<table>
<thead>
<tr>
<th>Tools</th>
<th>Techniques</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>CPLD Design Techniques</td>
<td>What's New in Foundation Series 3.1i</td>
</tr>
<tr>
<td>Schematic Editor</td>
<td>FPGA Design Techniques</td>
<td>Licensing Synthesis Products</td>
</tr>
<tr>
<td>Foundation Express</td>
<td>Project Flows</td>
<td>VHDL Reference Guide</td>
</tr>
<tr>
<td>HDL Editor</td>
<td>Application Notes</td>
<td>Verilog Reference Guide</td>
</tr>
<tr>
<td>State Editor</td>
<td>Entering Constraints</td>
<td>ABEL Reference Guide</td>
</tr>
<tr>
<td>LogiBLOX</td>
<td></td>
<td>Foundation Configuration Information</td>
</tr>
<tr>
<td>Logic Simulator</td>
<td>Foundation Series Tutorial</td>
<td>CPLD Schematic Library</td>
</tr>
<tr>
<td>Flow Engine</td>
<td>State Editor Tutorial</td>
<td>CPLD Attributes</td>
</tr>
<tr>
<td>Timing Analyzer</td>
<td>CPLD Design Flows</td>
<td>Technical Support</td>
</tr>
<tr>
<td>JTAG Programmer</td>
<td>Hierarchy Tutorial</td>
<td></td>
</tr>
<tr>
<td>Advanced Tools</td>
<td>Other Tutorials</td>
<td></td>
</tr>
</tbody>
</table>
State Editor Tutorial - Contents

Introduction
Comparison of State Machine Description Methods
The Purpose of This Tutorial
Creating a State Diagram
Implementation
Creating BLKJACK Project
Opening State Editor
Using HDL Design Wizard
Adding Ports
Adding Extra Ports
Defining Additional Variables
Adding a Reset State
Adding Initialization Actions
Adding a State to Request a Card
HDL Code Generation
Receiving and Handling of the New Card
Analyzing the Total Score
Testing the Final Score
Assigning Condition Priorities
Selecting State Encoding
Analyzing the Design
Locating and Correcting Errors
Synthesizing the Design
Functional Simulation
Physical Implementation
Conclusion
Introduction

State machine editors allow simple and easy graphical design entry. Since the state designs can be easily retarget to any devices, the state editors are becoming very popular with designers who value technological independence. This tutorial is for Foundation Series users who want to learn designing with the state machine editor, that can be used for graphical entry of Verilog, VHDL and ABEL designs. The example in this tutorial refers to a VHDL design entry.
Comparison of State Machine Description Methods

Let’s consider a very simple state machine that is used for controlling traffic lights. Since the best way to start any state machine design is with verbal description, let’s describe the operation of traffic lights controller as follows:

When the red light is on and timer signals that green light should be lit (GO_GREEN signal goes to High state) turn on the GREEN signal and turn other signals off. When the green light is on and timer signals that the yellow light should be lit (GO_YELLOW signal goes to High state) turn on the YELLOW signal and turn other signals off. When the yellow light is on and timer signals that the red light should be lit (GO_RED signal goes to High state) turn on the RED signal and turn other signals off.

State diagram is a graphical method of state machine description. As an example, a State diagram for traffic lights controller is shown below:

State machines can be converted into HDL programs. The HDL programs can then be converted into physical implementation (netlist) by logic synthesis software. The most popular HDL languages are VHDL, Verilog and ABEL. A VHDL description of a traffic lights controller is shown below:
The main advantages and disadvantages of state machine designs are listed in the table below.
<table>
<thead>
<tr>
<th>Verbal Description</th>
<th>State Diagram</th>
<th>HDL Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compact</td>
<td>very compact</td>
<td>lengthy</td>
</tr>
<tr>
<td>easy to read by humans</td>
<td>very easy to read by humans</td>
<td>hard to read by humans</td>
</tr>
<tr>
<td>difficult to implement</td>
<td>easy to implement</td>
<td>easy to implement</td>
</tr>
<tr>
<td></td>
<td>self-documenting</td>
<td>requires documentation</td>
</tr>
</tbody>
</table>
The Purpose of This Tutorial

