a four-phase stepper motor controller. Figure SM-7 shows a state diagram for EXAMPLE2. The stepper motor can be rotated clockwise or counterclockwise, depending on the sequence of signals presented to its windings. This sequence is controlled by the inputs **CW** (clockwise), **CCW** (counterclockwise), and **HS** (halfstep). The stepper motor is designed for 7.5 degree steps, i.e., each time a state changes, the motor rotates 7.5 degrees. By asserting the halfstep input, the state machine generates intermediate steps, enabling the motor to move in 3.75 degree increments.

Step 1 – Define Inputs and Outputs:

INPUTS: CW, CCW, HS
OUTPUTS: PH1, PH2, PH3, PH4 (phases 1 through 4)

Step 2 – Draw a State Diagram:

The design has four unique states per cycle and four intermediate (half) steps. (Refer to Figure SM-7.)

Altera EPLDs power up with all state variables at logic zero. Be sure to identify the power-up state of the device register. If the register powers up in an unidentified state, the next state is also unidentified. Unless this is your intention, you should ensure that your design goes to a defined state at power-up.

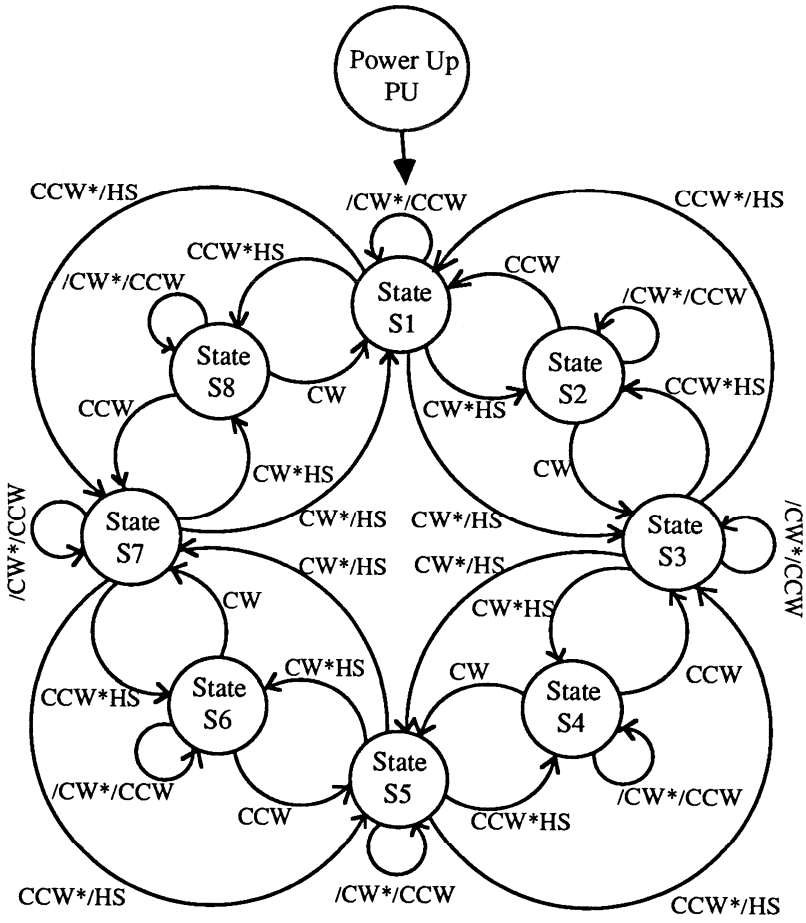


Figure SM-7. State Diagram for EXAMPLE2

Step 3 – Define State Variables and Assign State Values:

The feedbacks from the state register are state variables. In this design, the values of the state variables are also presented to the output pins of the design, i.e., the variables are direct outputs of the design. Although this condition is not possible for all designs, it is highly desirable because it allows you to save a large number of macrocells. State variables defined for this example are as follows:

```
STATES:   [  PH1   PH2   PH3   PH4   ]
```

State assignments are as follows:

```
PU        [  0     0     0     0   ]
S1        [  1     0     1     0   ]
S2        [  1     0     0     0   ]
S3        [  1     0     0     1   ]
S4        [  0     0     0     1   ]
S5        [  0     1     0     1   ]
S6        [  0     1     0     0   ]
S7        [  0     1     1     0   ]
S8        [  0     0     1     0   ]
```

Step 4 – Define State Transitions and Outputs:

By examining Figure SM-7, you can derive the following transition statements for EXAMPLE2:

```
PU:
     S1
S1:
     IF CW * /HS THEN S3
     IF CW * HS THEN S2
     IF CCW * /HS THEN S7
     IF CCW * HS THEN S8
S2:
     IF CW THEN S3
     IF CCW THEN S1
```

```
S3:
     IF CW * /HS THEN S5
     IF CW * HS THEN S4
     IF CCW * /HS THEN S1
     IF CCW * HS THEN S2
S4:
     IF CW THEN S5
     IF CCW THEN S3
S5:
     IF CW * /HS THEN S7
     IF CW * HS THEN S6
     IF CCW * /HS THEN S3
     IF CCW * HS THEN S4
S6:
     IF CW THEN S7
     IF CCW THEN S5
S7:
     IF CW * /HS THEN S1
     IF CW * HS THEN S8
     IF CCW * /HS THEN S5
     IF CCW * HS THEN S6
S8:
     IF CW THEN S1
     IF CCW THEN S7
```

Step 5 – Convert the Information into an SMF:

You now have all the information needed to generate the SMF. Figure SM-8 shows the SMF for EXAMPLE2, including explanatory notes.

Since the ADP offers automatic part and flipflop selection, you may specify the part name in the Part Section of the SMF (in this case, EP900) or type AUTO. The State Machine Converter assumes that a part with T flipflops is available and the ADP will automatically choose the part most likely to fit your design.

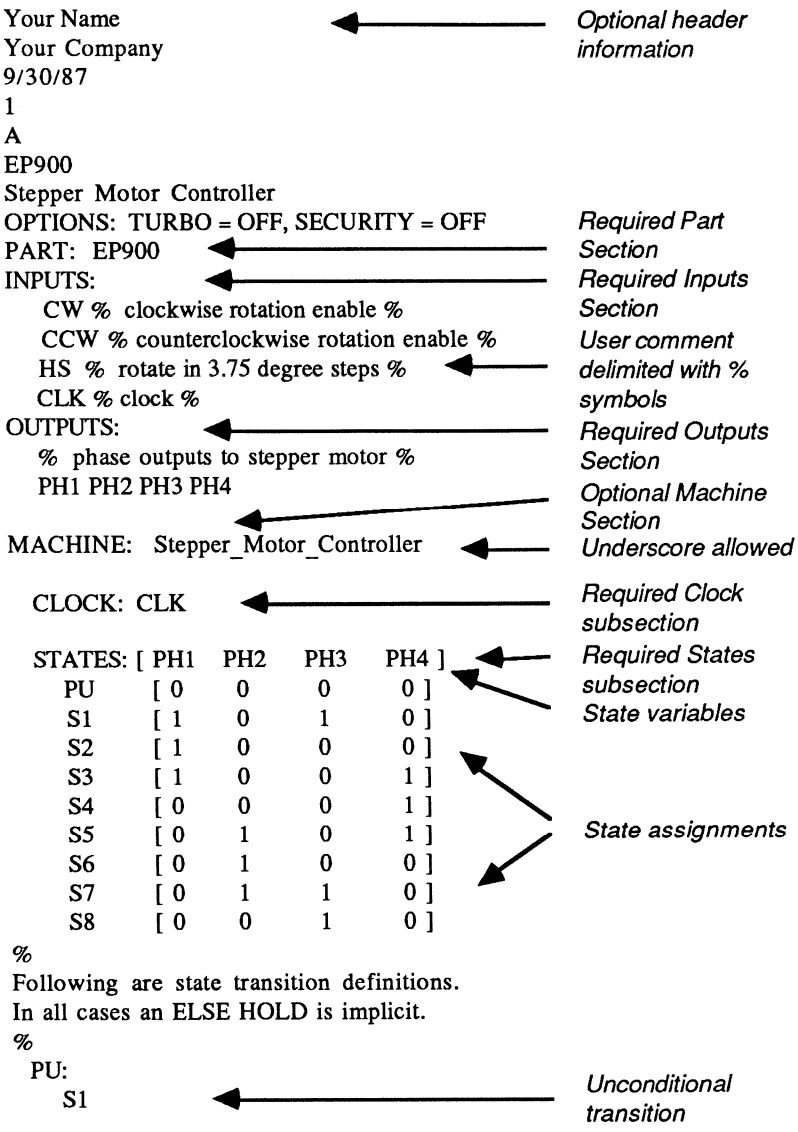


Figure SM-8. SMF for EXAMPLE2 (Part 1 of 2)

