

ACTIVE State Editor Training Guide

ALDEC Inc.
July 1996

Introduction

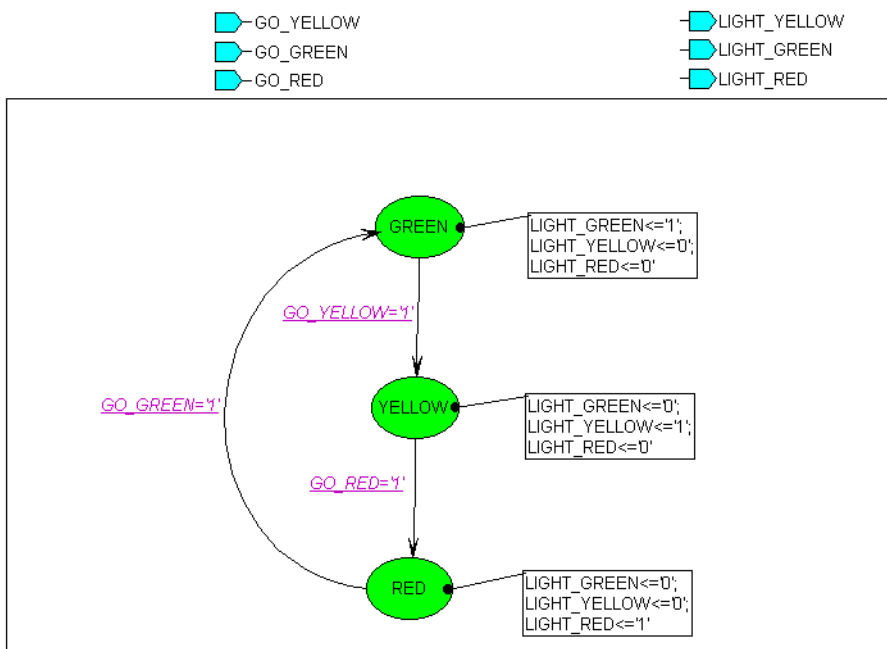
This tutorial is intended for the ACTIVE-CAD and Xilinx Foundation users. It presents the State Editor application used for graphical entry of ABEL and VHDL designs. This tutorial presents only the VHDL entry. For more detailed information and ABEL design issues please refer to the complete documentation of State Editor and advanced tutorials.

Comparison of State Machine Description Methods

Let's consider very simple state machine controlling traffic lights. The most obvious method of description is **verbal description**. Our state machine can be described as follows:

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

State diagram is a graphical method of state machine description. State diagram for lights controlling machine is shown below:



State machines can be also described using HDL languages such as VHDL, Verilog and ABEL. **VHDL description** of lights controlling machine is shown below:

```
entity lights is
  port (CLK: in STD_LOGIC;
        GO_GREEN: in STD_LOGIC;
        GO_RED: in STD_LOGIC;
        GO_YELLOW: in STD_LOGIC;
        LIGHT_GREEN: out STD_LOGIC;
        LIGHT_RED: out STD_LOGIC;
        LIGHT_YELLOW: out STD_LOGIC);
end;

architecture lights_arch of lights is

  type LIGHTS_type is (GREEN, RED, YELLOW);
  signal LIGHTS: LIGHTS_type;

begin

  process (CLK)
  begin

    if CLK'event and CLK = '1' then
      case LIGHTS is
        when GREEN =>
          if GO_YELLOW='1' then
            LIGHTS <= YELLOW;
          end if;
        when RED =>
          if GO_GREEN='1' then
            LIGHTS <= GREEN;
          end if;
        when YELLOW =>
          if GO_RED='1' then
            LIGHTS <= RED;
          end if;
        when others =>
          null;
        end case;
      end if;
    end process;

    LIGHT_GREEN <= '1' when (LIGHTS = GREEN) else
      '0' when (LIGHTS = RED) else
      '0' when (LIGHTS = YELLOW) else
      '0';

    LIGHT_YELLOW <= '0' when (LIGHTS = GREEN) else
      '0' when (LIGHTS = RED) else
      '1' when (LIGHTS = YELLOW) else
      '1';

    LIGHT_RED <= '0' when (LIGHTS = GREEN) else
      '1' when (LIGHTS = RED) else
      '0' when (LIGHTS = YELLOW) else
      '0';

  end lights_arch;
```

Main advantages and disadvantages of state machine description methods are listed in the table shown below.

