Active-HDL Help

State Machine Entry and Debugging Tutorial

State Machine Entry and Debugging Tutorial

Introduction

State machine editors allow simple and easy graphical design entry. Since the state machine designs can be easily retargeted to any devices, the state editors are becoming very popular with designers who value technological independence. This tutorial is dedicated for Active-HDL users who want to learn how to design with a State Diagram Editor and gain greater device and vendor independence.

The Purpose of This Tutorial

This tutorial will teach you how to use the **State Diagram Editor** for entering state machine diagrams and perform logic synthesis of the designs.

Creating a State Diagram

The sample design created in this tutorial is a state machine that plays the *Blackjack* game. The purpose of the game is to select a number of cards which sum 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 or equal to 21 will be the winner.

A request for an additional card is indicated by the *SAY_CARD* output (this signal is High or logical 1 when active), 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 the sum of cards exceeds 21. In case the total score is between 17 and 21, the machine should output the SAY_HOLD signal.

The total score should be shown on the *TOTAL* output. After reaching the *SAY_HOLD* or *SAY_BUST* condition, the machine should stay in the last state until the *NEW_GAME* signal is flagged on the input. The *NEW_GAME* signal also resets the state machine.

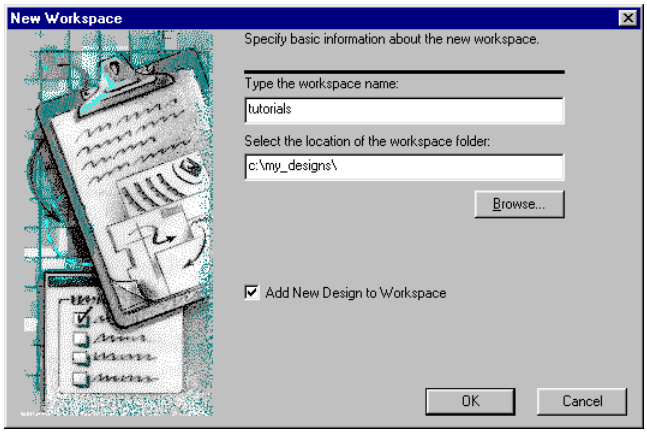
Implementation

The machine will be implemented by drawing a state diagram. The diagram will be automatically converted into the corresponding VHDL code. Then, you can verify the functionality of the machine by simulating the code in the VHDL simulator.

Creating BlackJack Project

1. Start **Active-HDL. The Getting started** window will appear.

2. Select the **Create New Workspace** option and click **OK**. This will open the **New Workspace** wizard.

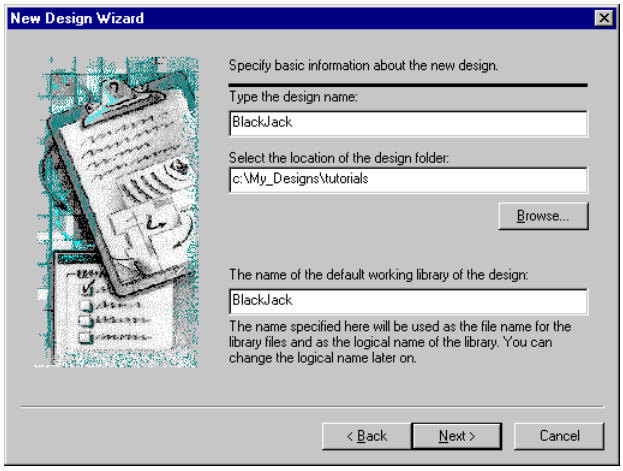


Provide the name for the workspace. Make sure that the **Add New Design to Workspace** checkbox is selected. Click the **OK** button. As a result, the new workspace will be created and the **New Design Wizard** will start.

3. Select the **Create an empty design** option, and then click **Next**.

4. Specify desired synthesis and implementation tools for the current design (if there are any installed). Choose default family, block diagram configuration and default HDL language. Click the **Next** button.

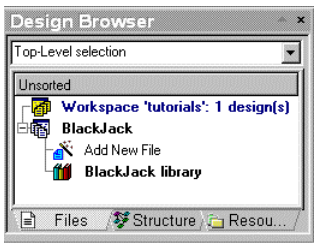
5. Enter *BlackJack* as the project name.



Click **Next**.

6. The last wizard dialog shows the complete project description. Click **Finish** to complete the creation of the new project or click **Back** to go back to previous dialogs to modify the settings.

After completing the last Wizard task the **Design Browser** window should contain the following:



Using the New Source File Wizard

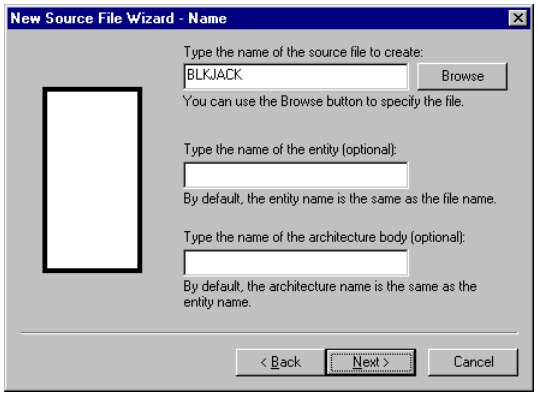
To open a new **State Editor** window, select **State Diagram** from the **File | New** menu. A series of wizard windows will appear. Use them to create a new diagram. To create a state diagram skeleton:

1. Choose **New | State Diagram from the File menu.** The **New Source File Wizard** window will appear.

Select the **Add the generated file to the design** check box, and then click the **Next** button.

2. Select the **VHDL** language option for the generated code, and click the **Next** button.

3. Type in **BLKJACK** as the file name, and click the **Next** button.

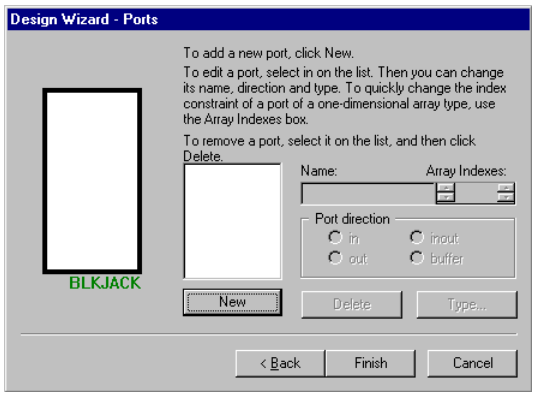


The **Design Wizard - Ports** window will appear.

NOTE: The default file extension of the state diagram file is *.ASF. The new file will be named BLKJACK.ASF.