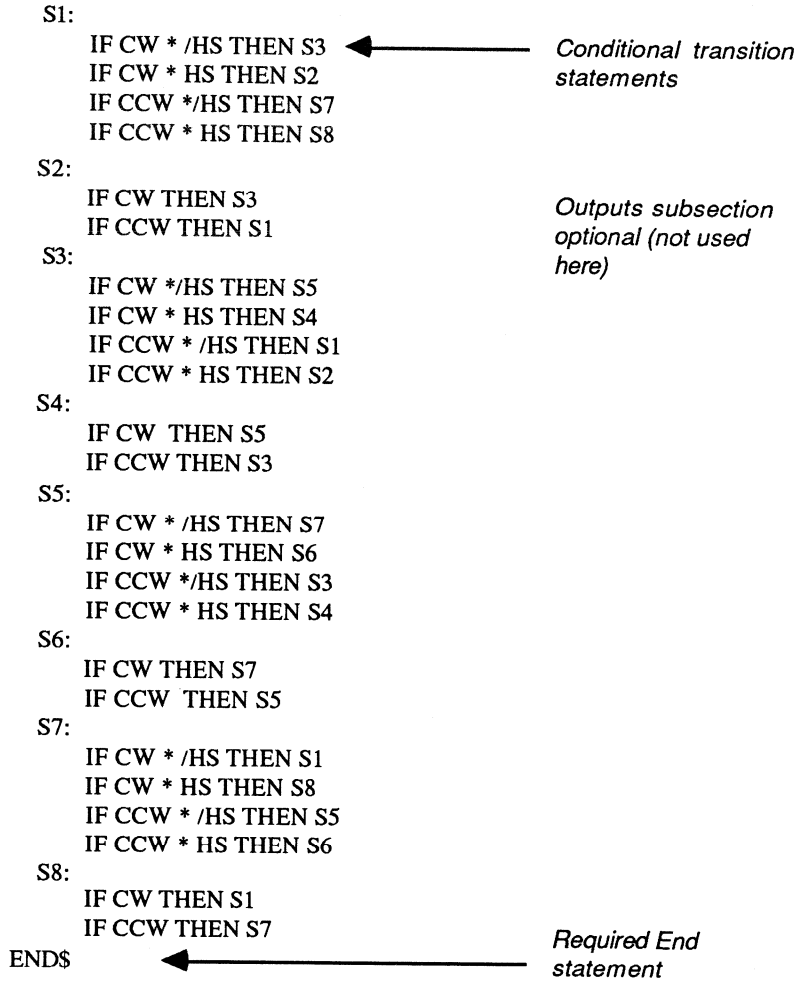


Figure SM-8. SMF for EXAMPLE2 (Part 2 of 2)

Step 6 – Save the SMF:

Save the file under the legal DOS filename **EXAMPLE2.SMF** and return to DOS.

Step 7 – Submit the SMF to the ADP:

Submit the file to the ADP as described in *Step 7* of *Example 1*, but press **Y** (Yes) at the **<F7> LEF Analysis** prompt. Figure SM-9 shows the Logic Equation File (LEF) generated by the ADP after **EXAMPLE2.SMF** has been converted into an ADF, translated, expanded, minimized, and analyzed by the LEF Analyzer.

Step 8 – Program the EPLD:

Program the EPLD as described in *Step 8* of *Example 1*.

This concludes the second sample session.

```
Your Name
Your Company
9/30/87
1
A
EP900
Stepper Motor Controller
<SMV Version information>
Input files : EXAMPLE2.ADF
ADP Options: Minimization = Yes, Inversion Control = No, LEF Analysis = Yes

<LEF Version information>
OPTIONS:  TURBO = OFF, SECURITY = OFF
PART:
    EP900
INPUTS:
    CW, CCW, HS, CLK
OUTPUTS:
    PH1, PH2, PH3, PH4
NETWORK:
    CLK = INP(CLK)
    CW = INP(CW)
    CCW = INP(CCW)
    HS = INP(HS)

    %
    The primitive JOJF was minimized to TOTF
    %
    PH1, PH1  =  TOTF(PH1.j.1, CLK, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    PH2, PH2  =  TOTF(PH2.j.3, CLK, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    PH3, PH3  =  TOTF(PH3.j.5, CLK, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    PH4, PH4  =  TOTF(PH4.j.7, CLK, GND, GND, VCC)
```

Figure SM-9. EXAMPLE2.LEF (Part 1 of 2)

```
EQUATIONS:
    PH4.j.7  =  PH4' * PH1 * PH2' * CW * HS'
             +  PH4' * PH1 * PH2' * PH3' * CW
             +  PH4 * PH1' * PH2 * PH3' * CW
             +  PH4' * PH1' * PH2 * CW' * HS' * CCW
             +  PH4' * PH1' * PH2 * PH3' * CW' * CCW
             +  PH4 * PH1 * PH2' * PH3' * CW' * CCW;

    PH3.j.5  =  PH3' * PH1' * PH2' * PH4'
             +  PH3' * PH1' * PH4' * CW
             +  PH3' * PH2' * PH4' * CCW * CW'
             +  PH3' * PH1' * PH2 * HS' * CW
             +  PH3 * PH1 * PH2' * PH4' * CW
             +  PH3' * PH1 * PH2' * CCW * HS' * CW'
             +  PH3 * PH1' * PH2 * PH4' * CCW * CW'

    PH2.j.3  =  PH2' * PH3' * PH4 * CW * HS'
             +  PH2' * PH1' * PH3' * PH4 * CW
             +  PH2 * PH1' * PH3 * PH4' * CW
             +  PH2' * PH3 * PH4' * CCW * CW' * HS'
             +  PH2' * PH1' * PH3 * PH4' * CCW * CW'
             +  PH2 * PH1' * PH3' * PH4 * CCW * CW';

    PH1.j.1  =  PH1' * PH2' * PH3' * PH4'
             +  PH1' * PH2' * PH4' * CW
             +  PH1' * PH2' * PH3' * CW' * CCW
             +  PH1' * PH3 * PH4' * CW * HS'
             +  PH1 * PH2' * PH3' * PH4 * CW
             +  PH1' * PH3' * PH4 * CW' * HS' * CCW
             +  PH1 * PH2' * PH3 * PH4' * CW' * CCW;
END$
```

Figure SM-9. EXAMPLE2.LEF (Part 2 of 2)

Example 3

Before going through this example, be sure to read *Design Guidelines* and *State Machine File Format*.

The third example, called EXAMPLE3, describes a gray code counter and display decoder. Such counters may be used for applications in which the decoded outputs of the counter drive circuitry that requires high reliability. In contrast to binary counters, only one bit changes per transition in gray code counters.

The EXAMPLE3 circuit is a four-bit gray counter driving a seven-segment decoder for display. The outputs of the counter are available on the EPLD pins. This design uses a state machine as well as a truth table. Figure SM-10 shows a state diagram for EXAMPLE3.

Step 1 – Define Inputs and Outputs:

The design has two inputs: **hold, g4clk**

The outputs are the state variables, as well as outputs to a pin:

q1, q2, q3, q4

The design also has seven segment decode outputs:

Saa, Sbb, Scc, Sdd, See, Sff, Sgg

Step 2 – Draw a State Diagram:

Figure SM-10 shows the state diagram for the gray code counter. Each state contains the following information:

- State Name
- State Assignment
- Decoder Output

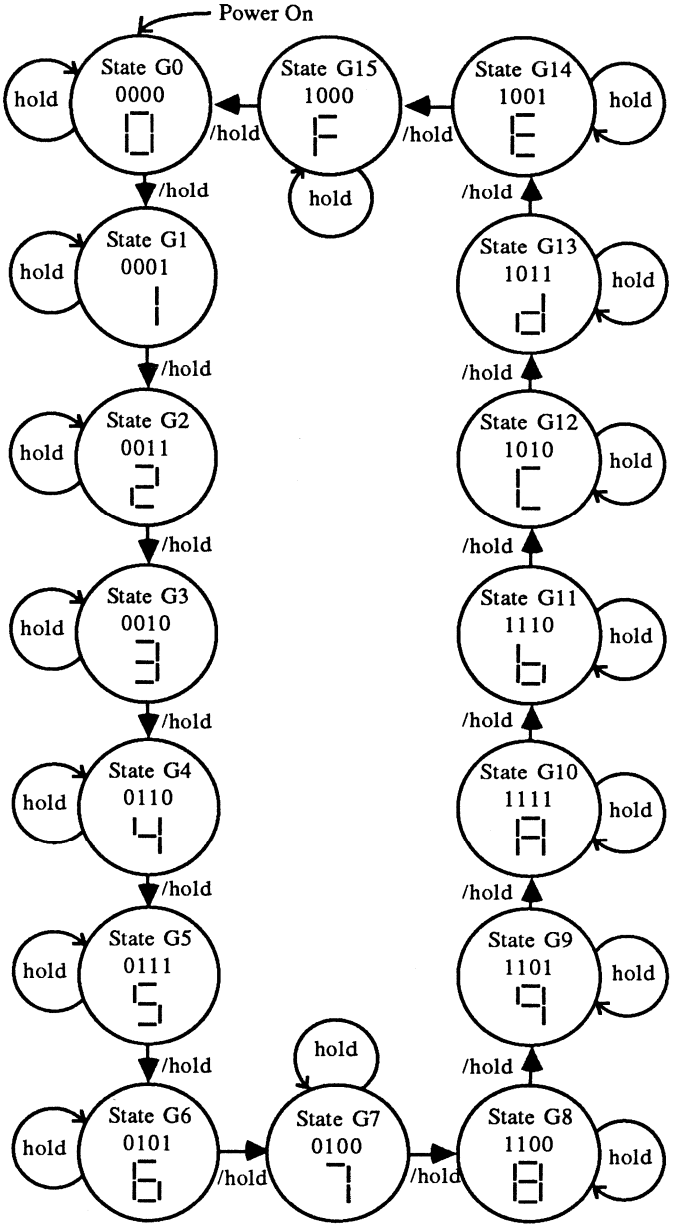


Figure SM-10. State Diagram for EXAMPLE3

Step 3 – Define State Variables and Assign State Values:

As indicated above, the outputs are also the state variables:

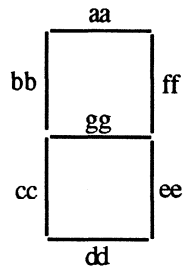
STATES: [q1 q2 q3 q4]

State assignments are as follows (they are also shown in Figure SM-10):

G0	[	0	0	0	0]
G1	[	0	0	0	1]
G2	[	0	0	1	1]
G3	[	0	0	1	0]
G4	[	0	1	1	0]
G5	[	0	1	1	1]
G6	[	0	1	0	1]
G7	[	0	1	0	0]
G8	[	1	1	0	0]
G9	[	1	1	0	1]
G10	[	1	1	1	1]
G11	[	1	1	1	0]
G12	[	1	0	1	0]
G13	[	1	0	1	1]
G14	[	1	0	0	1]
G15	[	1	0	0	0]

Step 4 – Assign the Segments:

The assignments for the display decoder are as follows:



Step 5 – Create a Truth Table:

The truth table is external to the state machine. However, it may be described in the SMF in the optional Truth Table Section (see *State Machine File Format*). The truth table for EXAMPLE3 is shown below (note that 0 indicates that a segment is lighted, 1 that it is not; this indicates the direct connection to the common anode displays where all anodes are tied to VCC):

T_TAB:	q4	q3	q2	q1	:	aa	bb	cc	dd	ee	ff	gg	;
% 0 %	0	0	0	0	:	0	0	0	0	0	0	1	;
% 1 %	0	0	0	1	:	1	0	0	1	1	1	1	;
% 2 %	0	0	1	1	:	0	1	0	0	1	0	0	;
% 3 %	0	0	1	0	:	0	1	1	0	0	0	0	;
% 4 %	0	1	1	0	:	1	0	1	1	0	0	0	;
% 5 %	0	1	1	1	:	0	0	1	0	0	1	0	;
% 6 %	0	1	0	1	:	0	0	0	0	0	1	0	;
% 7 %	0	1	0	0	:	0	1	1	1	0	0	1	;
% 8 %	1	1	0	0	:	0	0	0	0	0	0	0	;
% 9 %	1	1	0	1	:	0	0	1	1	0	0	0	;
% A %	1	1	1	1	:	0	0	0	1	0	0	0	;
% b %	1	1	1	0	:	1	0	0	0	0	1	0	;
% C %	1	0	1	0	:	0	0	0	0	1	1	1	;
% d %	1	0	1	1	:	1	1	0	0	0	0	0	;
% E %	1	0	0	1	:	0	0	0	0	1	1	0	;
% F %	1	0	0	0	:	0	0	0	1	1	1	0	;

Step 6 – Convert the Information into an SMF:

Figure SM-11 shows the SMF generated from the information determined so far:

You may specify the part name in the Part Section of the SMF (in this case, **EP600**) or type **AUTO**.

Step 7 – Save the SMF:

Save the file under the legal DOS filename **EXAMPLE3.SMF** and return to DOS.

Step 8 – Submit the SMF to the ADP:

Submit the file to the ADP as described in *Step 7* of *Example 1*, but press **Y** (Yes) at the **<F7> LEF Analysis** prompt. Figure SM-12 shows the Logic Equation File (LEF) generated by the ADP after **EXAMPLE3.SMF** has been converted into an ADF, translated, expanded, minimized, and analyzed by the LEF Analyzer.

Step 9 – Program the EPLD:

Program the EPLD as described in Step 8 of *Example 1*.

This concludes the third sample session.

```
Your Name
Your Company
9/30/87
1
A
EP600
4 Bit Gray Code UP Counter

OPTIONS:  TURBO = OFF, SECURITY = OFF
PART:    EP600

INPUTS:    hold, g4clk

OUTPUTS:
     % gray code counter outputs; q1 is least significant bit %
     q1, q2, q3, q4
     % 7 segment decode outputs %
     Saa, Sbb, Scc, Sdd, See, Sff, Sgg

NETWORK:
     % combinatorial outputs for the 7 segment display %
     Saa    = CONF(aa, VCC)
     Sbb    = CONF(bb, VCC)
     Scc    = CONF(cc, VCC)
     Sdd    = CONF(dd, VCC)
     See    = CONF(ee, VCC)
     Sff    = CONF(ff, VCC)
     Sgg    = CONF(gg, VCC)

MACHINE:   Gray4
     % this defines the State Machine %

     CLOCK:    g4clk
     STATES:   [ q4   q3   q2   q1  ]
       G0      [ 0    0    0    0   ]
       G1      [ 0    0    0    1   ]
       G2      [ 0    0    1    1   ]
       G3      [ 0    0    1    0   ]
       G4      [ 0    1    1    0   ]
       G5      [ 0    1    1    1   ]
       G6      [ 0    1    0    1   ]
       G7      [ 0    1    0    0   ]
       G8      [ 1    1    0    0   ]
       G9      [ 1    1    0    1   ]
       G10     [ 1    1    1    1   ]
```

Figure SM-11. SMF for EXAMPLE3 (Part 1 of 3)

```
G11     [ 1   1   1   0   ]
G12     [ 1   0   1   0   ]
G13     [ 1   0   1   1   ]
G14     [ 1   0   0   1   ]
G15     [ 1   0   0   0   ]

% state transition definitions %
% in all cases ELSE HOLD is implicit %
G0:
    IF /hold THEN G1
G1:
    IF /hold THEN G2
G2:
    IF /hold THEN G3
G3:
    IF /hold THEN G4
G4:
    IF /hold THEN G5
G5:
    IF /hold THEN G6
G6:
    IF /hold THEN G7
G7:
    IF /hold THEN G8
G8:
    IF /hold THEN G9
G9:
    IF /hold THEN G10
G10:
    IF /hold THEN G11
G11:
    IF /hold THEN G12
G12:
    IF /hold THEN G13
G13:
    IF /hold THEN G14
G14:
    IF /hold THEN G15
G15:
    IF /hold THEN G0
% 7 segment display decoder; note: 0=> segment ON %
```

Figure SM-11. SMF for EXAMPLE3 (Part 2 of 3)

```
% segment assignment:
        aa
     bb   ff
        gg
     cc   ee
        dd
%
% inputs of truth table                outputs of truth table %
 T_TAB: q4  q3  q2  q1  :  aa  bb  cc  dd  ee  ff  gg  ;
  % 0 %  0   0   0   0   :  0   0   0   0   0   0   1   ;
  % 1 %  0   0   0   1   :  1   0   0   1   1   1   1   ;
  % 2 %  0   0   1   1   :  0   1   0   0   1   0   0   ;
  % 3 %  0   0   1   0   :  0   1   1   0   0   0   0   ;
  % 4 %  0   1   1   0   :  1   0   1   1   0   0   0   ;
  % 5 %  0   1   1   1   :  0   0   1   0   0   1   0   ;
  % 6 %  0   1   0   1   :  0   0   0   0   0   1   0   ;
  % 7 %  0   1   0   0   :  0   1   1   1   0   0   1   ;
  % 8 %  1   1   0   0   :  0   0   0   0   0   0   0   ;
  % 9 %  1   1   0   1   :  0   0   1   1   0   0   0   ;
  % A %  1   1   1   1   :  0   0   0   1   0   0   0   ;
  % b %  1   1   1   0   :  1   0   0   0   0   1   0   ;
  % C %  1   0   1   0   :  0   0   0   0   1   1   1   ;
  % d %  1   0   1   1   :  1   1   0   0   0   0   0   ;
  % E %  1   0   0   1   :  0   0   0   0   1   1   0   ;
  % F %  1   0   0   0   :  0   0   0   1   1   1   0   ;

END$
```

Figure SM-11. SMF for EXAMPLE3 (Part 3 of 3)

```
Your Name
Your Company
9/30/87
1
A
EP600
4-Bit Gray Code Up Counter

<SMV Version information>
Input files : EXAMPLE3.ADF
ADP Options: Minimization = Yes, Inversion Control = No, LEF Analysis =
Yes

<LEF Version information>

OPTIONS:  TURBO = OFF, SECURITY = OFF
PART:
    EP600
INPUTS:
    hold, g4clk
OUTPUTS:
    q1, q2, q3, q4, Saa, Sbb, Scc, Sdd, See, Sff, Sgg
NETWORK:
    g4clk  =  INP(g4clk)
    hold  =  INP(hold)

    %
    The primitive JOJF was minimized to TOTF
    %
    q1, q1  =  TOTF(q1.j.1, g4clk, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    q2, q2  =  TOTF(q2.j.3, g4clk, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    q3, q3  =  TOTF(q3.j.5, g4clk, GND, GND, VCC)

    %
    The primitive JOJF was minimized to TOTF
    %
    q4, q4  =  TOTF(q4.j.7, g4clk, GND, GND, VCC)
```