Verbal Description	State Diagram	HDL Description
<ul style="list-style-type: none"> • compact • easy to read by humans • difficult to implement 	<ul style="list-style-type: none"> • very compact • very easy to read by humans • easy to implement • self-documenting 	<ul style="list-style-type: none"> • lengthy • hard to read by humans • easy to implement • requires documentation

The Purpose of This Tutorial

This tutorial will teach you how to use *Finite State Machine Editor* for entering state machine diagram and logic synthesis of the machine. We assume that you are familiar with other ACTIVE-CAD applications: *Project Manager* and *Simulator*.

Required Software

1. ACTIVE-CAD 2.2 or Xilinx Foundation v. 6.0.x
2. Metamor XVHDL synthesis
3. XACT software for implementation

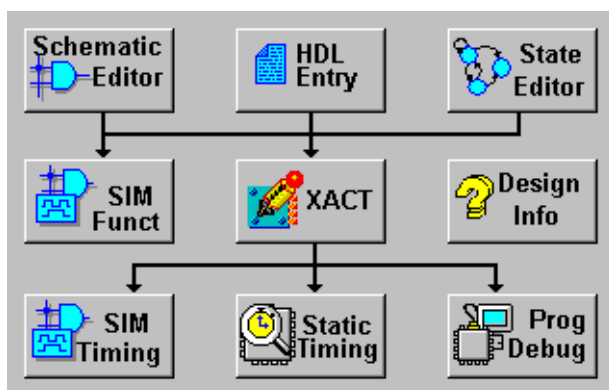
Enabling the State Editor in Xilinx Foundation Software

Xilinx Foundation Project Manager has State Editor program button disabled by default. To enable State Editor you should add a setting to *susie.ini* file.

Select **File | Configuration** in *Project Manager* and click **View Ini File** button in *Configuration* window. Insert new section below [Paths] section:

```
[Flow]
FSM_X6=On
```

Save *susie.ini* file and quit text editor. Restart *Project Manager* - *State Editor* will be enabled as shown below.



Creating a State Diagram

The design which will be created in this tutorial is a machine which plays the Blackjack game. The Xilinx 4003 device is used to target the design. The following is the description of how the design should behave:

The purpose of the game is to request a desired number of cards by showing the SAY_CARD output signal 1 and not exceed the total sum of cards more than 21. Each card can be a number between 2 and 11, where 11 is called ACE and can be counted as 1 if desired. The arrival of the card is flagged on the signal NEW_CARD changing from 0 to 1.

The machine should request additional cards when the total sum of cards is less than 17 and should flag a SAY_BUST signal when it exceeds 21. In case the total score is between 17 and 21 the machine should flag a 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 that state until NEW_GAME signal is flagged on the input.

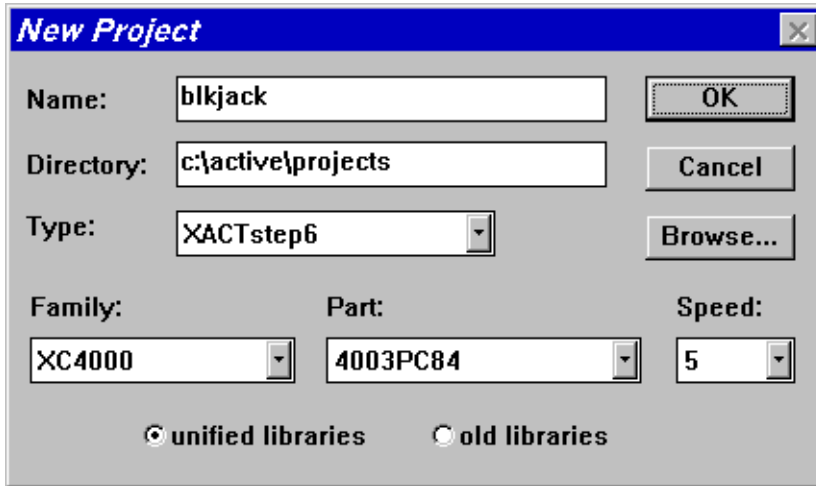
The NEW_GAME signal resets the machine.