This tutorial will teach you how to use Finite State Machine Editor for entering state machine diagrams and logic synthesis of the designs. We assume that you are familiar with other FOUNDATION SERIES applications such as Project Manager and Simulator.
Creating a State Diagram

The design created in this tutorial is a state machine which plays the Blackjack game. The design will be implemented in the XC9500 family device. The Blackjack game will be implemented according to the following description.

The purpose of the game is to select a number of cards the sum of which will be as close to 21 as possible. Each card has a decimal value between 2 and 11, where 11 is called ACE and can be counted as 1 if desired. The player with the highest number but lower than 21 will be the winner.

A request for an additional card is indicated by the SAY_CARD output (signal is High or logical 1), which is generated each time when the total sum of all cards is less then 17. However, it can never be generated if the total sum of cards exceeds 21.

The arrival of a new card is flagged by the NEW_CARD signal changing from 0 to 1.

The machine should output the SAY_BUST signal when it exceeds 21. In case the total score is between 17 and 21 the machine should flag the SAY_HOLD signal.

The total score should be shown on the TOTAL output. After reaching SAY_HOLD or SAY_BUST the machine should stay in the last state until NEW_GAME (reset) signal is flagged on the input.

The NEW_GAME signal resets the state machine.
Implementation

The whole machine implementation will be done by the Foundation Series software. The process consists of the following stages:

- drawing the state machine diagram using State Machine Editor
- generating VHDL code from the diagram
- synthesizing the design
- physical implementation of the design
Creating BLKJACK Project

1. Start **Project Manager**.
2. To create a new project - select the **Create New Project** option in the Getting Started dialog box.
3. Enter BLKJACK as the project name.
4. Select the Foundation Series F2.1i as the project **Type** and **HDL Flow**. As the design will be processes in the HDL flow you do not have to specify the target device at this point.

<table>
<thead>
<tr>
<th>New Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>Directory:</td>
</tr>
<tr>
<td>Type:</td>
</tr>
<tr>
<td>Flow:</td>
</tr>
</tbody>
</table>

5. Click the **OK** button to complete the new project setup. The **New Project** window will disappear and in its place a new **Project Manager** window will appear.

Note that the title bar of this window lists BLKJACK (the new name project) and informs that the design is not implemented.
Opening State Editor

1. To open the State Editor, click icon on the Flow tab of the Project Manager window.
2. Select Use HDL Design Wizard in the State Editor window
3. Click the OK button. This will start the Design Wizard.
Using HDL Design Wizard

The HDL Design Wizard allows you to enter ports and file names and to select hardware description language for design description.

To create a state diagram pattern:
1. Click Next > in the Design Wizard welcome window. The Design Wizard Language window will appear.

To begin creating the design, click Next.
2. Select the VHDL language in the Design Wizard Language window.

3. Click the Next > button. The Design Wizard - Name window will appear.

4. Type in BLKJACK as the file name and click the Next > button. The Design Wizard - Ports window for entering I/O signals (ports) will appear.

Note that the default file extension of state diagrams is ASF, and the new file will be called BLKJACK.ASF.
Adding Ports

To add a new port in the Design Wizard - Ports window, perform these operations:
1. Click the **New** button.
2. Enter the port name in the **Name** box.
3. Choose the port direction (Input, Output, Bi-directional) by selecting one of the options in the **Direction** field.

All assigned port names are displayed in the box above the **New** button and in the symbol preview area, which is located to the left of the port name listing.

Using the procedure described above enter the following ports:
- **input ports**: NEW_GAME, NEW_CARD, CARD[3:0]
  
  To enter the bus range [3:0], click the small buttons with arrows in the **BUS** field, located to the right of the **Name** box.
- **output ports**: SAY_CARD, SAY_HOLD, SAY_BUST;

  After entering the signal name, select the **Output** option in the **Direction** field.