Figure SM-12. EXAMPLE3.LEF (Part 1 of 3)

```
    Saa    = CONF(aa, VCC)
    Sbb    = CONF(bb, VCC)
    Scc    = CONF(cc, VCC)
    Sdd    = CONF(dd, VCC)
    See    = CONF(ee, VCC)
    Sff    = CONF(ff, VCC)
    Sgg    = CONF(gg, VCC)
EQUATIONS:
    gg  = q4' * q3' * q2'
        + q4' * q2' * q1'
        + q4 * q3' * q2 * q1';

    ff  = q4 * q3' * q2'
        + q4' * q2' * q1
        + q4' * q3 * q1
        + q4 * q2 * q1';

    ee  = q4 * q3' * q2'
        + q4 * q3' * q1'
        + q4' * q3' * q1;

    dd  = q4' * q3 * q1'
        + q4 * q3 * q1
        + q4' * q3' * q2' * q1
        + q4 * q3' * q2'* q1';

    cc  = q4' * q3 * q1'
        + q4' * q2 * q1'
        + q4' * q3 * q2
        + q4 * q3 * q2' * q1;

    bb  = q4' * q3' * q2
        + q3' * q2 * q1
        + q4' * q3 * q2' * q1';

    aa  = q3 * q2 * q1'
        + q4' * q3' * q2' * q1
        + q4 * q3' * q2 * q1;

    q4.j.7   = q4 * q3' * q2' * q1' * hold'
             + q4' * q3 * q2' * q1' * hold';

    q3.j.5   = q3' * q4' * q2 * q1' * hold'
             + q3 * q4 * q2 * q1' * hold';
```

Figure SM-12. EXAMPLE3.LEF (Part 2 of 3)

```
q2.j.3    = q2' * q4' * q3' * q1 * hold'
          + q2 * q4 * q3' * q1 * hold';
          + q2 * q4' * q3 * q1 * hold'
          + q2' * q4 * q3 * q1 * hold';

q1.j.1    = q1' * q4' * q3' * q2' * hold'
          + q1 * q4 * q3' * q2' * hold'
          + q1 * q4' * q3 * q2' * hold'
          + q1 * q4' * q3' * q2 * hold'
          + q1' * q4 * q3' * q2 * hold'
          + q1' * q4 * q3 * q2' * hold'
          + q1' * q4' * q3 * q2 * hold'
          + q1 * q4 * q3 * q2 * hold';
END$
```

Figure SM-12. EXAMPLE3.LEF (Part 3 of 3)

Design Guidelines

The following guidelines apply to State Machine Entry:

- A State Machine File must have a Part Section, Inputs Section, Outputs Section, and End Statement. All other sections are optional.
- The state machine clock that appears in the required Clock subsection of the Machine section is the actual node name that feeds the state machine register. (A state machine register is a group of flipflops whose outputs are the state variable values.) In a synchronous clock, this node is typically the left-hand side of an **INP** statement. Asynchronous clocks may be driven by a pin or by logic.
- If your state machine design requires an asynchronous clock that is driven directly from a pin, you must use a **CLKB** Primitive in the Network Section. (Refer also to *Altera Primitive Library* in the ***A+PLUS Reference Guide***.)
- An asynchronous clear input resets all state variable values to zero. The name of the clear signal is typically the left-hand side of an equation that contains one product term.
- Assigning pin numbers to both inputs and outputs may speed the fitting process.
- Every output can be an input to another state machine. (Multiple state machines can be described in a single SMF.)
- State names, state variable names, and state machine names are global and must be unique.
- State variables never require an I/O primitive statement in the Network Section; it is generated automatically.
- State variables may appear in the Outputs Section of the Declarations Section if you wish to use state variables as design outputs. (Pin outputs and feedbacks may have identical names.)

- State variables are interpreted as feedbacks from a macrocell; therefore, they may not appear in the Outputs subsections that follow Transitions subsections.

- Signal names in Outputs subsections (which follow Transitions subsections) are treated in the same manner as signals on the left-hand side of Boolean equations. Each signal name should be used at least once as an input to an I/O primitive or on the right-hand side of a Boolean equation. Signal names are not interpreted as feedbacks or pin outputs; therefore, they may not appear on the left-hand side of I/O primitives and they cannot be state variable names. Figure SM-13 illustrates the use of the States, Transitions, and Outputs subsections in a sample SMF. Figure SM-14 shows the corresponding ADF.

- Altera EPLDs always power up with all state variables at logic zero, i.e., [0, 0, 0 ... 0]. If your design depends on a specific starting state, you must know the power-up state of the device register. If the power-up state is not defined, the next state is undefined. You must specify the power-up state or make sure that your design goes to a known state after power-up.

- Auxiliary variables defined through Boolean equations or feedback from macrocells can appear within expressions describing conditional output and/or transitions from a state machine (e.g., in the Transitions and Outputs subsection of the Machine Section or inputs to a truth table).

- Transitions are evaluated in the order in which they are entered, i.e., the first transition has precedence over the second, the second over the third, etc. If no transitions are specified, the next state is the current state.

- A+PLUS offers two special features: (1) Turbo-Bit and (2) Security Bit.

  (1) The Turbo-Bit is a control bit for choosing speed and power characteristics of an EPLD. It can be set to **ON** or **OFF** in the Options Section of the SMF. If the Turbo-Bit status is not specified, it will default to **ON**. Before programming an EPLD, LogicMap also allows you to toggle the Turbo-Bit **ON** or **OFF**. (Note: the BUSTER and EP310 parts do not support the Turbo-Bit option. LogicMap will ignore Turbo-Bit information entered for these EPLDs.)

(2) The Security Bit prevents a device from being interrogated or inadvertently reprogrammed. It can be set to **ON** or **OFF** in the Options Section of the SMF. If the status of the Security Bit is not specified, it will default to **OFF**. Before programming an EPLD, LogicMap prompts you to indicate whether you wish to turn the Security Bit feature **ON** or **OFF**.

Table SM-2. Legal Characters for State Machine Names

Legal State Machine Name Characters		
A	X	u
B	Y	w
C	Z	x
D	a	y
E	b	z
F	c	0
G	d	1
H	e	2
I	f	3
J	g	4
K	h	5
L	i	6
M	j	7
N	k	8
O	l	9
P	m	_
Q	n	
R	o	
S	p	
T	q	
U	r	
V	s	
W	t	

```
PART:   EP310
INPUTS:   INp1, INp2, CLK
OUTPUTS:   OUTp1, OUTp2, OUTp3
NETWORK:
    OUTp1  =  CONF(O161,)
    OUTp2, FB1  =  RORF(O261,CLK,,,)
    OUTp3, FB2  =  RORF(AO3,CLK,,,)
EQUATIONS:
    AO3  =  FB1 * FB2 + O161 * O261;
MACHINE: 61
    CLOCK:   CLK
    STATES:  [ q0  q1  ]
      S0     [ 0   0   ]
      S1     [ 0   1   ]
      S2     [ 1   1   ]
      S3     [ 1   0   ]

S0:
    IF INp1 THEN S1
    S3
    OUTPUTS:
    IF /INp1 THEN O161
S1:
    S2
    OUTPUTS:
    O261
S2:
    S3
    OUTPUTS:
    IF /INp2 THEN O161
    O261
S3:
    S0
END$
```

Figure SM-13. Sample States, Transitions, and Outputs Subsections

```
<SMV Version information>
PART: EP310

INPUTS:
INp1, INp2, CLK

OUTPUTS:
OUTp1, OUTp2, OUTp3

NETWORK:
INp1  = INP(INp1)
INp2  = INP(INp2)
CLK   = INP(CLK)
OUTp1 = CONF(O161,)
OUTp2, FB1 = RORF(O261,CLK,,,)
OUTp3, FB2 = RORF(AO3,CLK,,,)
%
I/O's for State Machine "61"
%
q0 = NORF(q0.d, CLK, GND, GND)
q1 = NORF(q1.d, CLK, GND, GND)

EQUATIONS:

AO3 = FB1 * FB2 + O161 * O261;

% Boolean Equations for State Machine "61" %
% Current State Equations for "61" %
S0 = q0' * q1';
S1 = q0' * q1;
S2 = q0 * q1;
S3 = q0 * q1';

% SV Defining Equations for State Machine "61" %
q0.d' = S1.n
      + S0.n;
q1.d  = S1.n
      + S2.n;

% Next State Equations for State Machine "61" %
S1.n = (S0 * INp1);
S2.n = (S1);
S0.n = (S3);

% Output Equations for State Machine "61" %
O161  = (S2 * INp2')
      + (S0 * INp1');
O261  = (S2)
      + (S1);

END$
```

Figure SM-14. ADF Output for Figure SM-13

State Machine File (SMF) Format

A State Machine File is divided into several required and optional sections. The format and requirements of each SMF section are described below. Examples of each section are given in *Sample Sessions* and *Design Guidelines* (above). Figure SM-15 provides a full Backus-Naur Form (BNF) description of the SMF syntax. (Refer to *Appendix C* in the ***A+PLUS Reference Guide*** for information on the Backus-Naur Form.) Note that the SMF syntax is similar to the syntax of the Altera Design File (ADF).