Implementation


The machine will be implemented by drawing a state diagram and then generating VHDL code, synthesizing into Xilinx netlist and then implementing using XACT software.

1. Creating New BLKJACK Design

- Start *Foundation Project Manager*
- Create new project - select **File | New Project** and enter *BLKJACK* as the project name in *New Project* window



2. Opening State Editor

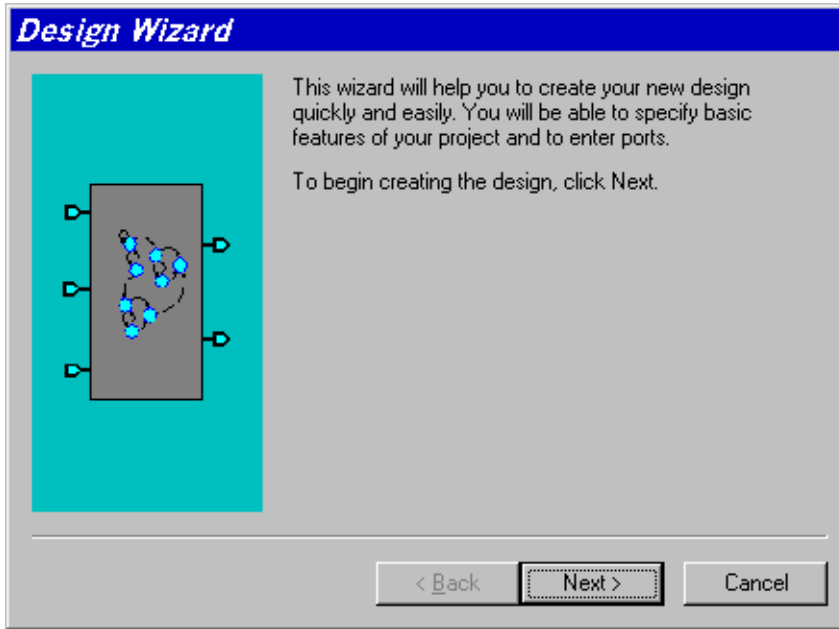
- Click  button in design flow section of *Project Manager*
- Click **Use FSM Design Wizard** button in *State Editor Welcome Window*



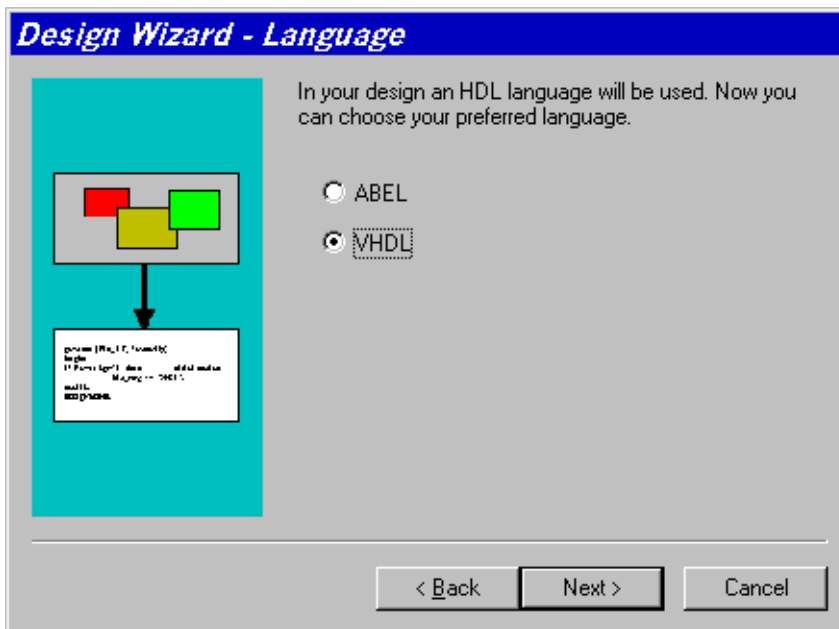
3. Using FSM Design Wizard

FSM Design Wizard allows creating state diagram file, selecting hardware description language to which diagram will be translated, definition of ports, etc. To create state diagram pattern follow instructions given below:

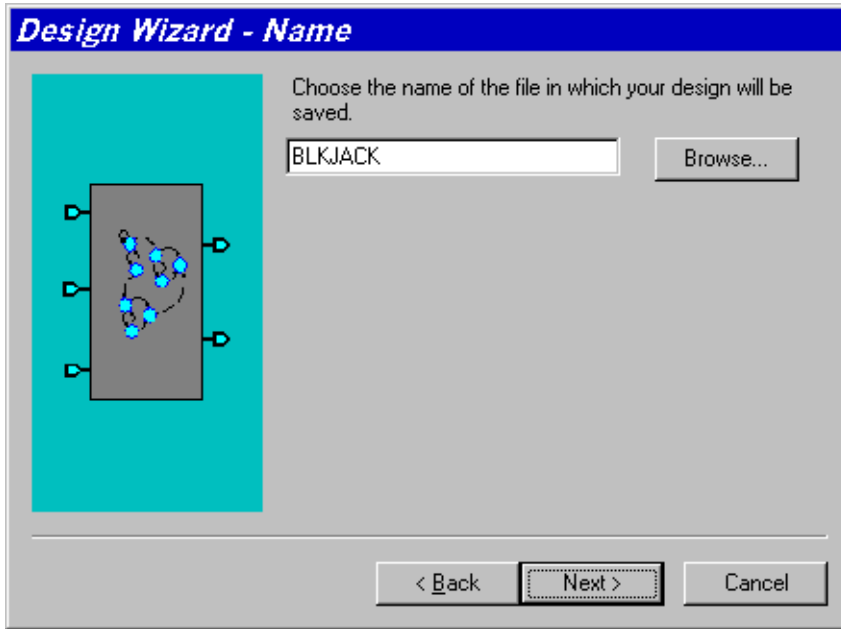
- Click **Next >** in Design Wizard welcome window



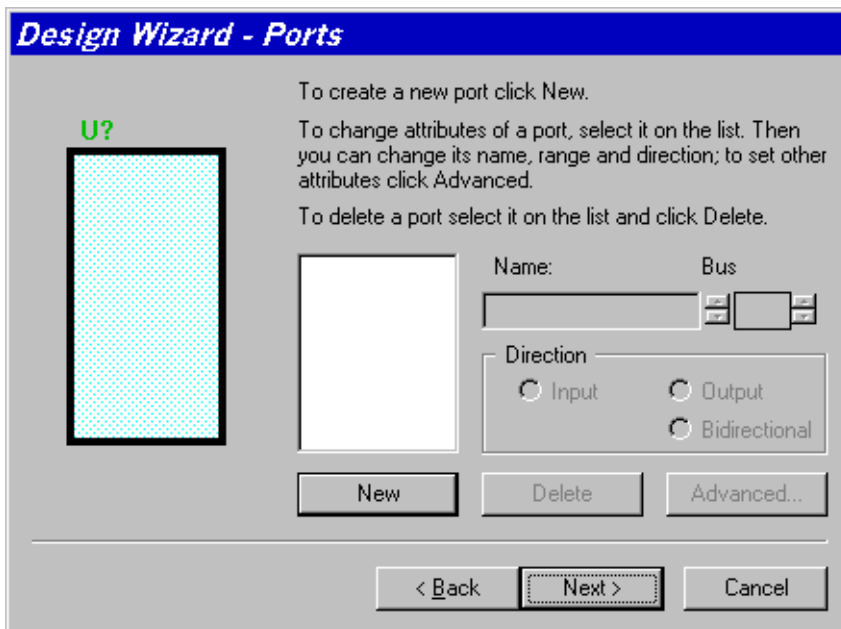
- Select **VHDL** language in *Design Wizard - Language* window and click **Next >** button



- Type *BLKJACK* as the file name in *Design Wizard - Name* window and click **Next >** button. Note that the default file extension of state diagrams is ASF. In this case the file will be called BLKJACK.ASF.