---

**Design Wizard - Ports**

<table>
<thead>
<tr>
<th>U?</th>
<th>Name</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_GAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAY_CARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW_CARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAY_HOLD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARD[3:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAY_BUST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **New**
- **Delete**
- **Advanced**

To create a new port click **New**.

To change attributes of a port, select it on the list. Then you can change its name, range and direction; to set other attributes click **Advanced**.

To delete a port select it on the list and click **Delete**.

---

Note that all assigned I/O ports appear within the outline of the symbol, which is displayed on the left-hand side of the above window.

**NOTE:** Clicking the **Advanced** button displays the Advanced Port Settings window which allows you to specify the port type.
This window allows you to define the selected input signal as a clock, and the output ports as either combinatorial or registered. The **Registered** ports hold the set value in an additional register. The **combinatorial** outputs change any time the input conditions change.

**NOTE:** State Editor supports only synchronous state machines, which means that all transitions from one state to another are performed only at the clock transitions and a design cannot operate without a clock signal.

1. Click the **Next >** button when all the above listed ports have been entered.
   
   Since no clock signal has been defined, the **Design Wizard** window will prompt you for entering the clock signal.

2. Select **Yes** to add automatically the CLK port to the **State Editor** input port listing.

3. After the I/O ports setup has been completed and the CLK signal has been added, the **Design Wizard Machines** window will be displayed.

4. Select **One** and click the **Finish** button.
The new state diagram created by Design Wizard is shown below.

**NOTE:** If you select multiple machines, they will be converted into separate and concurrent processes in VHDL.
Adding Extra Ports

Additional I/O ports can be added at any time to the state diagram by clicking the **Add Port** buttons, located at the bottom of the vertical toolbar. Since we need an output port showing the total score of the Blackjack game, let’s add it now.

1. Click the **Output Port** button and then click above the SAY_BUST port symbol. A new output port named Port1 will be placed above the SAY_BUST port.
2. Click the port symbol you have just placed on the diagram with the right mouse button and select **Properties** from the local menu. The **Port Properties** window will appear.
3. Enter TOTAL in the **Name** box and select 4:0 in the **Range** box. Also select the **Output** and **Registered** options.
4. Click the **OK** button to complete the add port operation.

![Port Properties](image)

**NOTE:** Since the total value of all cards can be as high as 26 (the highest total at which a new card is drawn is 16, and the highest value of the new card can be 10), the TOTAL port has to have a binary range from 0 to 31 which requires 5 bits of data [4:0].

New output ports listed in the upper right-hand corner of the State Editor main screen should look the following:

- ![TOTAL[4:0]]
- ![SAY_BUST]
- ![SAY_HOLD]
- ![SAY_CARD]
Defining Additional Variables

The state machine needs to know if any of the cards is an ACE, because when the score exceeds 21 (with ACE counted as 11), the ACE can be counted as 1 and lower the total value below 21.

The following procedure describes how to define a new variable, called Ace.

1. Click the button on the left toolbar.
2. Place the cursor below Sreg0 and click the mouse button.
3. Click the signal icon on the sheet with the right mouse button and choose Properties from the shortcut menu. The Signal Properties window will appear.
4. Select Properties.
5. Type Ace in the name field.
6. Select the Boolean type.
7. Click OK.
8. Click the mouse button on the blank space of the sheet.
Adding a Reset State

Each state machine needs to be initialized. In case of the Blackjack game the initialization takes place when a new game is started. Let’s add such initialization of all output ports to the state diagram right now.

1. Click the button, located at the top of the vertical toolbar.
2. Place the state symbol in the middle of the sheet, as shown below (use the left mouse button to place a new state and the right mouse button to cancel the state placement mode). The new state will automatically be called S1.