Keywords

The keywords **OPTIONS:**, **PART:**, **INPUTS:**, **OUTPUTS:**, **NETWORK:**, **EQUATIONS:**, **MACHINE:**, and **T_TAB:** must always be the first word in the line in which they appear. (Do not enter comments, tabs, or blank spaces before these keywords.)

The keywords which appear within the Machine Section, namely, **CLOCK:**, **CLEAR:**, **STATES:**, and **OUTPUTS:**, may be preceded by blank spaces or tabs.

White Space and Comments

White space is defined as blank spaces, tabs, carriage returns, and line feeds. White space may be inserted between syntax elements to enhance readability, *except* in the following cases:

- Within any name
- Between **<prefix>** and **<name>** or **<name>** and **<postfix>** in the Equations Section
- Between **<prefix>** and **<expression>** or **<expression>** and **<postfix>** in the Equations Section
- Between **<prefix>** and **<state var name>** or **<state var name>** and **<postfix>** in (keyword format) state definitions in the States subsection
- Within a state label (i.e., between **<name>** and **':'**) in the Transitions and Outputs subsections
- Before keywords, as described in the preceding section

Comments must be enclosed by percent symbols (**%**). They may contain any printable character except the delimiter **%**. They may be inserted wherever white space is allowed, *except* within the Header Section of a file.

Header Section

The optional Header Section of the State Machine File provides design documentation. This section has several important uses and restrictions:

- If it is present, it must be the first section in the file.
- It is included in the header of the Altera Design File, the Utilization Report, and the JEDEC File.
- Any printable character except the asterisk (*) is allowed.
- It may include EOL (end-of-line) characters.
- It is terminated by either the **OPTIONS:** or **PART:** keywords, which A+PLUS recognizes as the start of the Declarations Section.

Fields in the Header Section appear in the following order, with one item per line. The maximum character count for each field is indicated in parentheses.

1. Designer (48)
2. Company (60)
3. Date (24)
4. Number (24)
5. Revision (24)
6. EPLD (10)
7. Comment (512)
8. Any other information (may be more than one line)

Declarations Section

The Declarations Section specifies the EPLD used for the design, input and output signal destinations, and optional pin assignments. It also allows you to disable/enable the Turbo-Bit (a control bit for choosing speed and power characteristics of an EPLD) and the Security Bit (which prevents a device from being interrogated or inadvertently reprogrammed.)

The Declarations Section contains subsections that are referred to as the Options, Part, Inputs, and Outputs sections. Each section is described below.

Options Section

The Options Section is optional. It is identified with the keyword **OPTIONS:**, followed by either or both of the following option specifiers:

- **TURBO**, indicating Turbo-Bit. The values are **ON** (the default) or **OFF**. If **ON** or **OFF** is not specified, the Turbo-Bit defaults to **ON**. (The BUSTER (EPB1400) and EP310 parts do not support the Turbo-Bit option. LogicMap will ignore any Turbo-Bit information entered for these EPLDs.)

- **SECURITY**, indicating the Security Bit. The values are **ON** or **OFF** (the default). If **ON** or **OFF** is not specified, the Security Bit defaults to **OFF**.

Part Section

The Part Section is required. It is the first section of the SMF if no Header or Options Section is present. It consists of the keyword **PART:**, followed by a target EPLD part number or the term **AUTO** to specify automatic part selection. It is terminated by the **INPUTS:** keyword.

If automatic part selection is specified, the State Machine Converter assumes that a part with T flipflops is available and the ADP automatically chooses the part most likely to fit your design. For parts that don't have T flipflops (EP310, EP320, EP1210), you should specify the part name.

Automatic part selection will not be successful if you specify pin assignments, or if there are too many inputs, outputs, or macrocells in the design.

Inputs Section

The Inputs Section is required. It consists of the keyword **INPUTS:**, followed by a list of input signals, including optional pin assignments. Input signal names may contain up to eight characters, including any printable character except percent symbol (%), comma (,), equal symbol (=), at-symbol (@), or left and right parentheses (()). You may include specific pin assignments by appending an at-symbol (@) plus a one- or two-digit pin number to any input name on the list. (In the EP1800G, pin numbers are specified with an @-symbol plus one letter and a one- or two-digit number.) This section must follow the Part Section. It is terminated by the **OUTPUTS:** keyword. Example:

1. If you use an asterisk (*) in an input or output pin name, it will be converted into a tilde (~) in the JEDEC file generated by the ADP. You must ensure that this conversion will not create duplicate pin names (for example, if you use both * and ~ when naming pins).

2. The ADP creates internal node names that contain periods (.). User-assigned pin names that contain periods may occasionally conflict with these node names and cause unpredictable results.

3. The Functional Simulator ignores all user-defined pin names that contain periods (periods are allowed only for referencing internal nodes of I/O primitives).

Outputs Section

The Outputs Section is required. It consists of the keyword **OUTPUTS:**, followed by a list of output signals, including optional pin assignments. Output pin names and numbers have the same format as in the Inputs Section. The Outputs Section must follow the Inputs Section. It is terminated by the Network, Equations, or Machine sections.

Network Section

The Network Section is optional. It consists of the keyword **NETWORK:**, followed by one or more primitive statements. The Network Section is terminated by either the Equations or Machine section.

Refer to *Network Section Requirements* in *Boolean Equation Entry* (***A+PLUS User Guide***) for detailed guidelines on using primitive statements. See also *Altera Primitive Library* in the ***A+PLUS Reference Guide*** for a description of each primitive.

Equations Section

The Equations Section is an optional section that allows you to include Boolean equations in your state machine design. It consists of the keyword **EQUATIONS:**, followed by zero or more Boolean equations. (For detailed information on using Boolean equations, see *Boolean Equation Entry* in the ***A+PLUS User Guide***.) The Equations Section is terminated by the **MACHINE:** keyword. Note that entire equations may be substituted into the right-hand sides of other equations by substituting intermediate variables.

Machine Section

The Machine Section specifies the state machine, the clock that synchronizes the state machine, and the state variables. It also allows you to specify optional state transition and output definitions, and an asynchronous clear input to the state machine. Multiple state machines may be described in a single SMF.

The Machine Section consists of the keyword **MACHINE:**, followed by the name of the state machine. The name of the state machine is followed by optional and required subsections which are referred to as the Clock, Clear, States, Transitions, and Outputs subsections. (Note the distinction between the Outputs *subsection* of the Machine

Section and the Outputs *Section* of the Declarations Section.) The name of the state machine may contain up to 32 alphanumeric characters or underscores. Each subsection is described below.

Clock Subsection

The required Clock subsection specifies the clock that synchronizes the state machine. It consists of the keyword **CLOCK:**, followed by the a clock name containing up to eight alphanumeric characters. The clock name in a state machine is the name of the node feeding the state machine register. In a synchronous clock, this node is typically the left-hand side of an **INP** statement. The Clock subsection is terminated by the **CLEAR:** or **STATES:** keyword.

If a state machine design requires an asynchronous clock that is driven directly by a pin, use a **CLKB** Primitive in the Network Section. (Refer to *Appendix A* in the ***A+PLUS Reference Guide***.)

Clear Subsection

The optional Clear subsection allows you to specify an asynchronous clear input to the state machine. (The clear input resets all state variable values to zero.) It consists of the keyword **CLEAR:**, followed by a clear signal name containing up to eight alphanumeric characters. The clear signal in a state machine is the name of the node feeding the asynchronous clear input of the state machine register.

To implement an Active Low clear, use a **NOT** statement to invert the clear signal in the Network Section. For example:

```
INPUTS:    nSIGX, SIGY, SIGZ
     .          % nSIGX is an active low signal %
     .
     .
NETWORK:
nSIGX   =  INP(nSIGX)
SIGX    =  NOT(nSIGX)  % invert the node %
     .
     .
     .
CLEAR:     SIGX  % implements Active Low clear %
```

States Subsection

The States subsection is required. It consists of the keyword **STATES:**, followed by a list of state variable names in square brackets ([ ]). After the state variable names are listed, the value of each state variable must be assigned for each state. Each state name and state variable name may contain up to eight alphanumeric characters. State variable assignments may be given in one of two formats: positional or keyword format.

- Any one state machine can have a maximum of 16 state variables.
- The combination of state variable values that define a state must be unique within a given state machine.
- Each state variable corresponds to the feedback output of a flipflop. Therefore, the number of state variables defined is equal to the minimum number of flipflops (and hence macrocells) required to implement the current state machine.