Adding Ports

To add a new port in the **Design Wizard - Ports** window, perform the following operations:



1. Click the **New** button.

2. Enter the port name in the **Name** box.

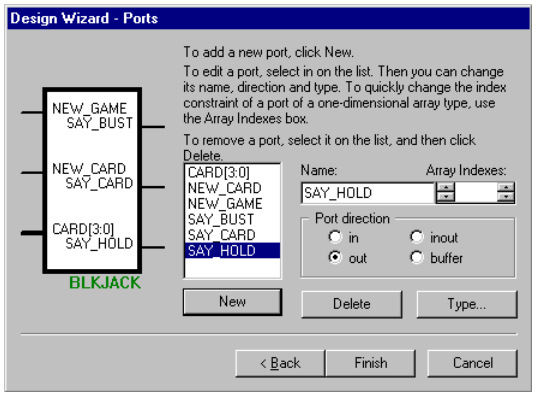
3. Choose the port direction (In, Out, Inout and Buffer) by selecting one of the options in the **Port direction** field.

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:

```
in ports: NEW_GAME, NEW_CARD, CARD[3:0]
```

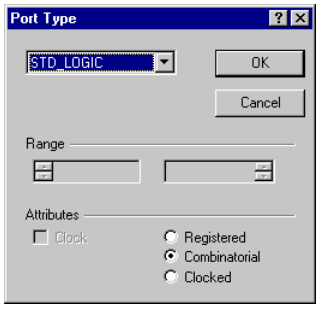
To enter the bus range [3:0] for the CARD signal, click the small buttons with arrows in the **Array Indexes** field, located to the right of the **Name** box.

```
out ports: SAY_CARD ,SAY_HOLD ,SAY_BUST;
```



After entering each of these signal names, select the **Out** option in the **Port direction** field.

Note that specified ports appear within the outline of the symbol, which is displayed on the left-hand side of the above window.

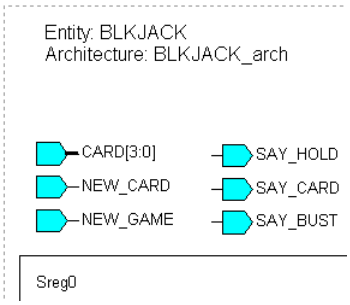


NOTES:
1. The **Type** button displays the **Port Type** window that allows you to specify the port type.
2. The State Diagram Editor supports only synchronous state machines, which means that all transitions from one state to another are performed only at the clock transitions. A design cannot operate without a clock signal.

Click **Finish** to close the wizard.

4. When the wizard prompts you for adding a clock port, click **Yes.**

The new state diagram created by the **Design Wizard** is shown below:



Adding Extra Ports

Additional I/O ports can be added to the state diagram at any time by clicking the **Output/Input Port** toolbar buttons. Since we need an output port showing the total score of the Blackjack game, lets add it now.

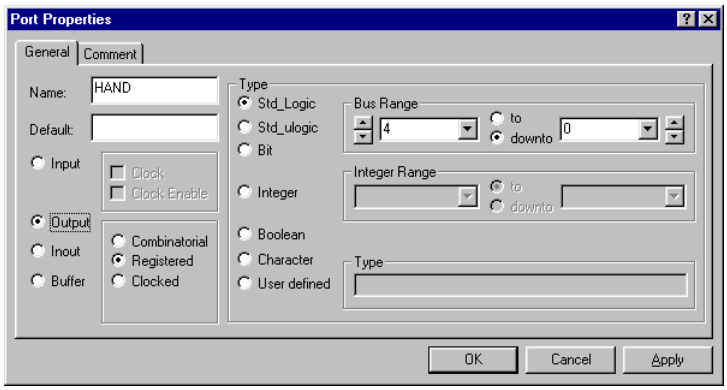
1. Click the **Output Port** button and then click above the *SAY_HOLD* port symbol. A new output port named *Port1* will be placed above the *SAY_HOLD* port.

2. Click the port symbol you have just placed on the diagram with the right mouse button and select **Properties** from the shortcut menu. The **Port Properties** window will appear.

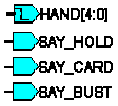
NOTE: Since the total value of all cards can be as high as 26 (the highest total at which a new card is drawn is 16 and the highest value of the new card can be 10), the range of the *HAND* port must be 0 to 31, which requires a 5-bit port.

3. Enter HAND in the **Name** box and select *4:0* in the **Range** box. Also, select the **Output** and **Registered** options.

4. Click **OK** to complete the operation.



The **Port Properties** 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. New output ports listed in the upper right-hand corner of the State Diagram Editor main screen should look as shown below:



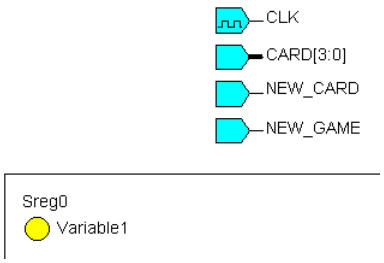
Defining Additional Variables

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 to lower the total value below 21.

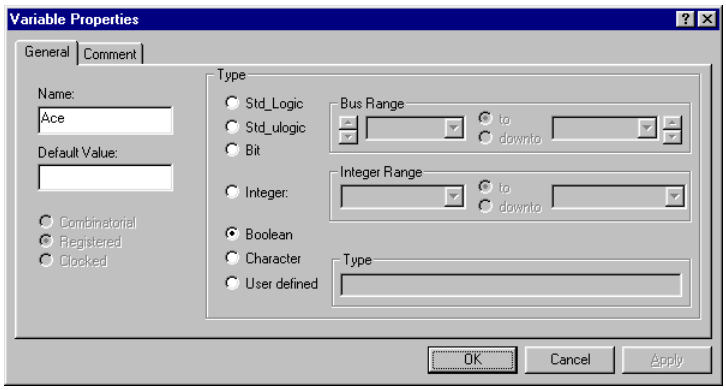
The following procedure describes how to define a new variable called *Ace*:

1. Click the **Signal** button on the left tool-bar  .

2. Place the cursor below the *Sreg0* text that is located in the upper left-hand corner of the diagram area outline.



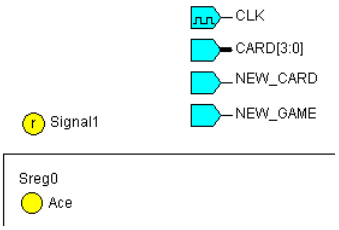
3. Switch to the **Select** mode, Click the *Variable1* icon with the right mouse button and choose **Properties** from the shortcut menu. The **Variable Properties** window will appear.



- Type *Ace* in the **Name** field.
- Select the *Boolean* type.
- Click **OK** to complete the operation.

Defining Additional Signals

Since the *HAND* port is the vector of the out mode, the state machine would not be able to read from it. We need an additional signal that will store the current total sum of already received cards. It should be of the same size as the *HAND* port, lets call it *TOTAL*.



1. Click the **Signal**  button on the toolbar.

2. Place the cursor below the *//diagram Actions* text which is located in the upper left-hand corner of the diagram workspace.

3. Click the *Signal 1* icon with the right mouse button and choose **Properties** from the shortcut menu. The **Signal Properties** window will appear.

Change the name of the signal from *Signal1* to *TOTAL* in the **Name** box and select *4:0* in the **Range** box.