3. To edit the state name, double click S1. An editing box will open.

4. Replace S1 with Start, by entering it in the editing box. To exit editing, click outside the edit box. The state bubble with the new name should look like this:

5. Select Sreg0 from the FSM command in the Machines menu. The Machine Properties window will be displayed. This window is used to define global settings of the selected state machine. Some of the global settings such as Sreg0 are named automatically.
6. Rename Sreg0 with Action in the Name box. This sets the name of the state machine signal used to store the current state in VHDL.

7. The Machine Properties window has several tabs for grouping different options within the window. Click the Reset tab and select NEW_GAME as the reset signal Name. Start as the reset State, synchronous reset Type, and high Active Level. This will assure that the NEW_GAME signal will be used as machine reset and when it is activated, the machine will automatically enter the Start state.

8. Click the OK button in the Machine Properties window. The state diagram should look as shown below:
NOTE: The triangular reset symbol is not a state; it indicates that the associated transition will be executed regardless of the current state of the machine, any time this the condition is met. This reset transition is for documentation purpose only.
Adding Initialization Actions

Once the reset condition is met, the machine will enter the **Start** state. To define actions that follow the reset, we will need to create an entry action. The entry action is automatically attached by the editor to the top of the state symbol and indicates that the specified code is executed only once when the machine enters this state.

1. Click the **Entry Action** button.
2. Position ‘dot end’ of the mouse pointer over the **Start** state and click the left mouse button.
3. In the edit box type in as follow:

   ```vhdl
   Ace := false;
   TOTAL <= "00000";
   SAY_CARD <= '0';
   SAY_HOLD <= '0';
   SAY_BUST <= '0';
   ```
4. Click the left mouse button outside the editing box.

   Note that you have to use the VHDL syntax:
   - lines end with semicolons
   - assignment operators <= can be used with ports and signals
   - assignment operator := is used with variables
   - quotation marks are used with string literals
   - apostrophes are used with character literals

   The above listed entry actions are shown below:

   ![Diagram](image)

   ```vhdl
   NEW GAME='1'
   ```

   **NOTE:** Combinatorial output port assignments in the reset state are also used as default values in other states. This means that the output will be set to the default value in all states, even where the value was not explicitly indicated. In this example, SAY_BUST will be ‘0’ in all states unless you add the SAY_BUST <= ‘1’ action.
Adding a State to Request a Card

Once the reset is performed, the state machine should start by requesting a card. This will be accomplished by adding the Hit_me state.

1. Add the Hit_me state below the Start state (use the procedure described above for entering the Start state).
2. To draw a transition between the Start and Hit_me, click the Transition button.
3. Next, click anywhere within the Start and Hit_me states. A line will be drawn between these states.
4. To exit the editing mode, click the left mouse button.

NOTE: The transition from the Start state to the Hit_me state is unconditional and it will be automatically executed in the next clock cycle, after entering the Start state (it means that the Reset state will last for one clock cycle).

To flag the request for a new card, the SAY_CARD should be set to ‘1’. Since this signal should remain active all the time while in this state, we have to use state action and not entry action.

NOTE: State actions are executed on each clock cycle as long as the machine remains in the given state. Entry Action is performed only when entering the selected state. The Exit Action is performed upon leaving the previously active state.

1. Click the State Action button.
2. Position the dangling ‘dot end’ of the mouse pointer over Hit_me state and click the left mouse button.
3. Type SAY_CARD <= ‘1’ in the edit box.
4. Click the left mouse button outside the edit box. After this operation the state diagram will look as shown below:
total := "00000";
Ace := false;
SAY_CARD <= '0';
SAY_HOLD <= '0';
SAY_BUST <= '0';

NEW_GAME='1'

Start

Hit_me

SAY_CARD <= '1'
HDL Code Generation

At this point, some section of the state machine has been created. Before the design gets bigger we will inspect the being generated code to show how the diagram elements correspond to the generated code.

**NOTE:** HDL Code generation checks for some diagram problems but does not check your VHDL statements. The full syntax check will be performed later when the design is synthesized.