Positional state variable format

This format consists of a state name, followed by a list of state variable values enclosed in square brackets. The values are **0**, **1**, or **X** (don't care). The values for each state are assumed to be in the same order as those in the list of state variables following the **STATES:** keyword.

Since the combination of state variable values must be unique for each state, you should exercise caution when assigning **X** (don't care) as a state variable value.

Keyword state variable format

This format consists of a state name, followed by a list of state variable names (complemented or uncomplemented) enclosed in square brackets. The list corresponds to a product term that is true when the current state machine is in the state presently being defined. Don't Care variables are left out. This format may span lines.

To prevent confusion between these two formats, state variables may not be named **0**, **1**, or **X**.

The States subsection is terminated with one of the following: the Transitions and Outputs subsections, the next Machine Section, a Truth Table Section, or the **END$** statement.

Transitions and Outputs Subsections

The Transitions and Outputs subsections are optional. They contain state transition and output definitions that correspond to each state.

- All states for the current state machine must have their transitions and outputs described in these subsections.
- The Transitions subsection for a state begins with the source **<state name>** followed immediately by a colon (**:**). It does not have an identifying keyword. The Transitions subsection always precedes the optional Outputs subsection for the state.
- Each transition is followed by the corresponding output(s) (i.e., destination(s)) of the state, which is identified with the **OUTPUTS:** keyword.
- State outputs are optional. If a state has no outputs, the **OUTPUTS:** keyword must not be present.
- The Transitions and Outputs subsections terminate with the next Machine Section, Truth Table Section, or the **END$** statement.

Transition Syntax

Transition definitions may be given in three forms: conditional transitions, unconditional transitions, and case statements.

(1) Conditional transitions:

Conditional transitions are entered in the following format:

```
state0:
IF condition THEN next state1
  % ELSE is implied %
IF condition THEN next state2
  .
  .
  .
IF condition THEN next stateN
  % for N clauses %
```

The requirements for conditional transitions are as follows:

- Only the first **IF-THEN** clause is required.
- Each condition must be a Boolean expression consisting of input variables, intermediate variables, and/or names of states from other state machines on which the current transition depends.
- The **<state name>** following **THEN** represents the next state of the current machine if the associated condition is true.
- A group of **IF-THEN** clauses is terminated by an unconditional transition, the start of the next set of state transition definitions, the **OUTPUTS:** keyword, the next Machine or Truth Table section, or the **END$** statement.

(2) Unconditional transitions:

The destination state of an unconditional transition is entered after the state name and colon that identify the source state or after the *last* conditional or case statement transition. For example:

```
state0:
  state1
```

(3) Case statement transitions:

Case statement transitions are entered in the following format:

```
state0:
CASE
condition1  :  state1
condition2  :  state2
condition3  :  state3
  .
  .
  .
conditionN  :  stateN
ENDCASE
```

The requirements for case statement transitions are identical to those for conditional (**IF-THEN**) transitions (described above).

Output Syntax

Signal names in the Outputs subsection are treated in the same manner as signals on the left-hand side of Boolean equations. These names may be used as inputs to I/O primitives or may appear on the right-hand side of Boolean equations. They are not interpreted as feedbacks or pin outputs; therefore, they may not appear on the left-hand side of I/O primitives. They may not be state variable names.

Output definitions may be conditional or unconditional (case statement output definitions are not allowed):

(1) Conditional outputs have the same syntax as conditional transitions, except for the state name, which is replaced by the output signal name. Outputs are not mutually exclusive, i.e., **ELSE** is not implied. All conditional outputs are evaluated whenever the state with which they are associated occurs.

(2) An unconditional output is the name of the signal that is asserted in the current state. If a signal is not asserted, it is set to zero for that state.

Truth Table Section

The Truth Table Section is optional. Multiple truth tables may be described in a single SMF. It is identified by the **T_TAB:** keyword.

- Each truth table requires its own **T_TAB:** keyword.
- Each Truth Table Section terminates with the next **T_TAB:** or **MACHINE:** keyword, or with the **END$** statement.
- A truth table output is interpreted as a Boolean variable, i.e., wherever the left-hand side of a Boolean equation is found, a truth table output may be found.
- All names are global to allow truth tables to be automatically linked.
- Legal values for a truth table are:

  1 => TRUE
  0 => FALSE
  X => DON'T CARE

  The default value for table entries is **FALSE (0)**.

Truth tables are entered in the following format:

```
T_TAB: input declarations    :  output declarations;
                row1 inputs  :  row1 outputs;
                row2 inputs  :  row2 outputs ;
                    .
                    .
                    .
                rowN inputs  :  rowN outputs;
                    % for an N-row truth table %
```

The input and output declarations are variable names defined elsewhere in the SMF. These declarations contain the names of the outputs of and inputs to other Boolean or non-Boolean resources. (A truth table is considered to be a Boolean resource.)

End Statement

The End Statement terminates the SMF. It consists of the string:

END$

```
<smf> ::= [<header>] <declarations> [<network>] [<equations>] {<machine> |
        <truth table>} 'END$'

<header> ::= <header char>:0:48 <EOL>
        <header char>:0:60 <EOL>
        <header char>:0:24 <EOL>
        <header char>:0:24 <EOL>
        <header char>:0:24 <EOL>
        <header char>:0:10 <EOL>
        <header char>:0:512 <EOL>
        {{<header char>} <EOL> {{<header char>} <EOL>}
<header char> ::= <tab> | <space> | <hex 21> | ... | <hex 29> | <hex 2B> |
        <hex 2C> | ... | <hex 7E>
            (i.e., any printable ASCII character except an asterisk (*))
<tab> ::= <hex 09>
<EOL> ::= <hex 0D> <hex 0A>
<space> ::= <hex 20>

<declarations> ::= [<options>] <part> <inputs> <outputs>
<options> ::= 'OPTIONS:' <opt spec> {[ ',' ] <opt spec>} <EOL>
<part> ::= 'PART:' <part name> <EOL>
<inputs> ::= 'INPUTS:' <i/o list> <EOL>
<outputs> ::= 'OUTPUTS:' <i/o list> <EOL>
<opt spec> ::= <opt name> [ '=' <opt value>]
<opt name> ::= 'TURBO' | 'SECURITY'
<opt value> ::= 'ON' | 'OFF'
<part name> ::= 'EP310' | 'EP310D' | 'EP320' | 'EP320D' | 'EP600' | 'EP600D'
        | 'EP600J' | 'EP610' | 'EP610D' | 'EP610J' | 'EP900' |'EP900D' |
        'EP900J' | 'EP910' | 'EP910D' | 'EP910J' | 'EP1210' | 'EP1210D' |
        'EP1210J' | 'EPB1400' | 'EPB1400D' | 'EPB1400J' | 'EP1800' |
        'EP1800J' | 'EP1800G' | 'AUTO'
<i/o list> ::= <i/o name> {',' <i/o name>}
<i/o name> ::= <pin name> [ '@' <pin number>]
<pin name> ::= <pin char>:1:8
<pin char> ::= <any printable ASCII character except comma, %, @, =, (, )>
<pin number> ::= [<letter prefix>] <digit> [<digit>]
<letter prefix> ::= 'A' | 'B' | ... | 'H' | 'J' | 'K' | 'L'
            (letter prefix allowed only in EP1800G pin numbers)
<digit> ::= '0' | '1' | ... | '9'
```

Figure SM-15. BNF Syntax for State Machine File (Part 1 of 3)

```
<network> ::= 'NETWORK:' <network list>
<network list> ::= <primitive equation> {<primitive equation>}
<primitive equation> ::= <parameters> '=' <primitive> '(' <parameters> ')'
<parameters> ::= <pin name> | <name> {',' <name>}
<name> ::= <name char>:1:8
<name char> ::= <digit> | <alphabet letter>
<alphabet letter> ::= 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
<primitive> ::= <any Altera Primitive Library primitive name>
        (e.g., CONF, INP, NORF)

<equations> ::= 'EQUATIONS:' {<logic equation>}
<logic equation> ::= <LHS> '=' <expression> ';' <EOL>
<LHS> ::= <prefix><name> | <name> | <name><postfix>
        (no white space is allowed between <prefix><name> and
        <name><postfix>)
<expression> ::= <name> | <prefix><expression> | <expression><postfix> |
      <expression> <infix> <expression> | '(' <expression> ')'
        (no white space is allowed between <prefix><expression> and
        <expression><postfix>)
<prefix> ::= '/' | '!'
<infix> ::= '+' | '*' | '#' | '&'
<postfix> ::= '''

<machine> ::= <state machine name> <clock> [<clear>] <states>
      [<transitions and outputs>]
<state machine name> ::= 'MACHINE:' <name char> {<name char> |
      '_'}:1:32
<clock> ::= 'CLOCK:' <name>
<clear> ::= 'CLEAR:' <name>
<states> ::= 'STATES:' '[' <state var name> {<state var name>}:1:15 ']' <state
      definition> <EOL> {<state definition> <EOL>}
<state var name> ::= (<name> except <state var value>)
        (i.e., state variables may not be named 0, 1, or X)
<state definition> ::= <positional definition> | <keyword definition>
<positional definition> ::= <name> '[' <state var value> {<state var
      value>}:1:15 ']'
<keyword definition> ::= <name> '[' <state var spec> {<state var spec>} ']'
```