Adding Machine Action

Now that there is a *TOTAL* signal we need to assign its value to the *HAND* port. Such an assignment should be performed concurrently. This can be done by using *Diagram Actions*. To add the Diagram Action object, choose **Diagram** from the **FSM | Action** menu.

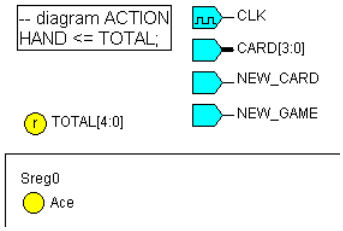
1. After choosing the **Action** command, the shape of the cursor changes. Locate the cursor with the attached frame outside the machine frame and click the left-mouse button. You have just added the Diagram Action object and it is now in the Edit mode.

2. Enter a new line and type in the following assignment:

```
HAND <= TOTAL;
```

3. Finish editing the Diagram Action object.

The diagram should look as follows:

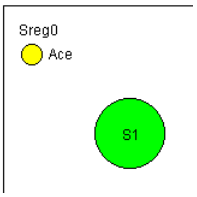


Adding a Reset State

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

1. Click the **State** button  located at the top of the vertical toolbar.

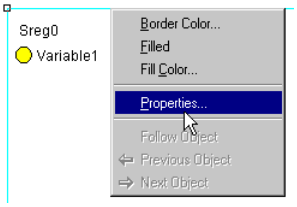
2. Place the state symbol in the middle of the sheet, as shown below (use the left mouse button to place a new state and the right mouse button to cancel the state placement mode). The new state will automatically be called *S1*.



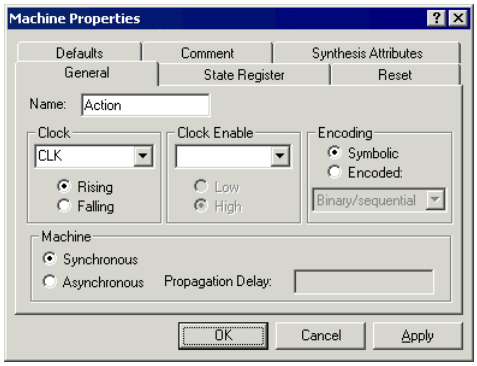
3. To edit the state name, double click *S1. An editing box will appear.*

4. Replace *S1* with *Start* by entering it in the editing box. To exit editing, click outside of the edit box. The state bubble with the new name is shown below:

5. Click within the blank area with the right mouse button and select **Properties** from the shortcut menu.



The **Machine Properties** dialog box will open. This window is used to define global settings of the selected state machine. Some of the global settings such as *Sreg0* are named automatically.

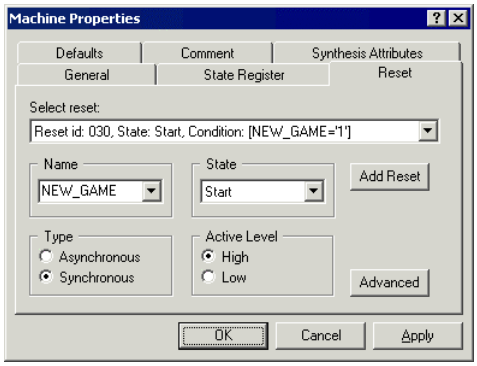


6. Replace *Sreg0* listed in the **Name** box with *Action*. This sets the name of the state machine signal used to store the current state in VHDL.

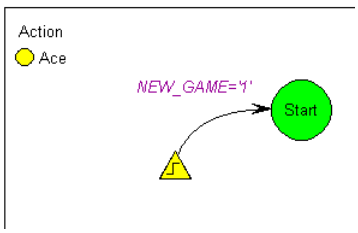
The **Machine Properties** window has several tabs for grouping different options within the window. Switch to the **Reset** tab, click the **Add Reset** button and select the following:

- *NEW_GAME* as the reset signal **Name**
- *Start* as the reset **State**
- *Synchronous* reset **Type**
- *High* **Active Level**

This will assure that the *NEW_GAME* signal will be used as machine reset and when it is activated or at logical 1 the machine will automatically enter the *Start* state.



7. Click the **OK** button in the **Machine Properties** window. The state diagram should now look as shown below:

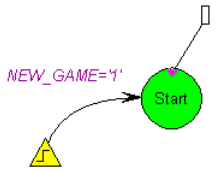


NOTE: The triangular reset symbol is not a state; it indicates that the associated transition will be executed any time this condition is met and will be carried out regardless of the current state of the machine. The reset transition is shown for documentation purpose only.

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 to the top of the state symbol and indicates that the specified code is executed only once, when the machine enters this state.

1. Choose the **Action/Entry** option form the **FSM** menu or click the **Entry Action** button  if it is available on the toolbar.



2. Position dot end of the mouse pointer over the *Start* state and click the left mouse button. This will show an edit box for entering the initial design conditions.

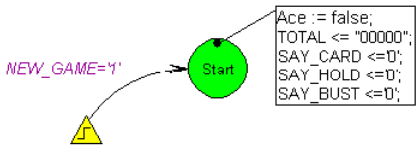
3. In the edit box, type in as follows:

```
Ace := false;
TOTAL <= "00000";
SAY_CARD <='0';
SAY_HOLD <='0';
SAY_BUST <='0';
```

Note that you have to use the VHDL syntax:

- lines must 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

4. Click the left mouse button outside the editing box. The above listed entry actions are shown in the figure:



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 a reset has been 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 then *Hit_me* states. A line will be drawn between these two states.

4. To exit the editing mode, click the left mouse button outside the state bubbles.

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). However, additional cards will be drawn only if the *SAY_CARD* is at 1.

New cards are being drawn only in the *Hit_me* state and only when the *SAY_CARD* is set to 1. For this reason, this signal should remain active all the time in this state. To accomplish this, we have to use the *state action* and not the *entry action*.

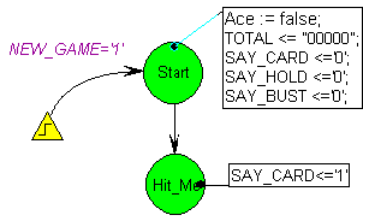
NOTE: State actions are executed on each clock cycle as long as the machine remains in the given state. **Entry Action** is performed only while entering the selected state. The **Exit Action** is performed upon leaving the previously active state.

1. Click the **State Action** button 

2. Position the dangling dot end of the mouse pointer over the *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 editing box. After this operation the state diagram will look as shown:



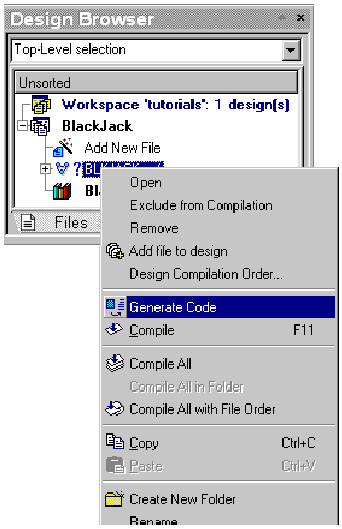
HDL Code Generation

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

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

1. Using the **File** tab of the **Design Browser,** click the *Blkjack.asf* file with the right mouse button.

2. Choose the **Generate Code** option from the shortcut menu 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. The State Diagram Editor uses standard templates to create the VHDL design. Understanding VHDL constructs is very helpful while debugging a design with a VHDL simulator.