1. Activate HDL Code Generation from theSynthesis menu.
2. The State Editor dialog box will appear. Click Yes to view the generated code.

**NOTE:** It is essential that you learn to relate the code that is being generated to the elements on the drawing. State editor uses standard templates to create the VHDL design. Understanding VHDL constructs is very helpful when debugging a design with a VHDL simulator.

VHDL code generated by the State Editor can be divided into several sections.

- **Library section** is always added at the beginning and provides the access to the IEEE library.

  ```vhdl
  library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
  ```

- **Entity declaration section** lists all ports defined on the state diagram.

  ```vhdl
  entity blkjack is
    port (CARD: in STD_LOGIC_VECTOR (3 downto 0);
          CLK: in STD_LOGIC;
          NEW_CARD: in STD_LOGIC;
          NEW_GAME: in STD_LOGIC;
          TOTAL: out STD_LOGIC_VECTOR(4 downto 0);
          SAY_BUST: out STD_LOGIC;
          SAY_CARD: out STD_LOGIC;
          SAY_HOLD: out STD_LOGIC);
  end;
  ```

- **Global declarations section defines:**
  - enumerated type and state variable of the selected type for every machine on the diagram.
  - objects and actions common to all machines.

  ```vhdl
  architecture blkjack_arch of blkjack is
  --auxiliary diagram declarations
  --diagram DECLARATIONS;

  -- SYMBOLIC ENCODED state machine: Action
type Action_type is (Hit_me, Start);
signal Action: Action_type;

  begin
  --concurrent signal assignment
  ```
--diagram ACTIONS;

- **Machine declarations section** defines objects local to each machine (process).

```vhdl
process (CLK)
    -- auxiliary machine declarations
    -- machine DECLARATIONS
    variable Ace: boolean;
```

- **Reset definition section**

```vhdl
begin
    if CLK'event and CLK = '1' then
        if NEW_GAME = '1' then
            Action <= Start;
            TOTAL <= "00000";
            Ace := false;
        else
            case Action is
                when Hit_me =>
                    Action <= Hit_me;
                when Start =>
                    Action <= Hit_me;
                when others =>
                    null;
                end case;
        end if;
    end if;
end process;
```

- **Machine action description section**

```vhdl
else
    case Action is
        when Hit_me =>
        when Start =>
            Action <= Hit_me;
        when others =>
            null;
    end case;
end if;
end process;
```

- **Output port assignment section**

```vhdl
-- signal assignment statements for combinatorial outputs
SAY_CARD <= '1' when (Action = Hit_me) else '0';
SAY_HOLD <= '0';
SAY_BUST <= '0';
end blkjack_arch;
```
Receiving and Handling of the New Card

Once a new card is requested, the machine should wait until it is received. The arrival of the new card will be marked by the NEW_CARD signal. The following will add a design section that handles the NEW_CARD signal.

1. Add a new state below the Hit_me state.
2. Rename the new state to Got_it, using the same procedure that was used for changing S1 to Start.
3. Click the right mouse button inside the state bubble, but outside the state name.
4. Select Properties from the shortcut menu.
5. Ensure that the Got_it name is in the Name field. Click OK.

6. Add a transition from the Hit_me to Got_it state.
7. Add a condition to the recently drawn transition.
8. Click the (Condition) button.
9. Click the transition line.
10. Type NEW_CARD='1' in the edit box.
11. Click the mouse button outside the edit box.

NOTES:
- state machine remains in Hit_me state until the NEW_CARD='1' condition is met
- SAY_CARD output returns to the default '0' value when the machine exits Hit_me state. This is because the '0' default value was defined in the Reset state.
- all actions are performed on the rising edge of the CLK signal, which has been defined in the Machine properties dialog box and can be changed if desired.
- Once the card was received, the total score needs to be updated. Also, the card must be checked if it is an Ace (11). If it is, the ACE flag will be set.