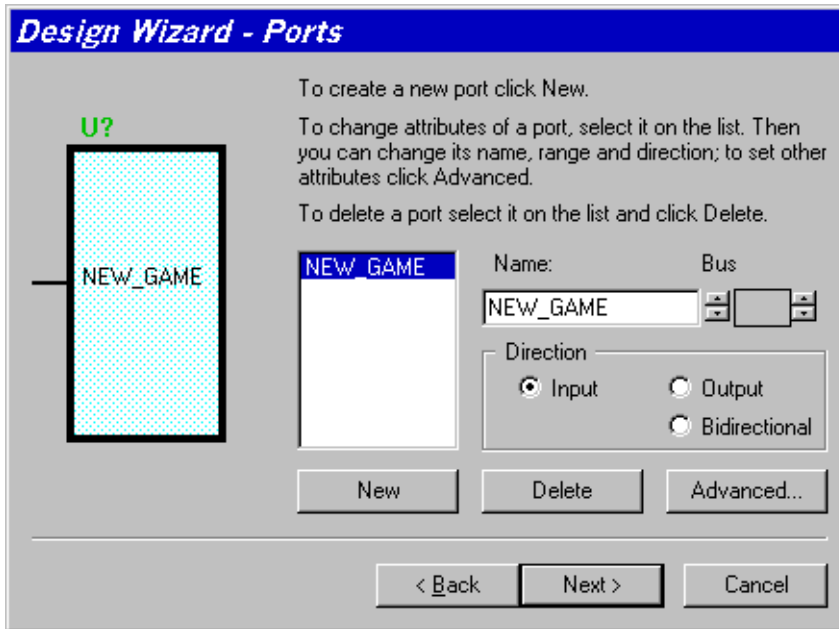
- *Design Wizard - Ports* window is displayed. It is used to add input and output signals used in your state machine.



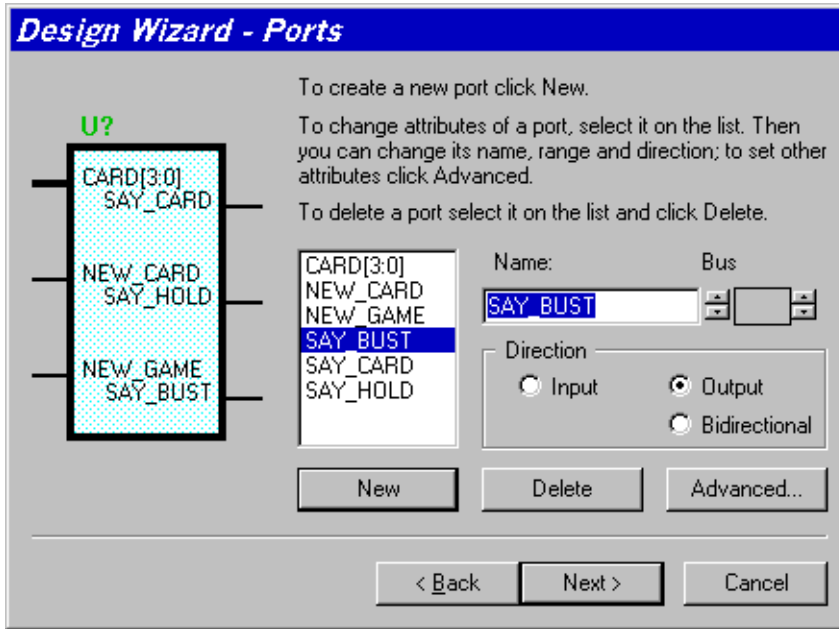
To add new port in that window:

- click **New**
- type port name in **Name:** box
- choose port direction by selecting one of the options in **Direction** box.

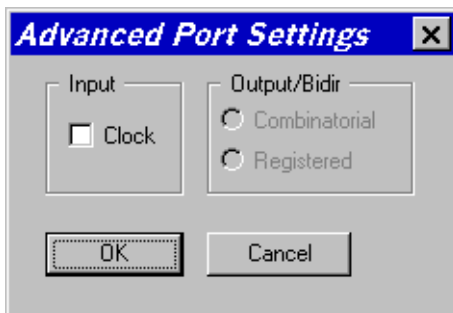
Port names are displayed in the box above **New** button and in the symbol preview area.