Choose the **View HDL Code** from the **FSM** menu to see generated VHDL code.

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

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

```
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 in the state diagram.

```
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;
    HAND: 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 in the diagram.
- objects and actions common to all machines.

```
architecture BlkJack of BlkJack is
-- diagram signals declarations
signal TOTAL: STD_LOGIC_VECTOR (4 downto 0);
-- BINARY ENCODED state machine: Action
attribute enum_encoding: string;
type Action_type is (HIT_ME, START);
attribute enum_encoding of Action_type: type is
  "0 " & -- Start
  "1" ; -- Hit_Me
signal Action: Action_type;
begin
-- concurrent signals assignments
--diagram ACTIONS
HAND <= TOTAL;
```

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

```
Action_machine: process (CLK)
--machine variables declarations
variable Ace: BOOLEAN;
```

- **Reset definition section** defines behavior of the machine on initialization

```
Begin
  if CLK'event and CLK = '1' then
    if NEW_GAME='1' then
      Action <= Start;
      Ace :=false;
      TOTAL <=00000";
```

- **Machine action description section** defines actions performed in specific states and transitions between the states

```
else
 case Action is
  when HIT_ME =>
  when Start =>
```

```
    Action <=HIT_ME;
  when others =>
    null;
 end case;
end if;
end if;
end process;
```

- **Output port assignment section** defines logic modeling combinatorial outputs

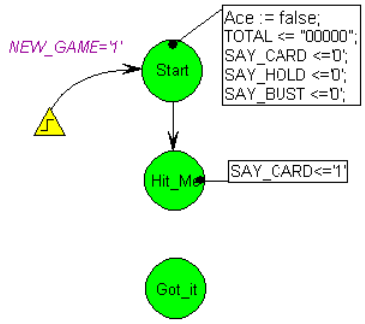
```
-- signal assignment statements for combinatorial outputs
SAY_CARD_assignment:
SAY_CARD <= '0' when (Action = Start) else
            '1' when (Action = Hit_Me) else
            '0';
SAY_HOLD_assignment:
SAY_HOLD <= '0' when (Action = Start) else
            '0';
SAY_BUST_assignment:
SAY_BUST <= '0' when (Action = Start) else
            '0';
end BlkJack;
```

Receiving and Handling of the New Card

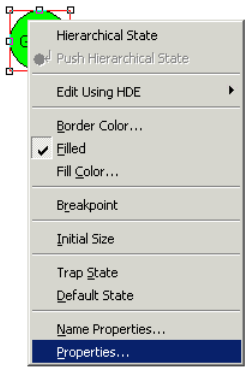
Once a new card has been requested, the machine should wait until it is received. The arrival of the new card will be marked by the *NEW_CARD* signal. The following will add a design section that handles the *NEW_CARD* signal:

1. Add a new state below the *Hit_me* state.

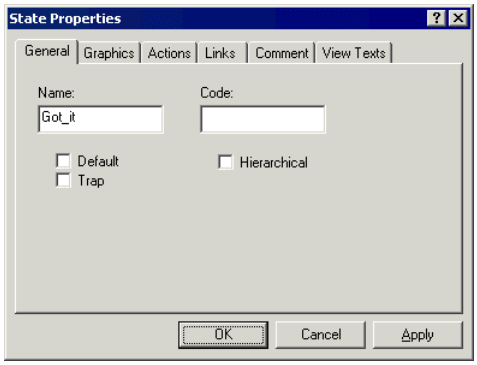
2. Rename the new state *Got_it,* using the same procedure that was used for changing *S1* to *Start.*



3. Click the right mouse button inside the state bubble, but outside the state name.

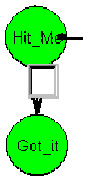


4. Select **Properties** from the shortcut menu.



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. Click the **Condition** button  to add a condition to the newly drawn transition.

8. Click the transition line.

9. Type *NEW_CARD='1'* in the edit box.

10. Click the mouse button outside the edit box.

NOTES:

- The state machine remains in the *Hit_me* state until the condition *NEW_CARD = 1* is met.
- The *SAY_CARD* output returns to the default 0 value when the machine exits the *Hit_me* state. This is caused by the 0 default value that has been defined in the *Reset* state.
- All actions are performed on the rising edge of the CLK signal that can be change, if desired.
- Once the card has been received, the total score needs to be updated. Also, the card

must be checked if it is an *Ace* (11). If it is so, the *ACE* flag will be set.

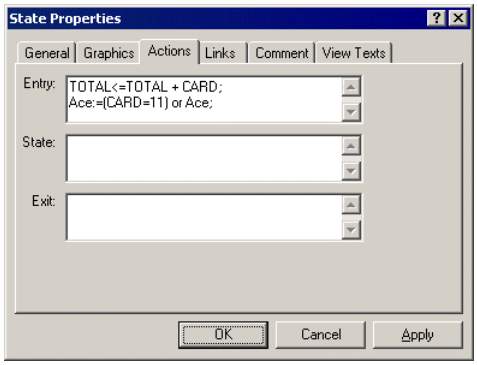
Since these actions need to be executed only once while entering the state, the *entry action*  will be used.

To assign entry actions to the state, use the **Actions** tab in the **State Properties** window:

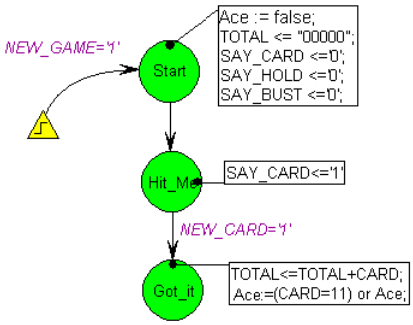
1. Click the state symbol with the right-mouse button.
2. Select **Properties** in the shortcut menu
3. Click the **Actions** tab.
4. Type in the **Entry** box:

```
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. The state diagram should now look as shown next:



Analyzing the Total Score

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

Adding a New Card

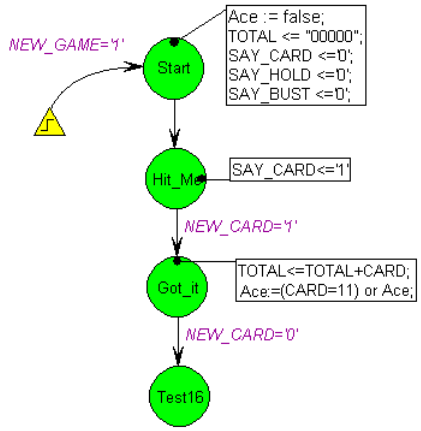
1. Add the *Test16* state below the *Got_it* state.

2. Add transition from the *Got_it* state to the *Test16* state.

3. Add the *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 next:



Detecting Hold/Bust Conditions

1. Add the *Test21* state below the *Test16* state.

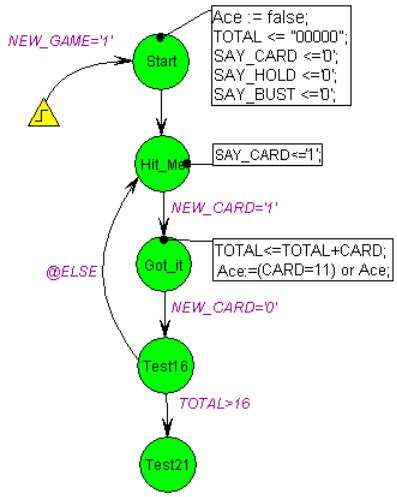
2. Add a transition from the *Test16* state to the *Test21* state; assign to it the *TOTAL > 16* condition.

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.

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 has not exceeded 21. If 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 *ACE*. If there was *Ace,* it may decrement the total value by 10 to recover from the bust situation.



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

2. Add a transition from the *Test21* state to the *Hold* state.

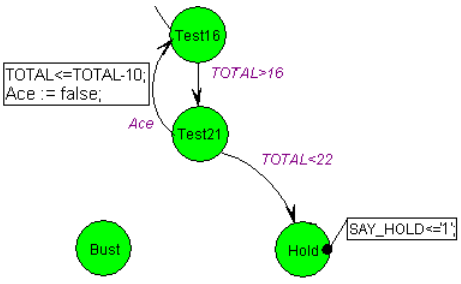
3. Add *TOTAL<22* condition to the Test21 => Hold transition.

4. Add the *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. Using the **Transition Action** button  , add *TOTAL <= TOTAL-10;* and *Ace:=false;* actions to the above defined transition. To do so, click the transition line and type in the action text.

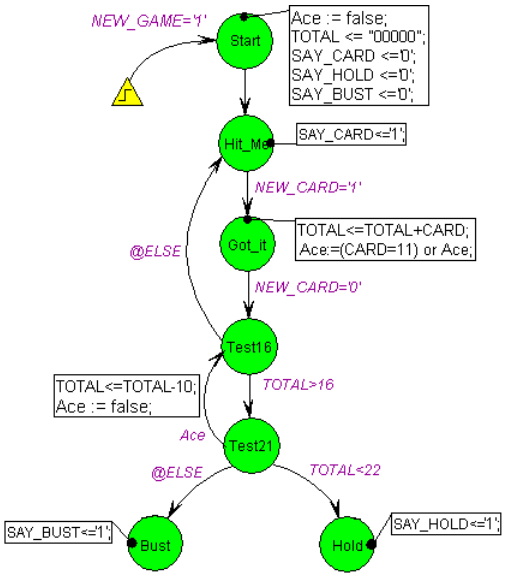


NOTE: Transition actions allow you to avoid redundant states. If you did not assign this action to a transition, another state would have to be created to execute the *TOTAL<= TOTAL-10;* action.

7. Add the *Test21* to the *Bust* state transition and the *@ELSE* condition to it.

8. Add the *SAY_BUST <= 1;* state action to the *Bust* state.

The final state diagram with all the states, transitions, conditions, and actions should look as follows:

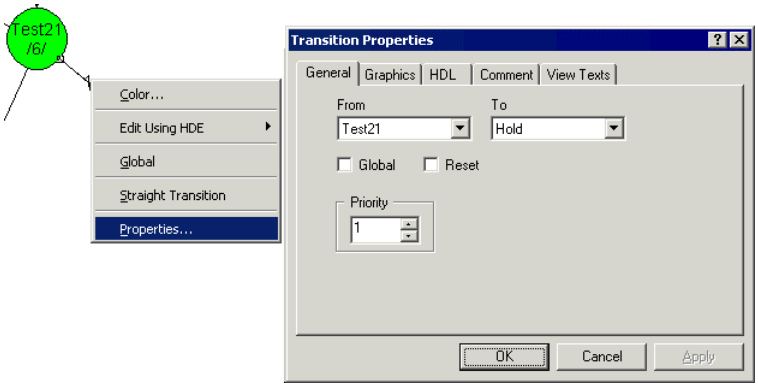


Assigning Condition Priorities

Note that in the *Test21* state, both the *TOTAL<22* and *ACE* conditions can be met at the same time. In such a case, 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 the Test21 => Hold transition and the priority **2** to the Test21 => Test16 transition.

To change the transition priority:

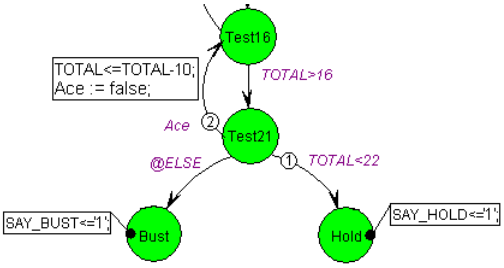
1. Click the transition line.



2. Click the right mouse button to display the shortcut menu.

3. Select the **Properties** menu option, the **Transition Properties** dialog box will open.

4. Select the required priority level using the **Priority** edit box.



4. Click the mouse button to complete the operation.

NOTE: Transitions without assigned priority are executed last.

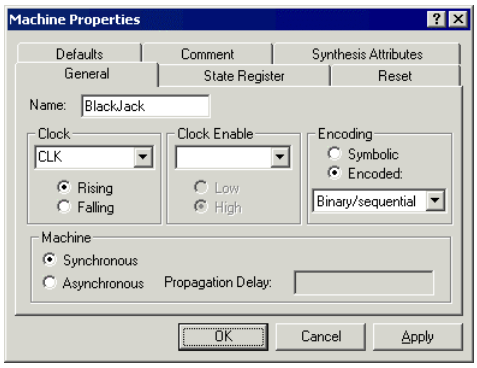
Selecting State Encoding