Since these actions need to be executed only once when entering the state the entry action will be used. To assign entry actions to the state, use the Actions card in the State Properties window:
1. Click the right mouse button inside the state bubble.
2. Select Properties in the shortcut menu.
3. Click the Actions tab.
4. Type in the Entry box:
   TOTAL<=TOTAL+CARD;
   Ace:=(CARD=11) or Ace;
5. Click the OK button to complete the operation.

**NOTE:** The above described actions must be defined as entry actions. Otherwise, the total will be incremented on each active clock edge as long as the machine remains in the Got_it state.

State diagram should now look the following:
Analyzing the Total Score

Once the new card is received, the total card score should be updated and tested. If it is less then 17, then the state machine should request a new card and go back to the Hit_me state.

**Adding a new card:**
1. Add the Test16 state below the Got_it state
2. Add transition from the Got_it state to the Test16 state
3. Add NEW_CARD='0' condition to the above defined transition.

**NOTE:** Adding the NEW_CARD='0' condition prevents the state machine from using the same card more than once.

After completing this step the state diagram will look as shown below:
Detecting Hold/Bust conditions:
1. Add the Test21 state below the Test16 state.
2. Add transition from the Test16 state to the Test21 state; assign to it the TOTAL > 16 condition.
3. Add a transition from the Test16 state to the Hit_me state; assign to it the @ELSE condition.

**NOTE:** The @ELSE transition is executed when no other conditions are met in this state. This transition will be executed if the TOTAL value is 16 or less and will cause the machine to request a new card in the Hit_me state.

The state diagram with 5 states should look the following:
Testing the Final Score

Once the machine gets to the Test 21 state, the total score is 17 or more. Now the state machine has to test if it did not exceed 21. If it did not, it will flag the SAY_HOLD signal. If the score is over 21, the SAY_BUST signal should be activated, indicating that the machine has lost the game. Before flagging the SAY_BUST signal, the machine should check if any of the previously drawn cards was an ACE. If there was an Ace, it may decrement the total value by 10 to recover from the “bust” situation.

1. Add two more states: *Bust* and *Hold*.

![State Diagram]

2. Add a transition from the *Test21* state to the *Bust* state and from the *Test21* state to the *Hold* state.
3. Add TOTAL < 22 condition to the *Test21 ⇒ Hold* transition.
4. Add SAY_HOLD <= '1' state action to the *Hold* state.
5. Add a transition from the *Test21* state to the *Test16* state with the ACE condition.

**NOTE:** Since ACE is a Boolean variable, no relational operators are required in the condition text

6. Add TOTAL <= TOTAL-10; action to the above defined transition (use the button).
7. Click the transition line and type the action text.

**NOTE:** Transition actions allow avoiding redundant states. If you would not assign this action to a transition, then another state would have to be created to execute the TOTAL <= TOTAL-10; action.

8. Add @ELSE condition to the *Test21 ⇒ Bust* transition.
9. Add SAY_BUST <= '1' state action to the *Bust* state.

The final state diagram with all states, transitions, conditions, and actions should look as shown below
Assigning Condition Priorities

NOTE that in the Test21 state both TOTAL < 22 and ACE conditions can be met at the same time and behavior of the machine depends on condition evaluation sequence. To avoid confusion, you should assign different priorities to each transition. Because our machine should first check if TOTAL is less than 22, assign the priority 1 to “Test21 ⇒ Hold” transition and the priority 2 to “Test21 ⇒ Test16” transition.

To change the transition priority:
1. Click the transition line.
2. Click the right mouse button.
3. Select Priority from the shortcut menu.
4. Select the required priority level.
5. Click the mouse button.

NOTE: Transitions without assigned priority are executed last
Selecting State Encoding

When state machines are compiled into logic, the current states are stored in a register, being a group of flip-flops. Each state is assigned to a separate flip-flop. However, if there are some concurrent state transitions, they may create a glitch in the combinatorial logic and the state machine can enter a wrong state. The recommended encoding type for all FPGAs is One-Hot which assigns such values to each state so that only one bit of a register is active at any one time. One hot encoding, however, consumes more flip-flops because each state requires a bit in a register and for that reason is not recommended for CPLDs which lack flip flop resources. State Editor supports one-hot and binary encoding. Other types of encoding can be created by manually assigning the codes to each state.