Figure SM-15. BNF Syntax for State Machine File (Part 2 of 3)

```
<state var spec> ::= <prefix><state var name> | <state var name> | <state var
      name><postfix>
            (no white space is allowed between <prefix><state var name> and
            <state var name><postfix>)
<state var value> ::= '0' | '1' | 'X'
<transitions and outputs> ::= {<state label> <state trans> [<state output>] }
<state label> ::= <name>':'
            (no white space is allowed between <name> and <:>)
<state trans> ::= <conditional trans> | <case stmt trans> | <uncond trans>
<conditional trans> ::= 'IF' <expression> 'THEN' <name>
            (ELSE is implied)
<case stmt trans> ::= 'CASE' <expression> ':' <name> {<expression> ':'
      <name>} 'ENDCASE'
<uncond trans> ::= <name>
<state output> ::= 'OUTPUTS:' (<conditional output> | <uncond output>
      {<conditional output> | <uncond output>})
<conditional output> ::= 'IF' <expression> 'THEN' <name>
<uncond output> ::= <name>

<truth table> ::= 'T_TAB:' <input list> ':' <output list> ';' <EOL>
      <row of inputs> ':' <row of outputs> ';' <EOL>
      {<row of inputs> ':' <row of outputs> ';' <EOL>}
<input list> ::= <name> {[ ',' ] <name>}
<output list> ::= <name> {[ ',' ] <name>}
<row of inputs> ::= <state var value> {[ ',' ] <state var value>}
<row of outputs> ::= <state var value> {[ ',' ] <state var value>}

'END$'
```

Figure SM-15. BNF Syntax for State Machine File (Part 3 of 3)

State Machine Converter Messages

During SMF-to-ADF conversion, the State Machine Converter (SMV) displays Error, Information, and Warning messages on screen. These messages are prefixed with ***ERR-SMV-, ***INFO-SMV-, and ***WARN-SMV-, respectively. (Messages with other prefixes are listed in *A+PLUS Messages* in the ***A+PLUS Reference Guide***.)

Error messages indicate errors that must be corrected before file conversion can continue. Warning messages indicated potential problems; information messages simply report SMV processing status.

Many messages are preceded by a line of the following format, which indicates the source of the message:

Line <SMF line #>: <text of SMF line>

Error Messages

ERR-SMV- Can't open input SMF

CAUSE: The State Machine Converter was unable to open your input design file. It is possible that the file/pathname was spelled incorrectly, the filename does not have the required extension **.SMF**, the file is corrupted, or the disk is full or corrupted. Also, there may not be enough memory available to open the file.

ACTION: Ensure that you have spelled the file and path names correctly (including the extension **.SMF**), that you have 128K of disk space and enough memory to run A+PLUS, and that the disk and SMF are not corrupted.

ERR-SMV- Can't open output ADF

CAUSE: The State Machine Converter was unable to open the ADF output file. It is possible that the disk is full or corrupted or that available memory is insufficient to open the file.

ACTION: Ensure that you have 128K of disk space, enough memory to run A+PLUS, and that your disk is not corrupted. If you are combining more than 3 SMFs, be sure that **FILES=20** and **BUFFERS=12** have been included in your **CONFIG.SYS** file.

ERR-SMV- Can't use Case stmt in Outputs subsection: <machine name>

CAUSE: You have used a Case statement in the Outputs subsection of the specified state machine design.

ACTION: Use only conditional (**IF-THEN**) and unconditional output statements in the Outputs subsection (Case statements may be used only in the Transitions subsection).

ERR-SMV- Can't use state name in expression: <machine name>

CAUSE: You have used a state name in an expression (i.e., the condition in a conditional or Case statement transition) in the Transitions and/or Outputs subsection of your design.

ACTION: For each expression, use a Boolean expression consisting of input variables, intermediate variables, and/or names of states from other state machines on which the current transition depends.

ERR-SMV- Can't use state output names in Inputs Section: <machine name>

CAUSE: You have used the name of a state output in the Inputs Section of your design file.

ACTION: Ensure that state outputs do not appear in the design's Inputs Section.

ERR-SMV- Can't use state output names in Outputs Section: <machine name>

CAUSE: You have used the name of a state output in the Outputs Section of your design file.

ACTION: Ensure that state outputs do not appear in the design's Outputs Section.

ERR-SMV- Can't use state variable names in Outputs Subsection: <machine name>

CAUSE: You have used a state variable name in the Outputs subsection as well as in a state definition in the States subsection.

ACTION: Ensure that state variables are unique within a single SMF. (They may not appear in the design's Outputs subsection.)

ERR-SMV- Duplicate state name: <name>

CAUSE: Your input SMF contains the state **<name>** in two or more places.

ACTION: Ensure that all state names are unique within a single SMF.

ERR-SMV- Duplicate state variable assignment: (<str>)

CAUSE: More than one state had the same state variable assignment within a single state machine.

ACTION: Ensure that all state variable assignments are unique within a single state machine.

ERR-SMV- Duplicate state variable name: <name>

CAUSE: You have used the same state variable **<name>** in two or more state machines.

ACTION: Ensure that all state variable names are unique within a single SMF.

ERR-SMV- Equation too long

CAUSE: Your input SMF contains an equation with more than 256 characters on a single line.

ACTION: Split the equation over two lines.

ERR-SMV- I/O error -- temporary file

CAUSE: The disk is full or corrupted, or there has been a disk I/O error, so the State Machine Converter can't use a temporary file needed for design processing. Also, there may not be enough memory available to open the file.

ACTION: Ensure that you have 128K of disk space and enough memory to run A+PLUS, that your DOS path includes the current directory, that the diskette is not write- or read-protected, corrupted, or not in the disk drive.

ERR-SMV- Illegal character in <name> section

CAUSE: An illegal or misplaced character was detected in the section indicated.

ACTION: Refer to Figure SM-16 in *State Machine Entry*, "BNF Syntax for State Machine File," for a precise description of the characters which may be used in each section of the SMF.

ERR-SMV- Illegal Outputs subsection in <machine name>

CAUSE: The Outputs subsection contains an output statement whose syntax does not conform to either of the two permitted formats or the output signal name is not a node name.

ACTION: Ensure that output signal names are not state names or state variable names. Use conditional (**IF-THEN**) or unconditional output statement format in the Outputs subsection (Case statements are not allowed).

ERR-SMV- Illegal pin assignment: (#,<pin name>)

CAUSE: The pin assignment string (#) for **<pin name>** contains an illegal character or an unexpected End-of-File (EOF).

ACTION: Specify pin assignments by appending an at-symbol (@) plus a one- or two-digit number to the signal name. (In the EP1800G, pin numbers are specified with an @-symbol plus one letter and a one- or two-digit number.)

ERR-SMV- Illegal state machine clear name: (<code>,<node name>)

CAUSE: The **<node name>** of the state machine Clear is too long or contains an illegal character. (**<code>** is an internal diagnostic code.)

ACTION: Ensure that all node names contain eight or fewer alphanumeric characters (all other characters are illegal).

ERR-SMV- Illegal state machine clock name: (<code>,<node name>)

CAUSE: The **<node name>** of the state machine Clock is too long or contains an illegal character. (**<code>** is an internal diagnostic code.)

ACTION: Ensure that all node names contain eight or fewer alphanumeric characters (all other characters are illegal).

ERR-SMV- Illegal state machine name

CAUSE: The name of your state machine contains over 32 characters or characters not permitted in state machine names.

ACTION: Use a state machine name that contains 32 or fewer alphanumeric characters and underscores (all other characters are illegal).

ERR-SMV- Illegal Transition statement in <machine name>

CAUSE: The Transitions subsection contains a transition statement whose syntax does not conform to any of the three permitted formats or the destination of a transition is not a state or node name.

ACTION: Ensure that the destinations of transitions are not state variable names. Use conditional (**IF-THEN**), unconditional, or Case statement transitions in the format described in *State Machine Entry* in the ***A+PLUS User Guide***.

ERR-SMV- Internal error: <str>

CAUSE: There is an internal error.

ACTION: Should you encounter this message, please contact Altera Applications and describe the error text/number.

ERR-SMV- Mismatched parentheses in <machine name>

CAUSE: A left or right parenthesis is missing from an expression in the Transitions or Outputs subsection of your design.

ACTION: Ensure that an equal number of left and right parentheses are present in each expression.

ERR-SMV- Missing <str> Section

CAUSE: An EOF was detected before the specified section (i.e., Inputs, Outputs, or Part) was found. These sections are required.

ACTION: Ensure that each SMF has the required sections listed above. Remove blank spaces, tabs, and non-printing characters that appear before the keywords for these sections. Be sure to specify an EPLD name or **AUTO** in the Part Section.

ERR-SMV- Missing destination state in <machine name>

CAUSE: A transition in the Transitions or Outputs subsection of your design does not have a destination state (i.e., there is nothing after the **THEN** in a conditional transition or after the colon (**:**) in a Case statement or unconditional transition).