- Using procedure described above define:
 - ◇ input ports:
 - NEW_GAME
 - NEW_CARD
 - CARD[3:0]
 - ◇ output ports:
 - SAY_CARD
 - SAY_HOLD
 - SAY_BUST
- Click **Next >** when all ports listed above are defined. See picture below for reference.



NOTE: The **Advanced** button displays *Advanced Port Settings* window below and allows to specify the type of port.



You can define a selected signal as clock, and for the output port you can specify if they are combinatorial or registered. **Registered** ports hold the value set in an additional register and the **combinatorial** outputs change any time the input conditions change.

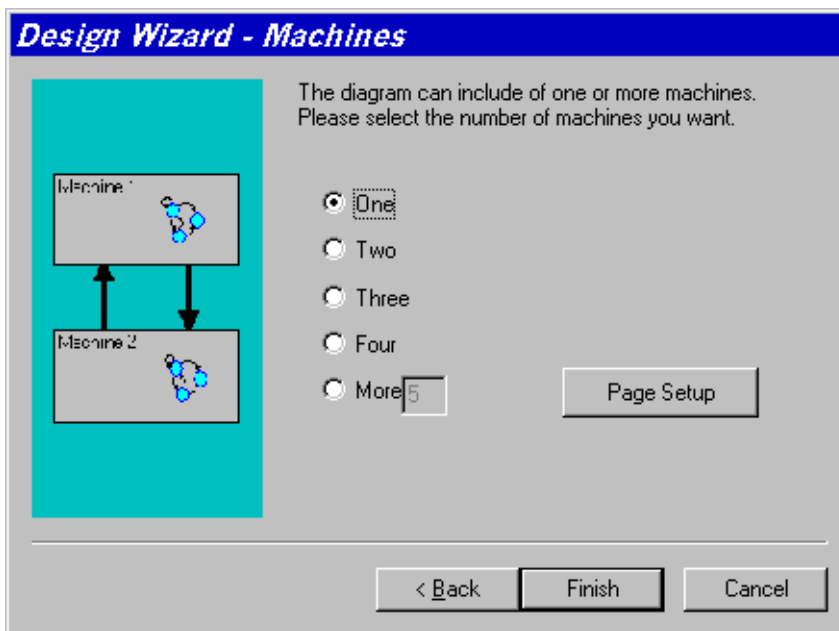
NOTE: State Editor support only synchronous state machines, which means that all transitions from one state to another are performed only at the clock transitions. Since no clock was defined the Wizard will prompt you for creating a clock signal.