The binary encoding method will be used in this tutorial for simplicity of analysis in the simulator.
1. Activate Action from the Machines command in the FSM menu.
2. Select the Binary Encoded option from the Machine Properties window.

```
Machine Properties

Name: Action
Clock: CLK
Encoding: Binary

OK Cancel Apply Now
```

**NOTE:** The Symbolic encoding command allows the synthesis tool to select automatically the preferred state encoding method. Metamor XVHDL defaults to binary encoding if the codes are not assigned any values.

3. Select Preferences from the Tools menu and check the Display State Codes command.
4. Click the OK button to complete the operation.
Preferences

Graphics

- Text Scaling
- Display State Codes

OK  Cancel
Analyzing the Design

After entering the design source in the State Editor the diagram file has to be added to the project. When you add the diagram to the project, the VHDL code is generated, and then analyzed by the synthesis tool.

1. Save the diagram.
2. Select **Add to project** from the **Project** menu in the **State Editor** menu. This will add the top level document the **Project Manager** hierarchy navigator.
3. Switch to the **Project Manager** window.
4. In Project Manager you will see the `blkjack.asf` file icon displayed on the **Files** tab. The red cross over the icon shows that errors have been detected during source code analysis.
Locating and Correcting Errors

The output ports cannot be used for reading by a state machine. If reading of a port is required, it should be defined as bi-directional. Note that the ‘TOTAL’ signal is not readable because it has been assigned the OUT mode and an error will be reported every time TOTAL appears on the right-hand side of signal assignment.

To correct this error:
1. Switch to State Editor.
2. Click TOTAL port symbol with the right mouse button.
3. Select Properties in the shortcut menu. The Port Properties dialog box will open.
4. Change the port type to Bi-directional.
5. Save the diagram.
6. Switch to Project Manager. Note that after editing the diagram its icon has changed to blkjack.asf, indicating that the file needs analysis.
7. Click the icon with the right mouse button, and then choose Analysis from the shortcut menu. This time there should be no errors and the file icon should changed to blkjack.asf. To the left from the file icon the small + is displayed. Click this sign to see the design hierarchy. There is only one design entity called blkjack:

   □ blkjack.asf
   □ blkjack
To synthesize the code generated from the state diagram click the Synthesis icon on the Flow tab. When the Synthesis/Implementation dialog box comes up set the available options as shown below:

Click OK and wait until the synthesis is finished. Switch to the Versions tab. You should see the hierarchy as shown below:

Click the version icon with the right mouse button, and then select View Report from the shortcut menu to open the synthesis report for viewing.

NOTE: Synthesis report is very important, because it shows how many resources have been used by the design. Making some changes on the diagram may drastically increase or decrease the number of used flip-flops (CLBs).
Functional Simulation

The next step in design implementation is optional functional simulation. It allows the user to check if the design meets the requirements against the functionality.

1. Click the button on the Flow tab to start the Logic Simulator.
2. Select Add Signals from the Signal menu.
3. Double-click CLK, NEW_GAME, NEW_CARD, (CARD0,CARD3), (TOTAL0,TOTAL4), SAY_BUST, SAY_CARD, SAY_HOLD, and ACTION_0, ACTION_1, ACTION_2 signals to select them for observation in the Waveform Viewer.
4. Close the dialog box.

**NOTE:** ACTION_0, ACTION_1, ACTION_2 represent the state variables. They enable monitoring of the current machine states.