When state machines are compiled into logic, the current states are stored in a state 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 transitions at any given time. *One-Hot* encoding, however, consumes more flip-flops because each state requires a bit in a register. For that reason, it is not recommended for CPLDs that lack flip flop resources. The State Diagram Editor supports *One-Hot* and *Binary* encoding. Other types of encoding can be created by manual assigning codes to each state.

The binary encoding method will be used in this tutorial to simplify design analysis in the simulator.

1. Invoke the **Machine Properties** window. (To do so, click the blank state diagram area with the right mouse button).

2. Select the **Encoded** option and **Binary/sequential**; then click **OK**.



NOTE: The **Symbolic Encoding** option allows a synthesis tool to automatically select the preferred state encoding method.

Compiling the Design

To compile the design, choose **Compile** from the **Design** menu or click the **Compile** icon . Active-HDL will automatically generate a VHDL code corresponding to the state diagram.

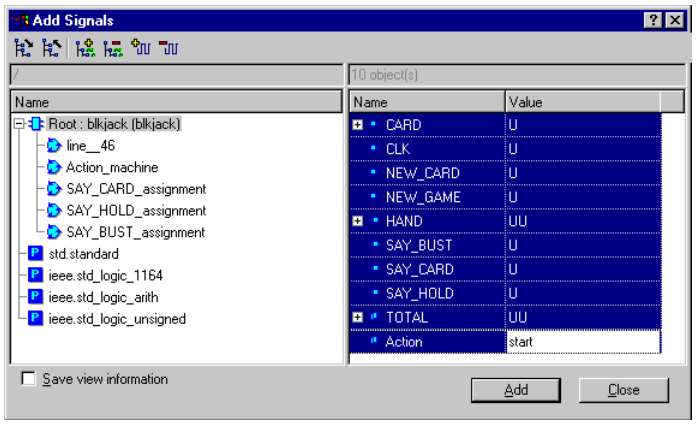
Using the **Top-level Entity** box on the **Structure** tab of the **Design Browser**, select the top-level entity **blkjack (blkjack)**.

Simulation

1. To initialize simulation, select **Initialize** from the **Simulation** menu.

2. Choose the **New | Waveform** option from the **File** menu or click the **New Waveform** icon to open a new waveform window.

3. Select the **Structure** tab of the **Design Browser** and click the plus sign next to the **Root**. The structure will expand and the root signals will appear at the bottom of the **Design Browser** window.

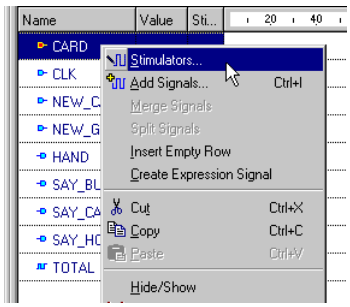


4. Select **Add Signals** from the **Waveform** menu The **Add Signals** dialog box will appear. Pressing the **Ctrl** key, select the following signals: *CARD, CLK, HAND, NEW_CARD, NEW_GAME, SAY_BUST, SAY_CARD, SAY_HOLD*, and *TOTAL*.

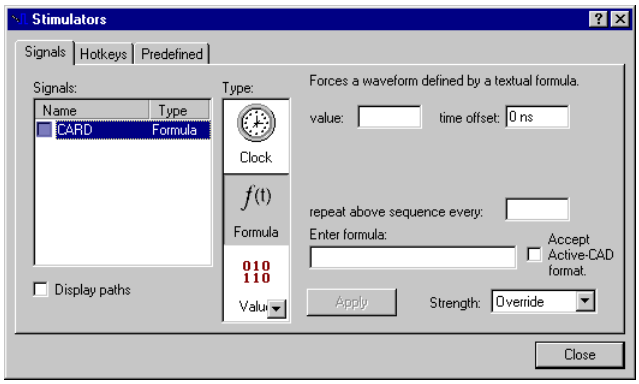
5. Click **Add** to add the signals to the **Standard Waveform Viewer** window.

6. To assign stimulators to the signals:

- Right click any signal name to open **Stimulators** dialog box and select the required signal in the **Standard Waveform Viewer** window.



- Using the **Signals** tab of the **Stimulators** dialog, select **Formula** as the stimulator type and **Override** in the **Strength** field and define the formula values.



- To add formula to other signal do not close the **Stimulators** window, click any signal names in the **Waveform Viewer** window to import them into the **Stimulators** window.

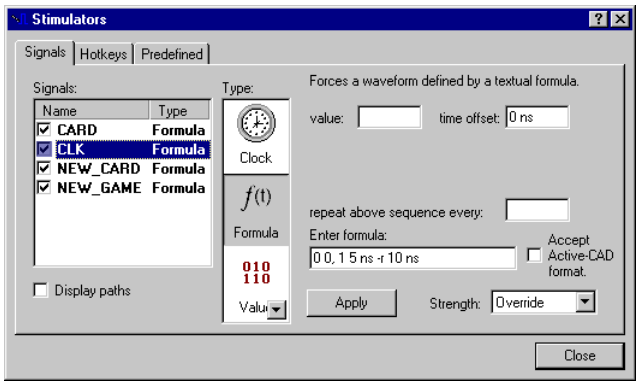
7. Using the procedure described above, assign the following stimulators to the signals:

- NEW_CARD 0 0, 1 80 ns -r 100 ns
- NEW_GAME 1 0, 0 20 ns -r 600 ns
- CARD 16#A 0, 16#A 100 ns, 16#3 170 ns, 16#B 300 ns, 16#4 400 ns
- CLK 0 0, 1 5 ns -r 10 ns

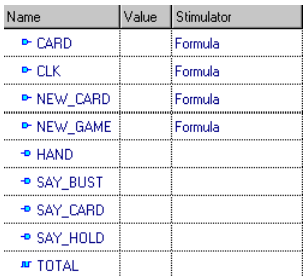
The above formulas are based on a very simple syntax:

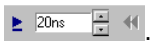
```
value [ time { , value time } [ -r period ] ]
```

[ ] Square brackets indicate an optional item
{ } Braces indicate an optional item which can be repeated



The signal names in the **Waveform** window should look as follows:

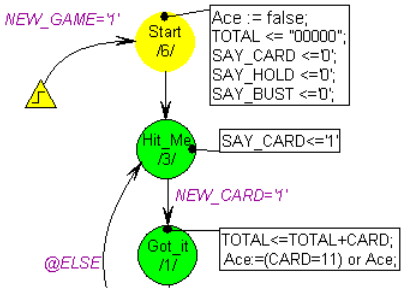


8. Set the simulation step to 20ns in the **Increase/Decrease Time** box located in the main toolbar 

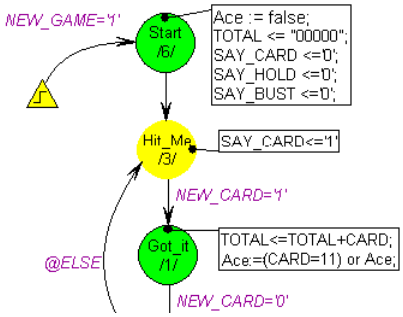
. The **State Machine Editor** displays the current active and inactive states with

different colors. While other inactive states are green, the active one changes its color from green to yellow.

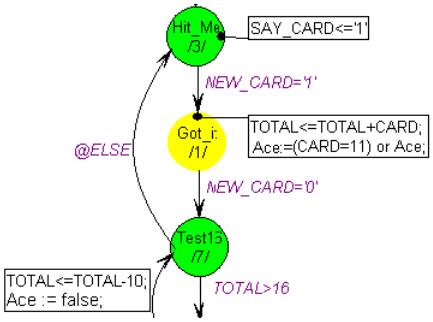
9. Perform several simulation steps using the **Trace Over** icon  from the **State Machine Editor** toolbar until you reach 5ns+1. This is where the game begins and all the *SAY_* signals are reset. At this point, the *Start* state is the active one and is drawn with a yellow color.



10. Click the **Run For** icon  once. Now, the control is passed to the *Hit_me* state. Watch the change of colors on the state diagram. The *SAY_CARD* signal has now the value of 1.



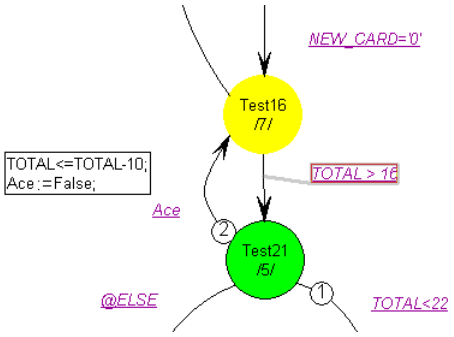
11. Perform the simulation clicking the **Run For** icon until you reach 85ns. You are now at the *Got_it* state. This means that you have been given a card.



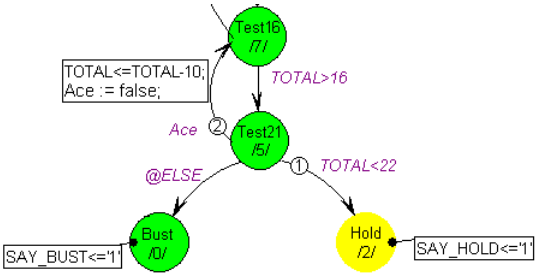
12. Click the **Run For** icon once. The machine will check whether the total sum of your cards is 16. As it is your first card, it will be not.

13. Perform the steps from the points 10 to 12 and observe how the state machine works. This is the live control flow of the Black Jack machine. In the **Design Browser** window, you can see how the signal values are changing while the control is passed from one state to another.

14. When you reach 420ns you will see that the control is passed to the *Test21* state and then because of the **Ace** condition is true the **Total** value is reduced by 10 to prevent it from exceeding the total of 21 points. The control is then passed to the *Test16* state.



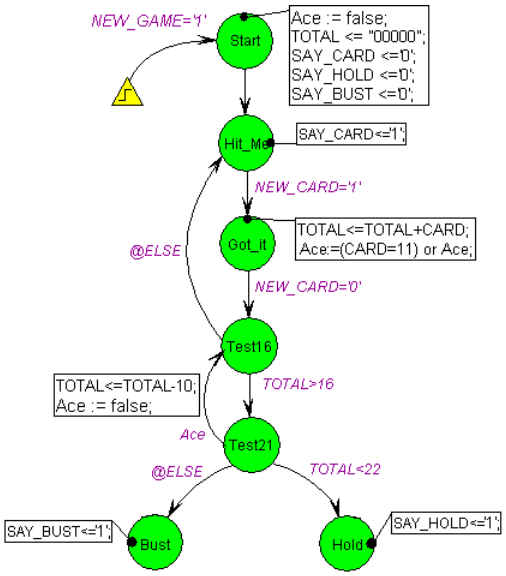
15. Click the run icon until you reach the **Hold** state. This means that you have won.



If you want to play more you can change the *CARD* signal values and observe the game results. Just remember that the values are specified with the hexadecimal radix.

Using Hierarchical States

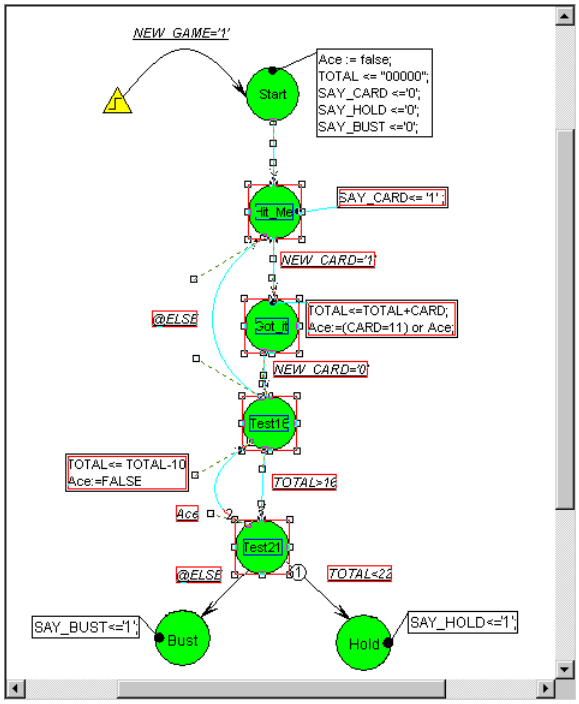
The *BlackJack* machine also can be implemented by drawing a state diagram employing the hierarchical state objects. We will modify an existing state diagram to present how to create a hierarchy in your FSM. The final state diagram shown in the **Testing the Final Score** section - point no. 8 will be used.



You can create a hierarchical state by selecting and converting the existing state symbols. To do so, select the states that you want to convert and choose the **Convert to Hierarchical State** option from the **FSM | Hierarchy** menu or the FSM toolbar. However, we recommend that you also get familiar with the below procedure describing in details (steps # 1-13) how to create a state machine hierarchy.