ACTION: Add the name of a destination state to the transition statement.

ERR-SMV- Missing expression in <machine name>

CAUSE: A conditional (**IF-THEN**) or Case statement transition in your design is missing an expression (i.e., a Boolean expression).

ACTION: Ensure that conditional statements have the format "**IF <expression> THEN <name>**" and Case statements have the format "**<expression> : <name>**".

ERR-SMV- Missing node name in <machine name>

CAUSE: An expression in Transitions and/or Outputs subsection of your design is missing a node name.

ACTION: Check the expressions for syntax errors. For example, you may have left a space between a node name and its complement, or typed two operators in succession.

ERR-SMV- Missing operator in <machine name>

CAUSE: An expression in the Machine Section of your design is missing a binary operator (i.e., +, #, *, or &).

ACTION: Insert one of the binary operators given above. (Note: permitted NOT-operators are !, /, and '.)

ERR-SMV- Missing or illegal Clock subsection: (<code>,<str>)

CAUSE: The SMV found <str> instead of the required **CLOCK:** keyword on the first new line of text following the state machine name. (<code> is an internal diagnostic code.)

ACTION: Insert the Clock subsection as the first new line of text after the state machine name. Ensure that there are no characters other than spaces or tabs before the **CLOCK:** keyword.

ERR-SMV- Missing or illegal States subsection

CAUSE: The SMV did not find the required **STATES:** keyword on the first new line of text following the Clear (or if Clear is not used, Clock) subsection.

ACTION: Insert the States subsection. Ensure that there are no characters other than spaces or tabs before the **STATES:** keyword.

ERR-SMV- Missing state variable assignment

CAUSE: You have not provided state variable values for one or more states in the States subsection.

ACTION: Assign all state variable values in the States subsection.

ERR-SMV- Missing state variable name

CAUSE: You have not provided the name of one or more state variables in the States subsection.

ACTION: Declare all state variable names.

ERR-SMV- Network Section line too long

CAUSE: A line in the Network Section contains more than 256 characters, which is the maximum permitted on a single line.

ACTION: Split the line into two or more lines.

ERR-SMV- Out of memory

CAUSE: Available memory is insufficient to complete the current processing step. Other resident programs may be occupying memory, such as RAM-disks, print spoolers, communication packages, and keyboard enhancers. This message may also occur if you have used the **DOS Command** (**<F8>**) function and then *re*-invoked A+PLUS from the temporary DOS environment (a duplicate copy is read into memory if you reinvoke A+PLUS).

ACTION: Use the DOS command **CHKDSK** to check available memory and disk space. Temporarily relocate other programs to make enough memory available to run A+PLUS and do not use background processes (e.g., Sidekick) when running A+PLUS. If you have used **<F8>**, quit A+PLUS and type **EXIT** to return to the "original" A+PLUS. If you are processing a very large input file, you may wish to run the ADP directly from DOS, which will increase the available memory by approximately 70K (for details, see *A+PLUS and ADP Reference*).

ERR-SMV- State variable definition must begin with '[': (<code>,<str>)

CAUSE: The state variable definition field following the **STATES:** keyword begins with **<str>** instead of the required left square bracket ([). (**<code>** is an internal diagnostic code.)

ACTION: Use square brackets ([ ]) to enclose the state variable names and state variable values in the States subsection.

ERR-SMV- State variable definition must end with ']'

CAUSE: The state variable definition field (in the States subsection) does not terminate with a right square bracket (]).

ACTION: Ensure that the state variable names and each group of state variable values are enclosed in square brackets ([ ]).

ERR-SMV- States subsection terminated incorrectly

CAUSE: Your input SMF contains a state machine whose States subsection is not terminated by a ']' or not followed by a Transitions subsection, Machine Section, Truth Table Section, or **END$** statement.

ACTION: Ensure that the States subsection ends with one of the items listed above.

ERR-SMV- Syntax error in <machine name>

CAUSE: A Boolean expression in your input SMF contains a syntax error. For example, this message occurs if the SMV finds a name (possibly complemented) that is not followed by a binary operator or a parenthesis.

ACTION: Check the SMF for syntax errors.

ERR-SMV- Too many state variables

CAUSE: Your input file contains a state machine with more than 16 state variables, which is the maximum allowed.

ACTION: Reduce the number of state variables in each state machine to 16 or less. (You may use multiple state machines in a single SMF.)

ERR-SMV- Too many/few state variable values

CAUSE: The number of state variable values given in a state definition is not equal to the number of state variable names listed at the beginning of the States subsection.

ACTION: Ensure that there are as many state variable values as state variable names.

ERR-SMV- Truth table: missing <str> declarations

CAUSE: The **T_TAB:** keyword is not followed by any **<str>** (i.e., input or output) declarations, which are required.

ACTION: Add truth table inputs/outputs to your design file. Ensure that input and output declarations are separated by a colon (:).

ERR-SMV- Truth table: too many/few <str> arguments

CAUSE: The number of arguments specified for **<str>** (i.e., inputs or outputs) does not match those given in the input/output declarations of the truth table (i.e., on the first line).

ACTION: Ensure that there are as many input and output arguments as input and output declarations in each truth table.

ERR-SMV- Undeclared input: <str>

CAUSE: The **<str>** given in an **INP** primitive in the Network Section of your input file was not declared in the Inputs Section.

ACTION: Add the specified **<str>** to the Inputs Section or remove the **INP** primitive statement.

ERR-SMV- Undefined state: (<state name>) in <machine name>

CAUSE: The indicated **<state name>** used in the Transitions subsection is not defined in the States subsection.

ACTION: Define all states in the States subsection.

ERR-SMV- Unexpected EOF in <section name>

CAUSE: The End-of-File (EOF) was reached prematurely. This message indicates that the input SMF is incomplete, is not terminated with the required **END$** statement, or that a statement in the Network Section has more than 256 characters on a single line.

ACTION: Ensure that the last line of the SMF is the **END$** statement and that lines in the Network Section contain 256 characters or less.

ERR-SMV- Unrecognized section

CAUSE: The input file contains an unrecognized keyword or symbol name.

ACTION: Check the SMF for syntax errors and remove all keywords other than **OPTIONS:**, **PART:**, **INPUTS:**, **OUTPUTS:**, **NETWORK:**, **EQUATIONS:**, **MACHINE:**, **CLOCK:**, **CLEAR:**, **STATES:**, **OUTPUTS:**, and **T_TAB:**.

ERR-SMV- Unrecognized symbol in state assignment

CAUSE: The state assignment (state variable value) in the States subsection contains a string which is not a **1**, **0**, or **X** (in positional format), or a complemented or uncomplemented state variable name (in keyword format).

ACTION: Check the States subsection for errors.Ensure that the state variable values are assigned in only one of the two formats given above.

Information Messages

INFO-SMV- Beginning Execution

CAUSE: The State Machine Converter (SMV) has begun converting your input design file into an Altera Design File (ADF).
ACTION: No action is required.

INFO-SMV- Conversion completed successfully

CAUSE: The SMV has successfully converted the SMF into an ADF.
ACTION: You may submit the resulting ADF to the ADP at any time.

INFO-SMV- Conversion terminated abnormally

CAUSE: The SMV was unable to complete the SMF-to-ADF conversion. This message always follows other error and/or warning messages.
ACTION: Correct the errors and resubmit the file to the SMV.

Warning Messages

WARN-SMV- Empty truth table: no statements

CAUSE: The truth table contains only one line (i.e., the **T_TAB:** keyword and input and output declarations are not followed by input and output arguments).
ACTION: Enter the input and output arguments (statements) for the truth table or remove the line containing the **T_TAB:** keyword.

WARN-SMV- No exit from state: (<state name>) in <machine name>

CAUSE: The **<state name>** has no transitions leading to other states (i.e., there is no way to leave **<state name>**). You may have misspelled a state name.
ACTION: Check the current entries for destination states in the Transitions subsection. Ensure that all transitions can lead to another state.

WARN-SMV- Symbol too long in <machine name>

CAUSE: There are too many characters in a state variable name, state name, or node name.

ACTION: Use a maximum of 8 alphanumeric characters in the names listed above.

WARN-SMV- Unreachable Case statement: (<state name>) in <machine name>

CAUSE: The SMV detected a Case statement that was preceded by an unconditional transition.

ACTION: Ensure that the order of transitions allows all transitions to be reached. (All transitions are evaluated in the order in which they are entered.) If the unconditional transition represents the implied **ELSE**, it must follow the Case statement transition.

WARN-SMV- Unreachable conditional statement: (<state name>) in <machine name>

CAUSE: The SMV detected a conditional transition (**IF** statement) that was preceded by an unconditional transition.

ACTION: Ensure that the order of transitions allows all transitions to be reached. (All transitions are evaluated in the order in which they are entered.) If the unconditional transition represents the implied **ELSE**, it must follow the conditional transition.

WARN-SMV- Unreachable unconditional statement: (<state name>) in <machine name>

CAUSE: The SMV detected two unconditional transition statements in succession.

ACTION: Check the input SMF for errors and ensure that each unconditional transition follows a conditional transition.