The selected signals should look as in the picture below:

```
<table>
<thead>
<tr>
<th>i_CLK</th>
<th>i_NEW_GAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_NEW_CARD</td>
<td></td>
</tr>
<tr>
<td>l_CARD_0_ibuf</td>
<td></td>
</tr>
<tr>
<td>l_CARD_1_ibuf</td>
<td></td>
</tr>
<tr>
<td>l_CARD_2_ibuf</td>
<td></td>
</tr>
<tr>
<td>l_CARD_3_ibuf</td>
<td></td>
</tr>
<tr>
<td>l_TOTAL_0_in</td>
<td></td>
</tr>
<tr>
<td>l_TOTAL_1_in</td>
<td></td>
</tr>
<tr>
<td>l_TOTAL_2_in</td>
<td></td>
</tr>
<tr>
<td>l_TOTAL_3_in</td>
<td></td>
</tr>
<tr>
<td>l_TOTAL_4_in</td>
<td></td>
</tr>
<tr>
<td>o_SAY_BUST</td>
<td></td>
</tr>
<tr>
<td>o_SAY_CARD</td>
<td></td>
</tr>
<tr>
<td>o_SAY_HOLD</td>
<td></td>
</tr>
<tr>
<td>l_Action_0</td>
<td></td>
</tr>
<tr>
<td>l_Action_1</td>
<td></td>
</tr>
<tr>
<td>l_Action_2</td>
<td></td>
</tr>
</tbody>
</table>
```

**NOTES:**
- L indicates the least significant bus line. To ensure the correct display of bus values, check if L is assigned to bus displays element with index 0. If L is displayed with the highest bus member, you can reverse the bus order by clicking bus names, while holding the Shift key, and then selecting the Change Direction option of the Bus command in the shortcut menu.
- The Waveform Viewer window can display buses either as expanded or compacted. To change the display mode, click the button.

5. Select Add Stimulators from the Signal menu and assign B0 stimulator to the CLK input
6. Click Formula in the Stimulator Selection window and define F0, F1, and F2 stimulators:
7. Assign \((H20L580)10\) formula to F0 stimulator.
8. Assign \((L80H20)10\) formula to F1 stimulator.
10. Click Close.
11. Attach the F0 stimulator to the NEW_GAME input, F1 stimulator to the NEW_CARD input, and F2 stimulator to the CARD input bus (this bus represents the card values from 2 to 11).
12. While holding the Ctrl key, select CARD, ACTION_0 and TOTAL buses.
13. Select the Display Decimal option of the Bus command from the Signal menu.

The signal names in the Waveform Viewer should look as below.
14. Simulate using the **Step** and **Long** (step) buttons until machine displays signals SAY_HOLD or SAY_BUST.

**NOTE:** Remember that ACTION bus value is equal to the current state code. You can change the card sequence by editing the **F2** stimulator formula.

Sample simulation results are shown below.

<table>
<thead>
<tr>
<th>CLK</th>
<th>B0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NEV_GAME</td>
<td>F0</td>
<td></td>
</tr>
<tr>
<td>NEV_CARD</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>E/Card_0_ibuf.</td>
<td>F2</td>
<td>A</td>
</tr>
<tr>
<td>D/TOTAL_0_in.</td>
<td>1</td>
<td>OA</td>
</tr>
<tr>
<td>SAY_BUST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAY_CARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAY_HOLD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B/Action_0 (hex)</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Physical Implementation

The last step in processing the project is physical implementation of the design.

To initiate the process click the **Implementation** button on the **Flow** tab. Wait until the process is finished, and then switch to the **Versions** tab. The implementation hierarchy should look as shown below:

- **ver1 (XC9509-9536PC44-10)**
- **ver1 (functional structure)**
- **ver1-Optimized (optimized structure)**
- **ver1 (Implemented, OK)**

Now the design is ready for either timing verification or device programming. There are two timing verifications tools available from the **Flow** tab: **Timing Simulation** or **Timing Analyzer**. Using them allows the user to check if the design implementation meets the timing requirements before programming the device.
Conclusion

As shown above, the State Editor is a powerful tool for creating control logic designs in HDL languages. The graphical diagrams provide excellent documentation of the design and allow generating efficient and reliable synthesis results.