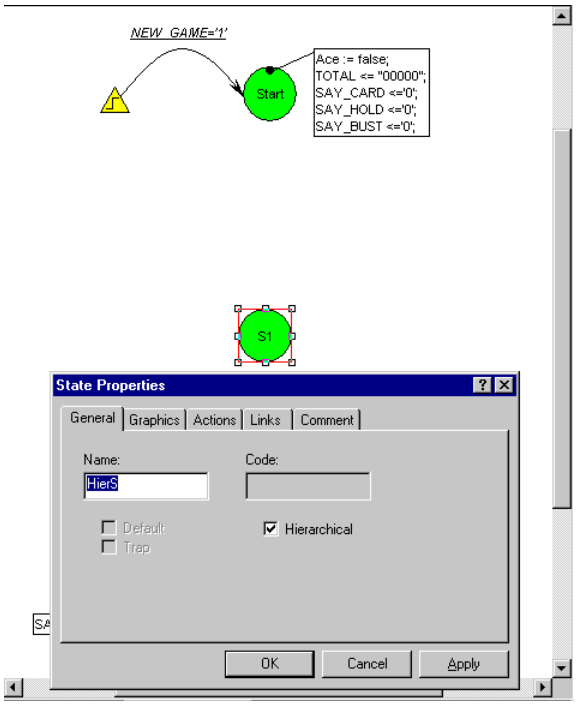
1. Select the following states and their transitions on the original state diagram:

- *Hit_Me*
- *Got_it*
- *Test16*
- *Test21*

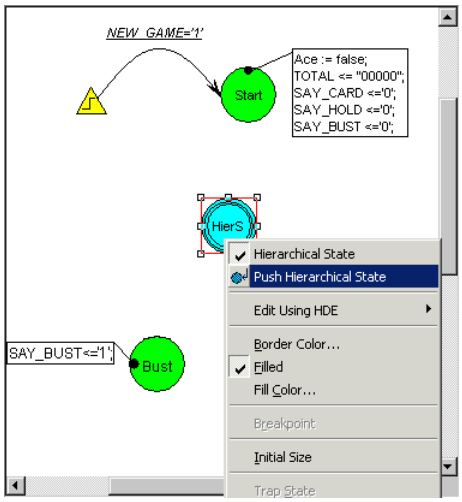


2. Cut the selection to the clipboard.

3. Place a new state object. Invoke the **State Properties** window from the pop-up menu, enter **HierS** in the **Name** field, and check the **Hierarchical** option. Next, close the dialog box by clicking **OK**.

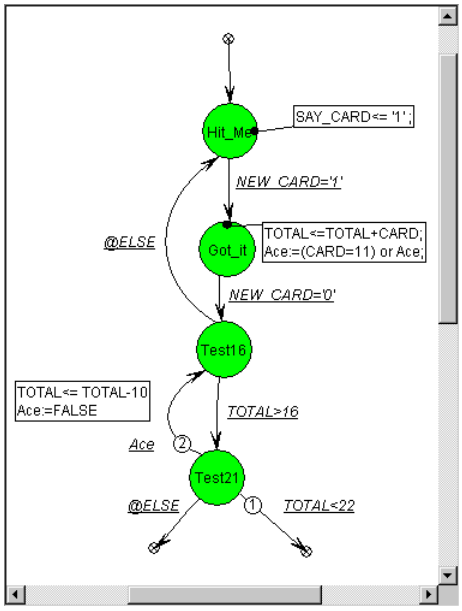


4. Right-click on the hierarchical state and choose the **Push Hierarchical State** option from the pop-up menu. You can also choose the **Push Hierarchical State** command from the **FSM | Hierarchy** menu or select the **Push Hierarchy** icon from the toolbar.



5. Select all objects in the child sheet and delete them.

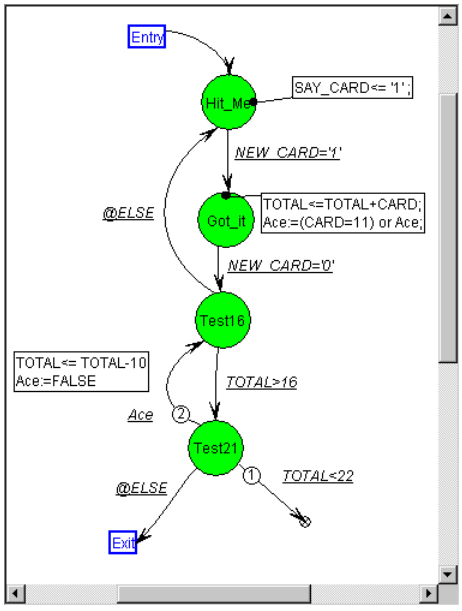
6. Paste the objects that have previously copied to the clipboard (in step #2).



7. Place the **Hierarchy Entry** object on the child (hierarchical) diagram and connect it to the *Hit_Me* state. To do so, choose the **FSM | Hierarchy Entry/Exit | Entry** menu command or click the **Hierarchy Entry** icon on the FSM toolbar (see figure below).

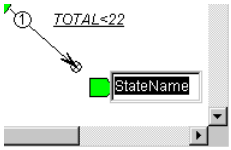
8. Place the **Hierarchy Exit** object on the child (hierarchical) diagram and connect it to *Test21* state (see figure below).

9. Drag the ends of transitions connecting the states with the **Entry** and **Exit** objects as it is shown on the figure below:

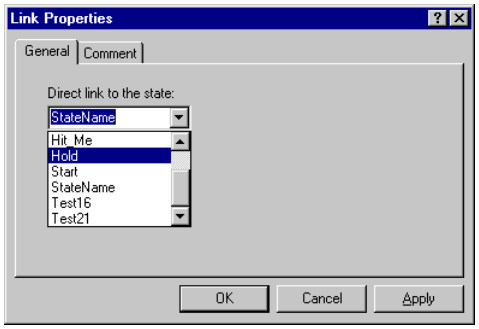


10. Place the **Link** object on the child sheet and connect it to the transition coming out from the *Test21* state. You can use the **FSM | Link** menu command or the **Link** toolbar icon.

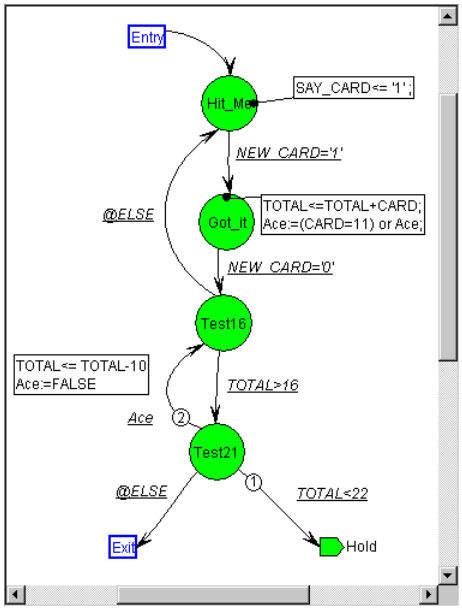
11. Rename the default link's name. You can either edit the Link name in-place or invoke the **Link Properties** window.



12. Enter *Hold* in the Link name field (as it shown in the figure above) or choose the target *Hold* state in the **Link Properties** window presented below.



13. Connect the transition to the **Link** object. Now, the child state diagram is ready-to-use and you can save the changes.

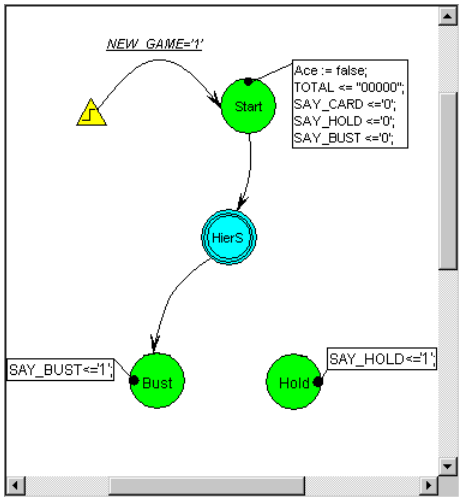


14. From the **FSM** or shortcut menu, choose the **Pop Hierarchy** option to get back to the top-level diagram sheet.

15. Place the unconditional transition between the following states:

- From the *Start* state to the *HierS* state
- From the *HierS* state to the *Bust* state

Your state diagram should look as it is presented below:



16. Save the changes and compile your hierarchical state diagram.

To compile your design, choose **Compile** from the **Design** menu or click the **Compile** icon  . Active-HDL will automatically generate the VHDL code corresponding to the hierarchical state diagram.

Enjoy your game and thank you for using Active-HDL !