- Design Wizard suggests adding CLK port -

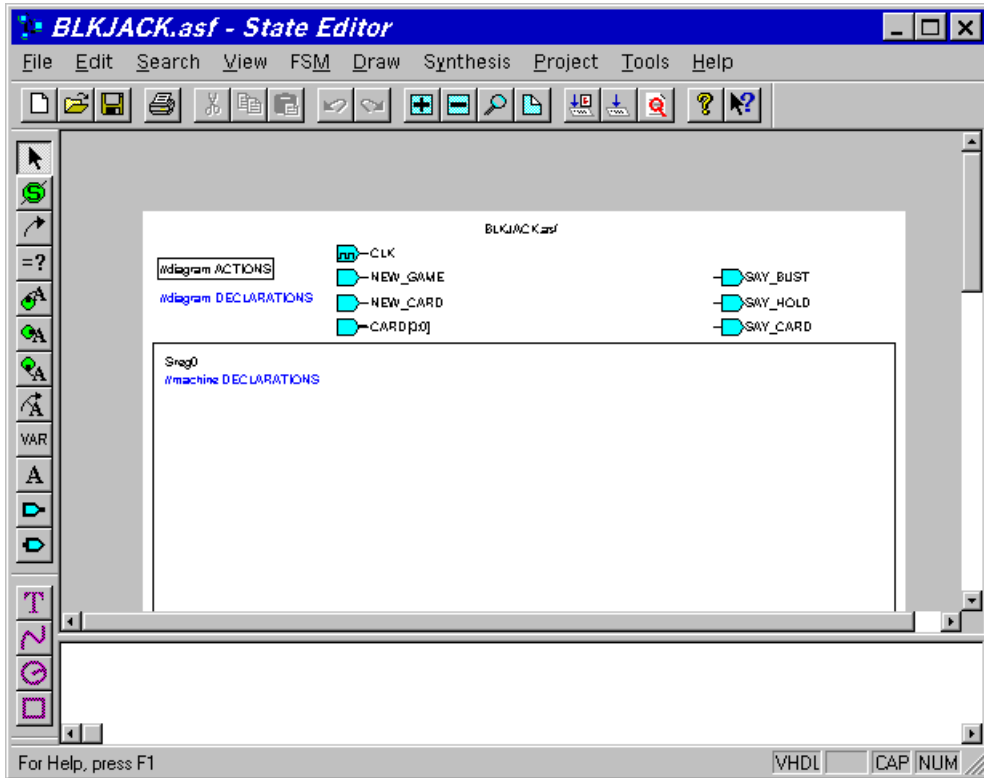


click **Yes** to add that port

- In *Design Wizard - Machines* window select **One** machine on the diagram and click **Finish**.
- **NOTE:** You can use State Editor to design concurrent state machines which are translated into separate processes in VHDL.




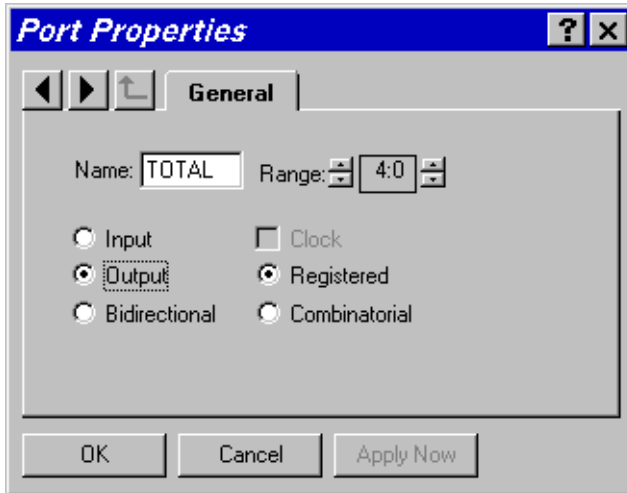
- The new state diagram created by the Design Wizard is shown below.



4. Adding Extra Ports

In case you did not add some ports in the FSM Wizard you can do it afterwards using the Add port button. In this case the output port showing the total score of the game was not added.

- Click  (Output Port) button, then click above the *SAY_BUST* port symbol to add new output port.
- Click right mouse button on the port symbol you have just placed on the diagram and select **Properties** from the local menu
- In *Port Properties* window type **TOTAL** in **Name:** box, select 4:0 in **Range:** box, select **Output** and **Registered** options
- Click **OK**



NOTE: The TOTAL port has to have range from 0 to 31 which requires 5 bits of data [4:0]

New output ports definition section should look the following:

```

┌─┴─┬─▶ TOTAL[4:0]
┌─┴─┬─▶ SAY_BUST
┌─┴─┬─▶ SAY_HOLD
┌─┴─┬─▶ SAY_CARD
  
```

5. Defining Additional Variables

In our machine there will be a need to know if one of the cards is an ACE. This will be helpful to count ACE as 1 instead of 11 in case the score exceeds 21. The following procedure describes how to define a new *Ace* variable.





- Double click *//machine DECLARATIONS* text below *Sreg0* text


BLKJACK.asf

```

//diagram ACTIONS
//diagram DECLARATIONS
Sreg0
//machine DECLARATIONS

```

 CLK
 NEW_GAME
 NEW_CARD
 CARD[3:0]

NOTE: If, for some reason, *//machine DECLARATION* text is missing, you can still add machine declarations by clicking  (Declarations) button

- In the edit box

```

Sreg0
//machine DECLARATIONS

```

type

variable Ace: boolean;

below *//machine DECLARATION* text





- Click right mouse button outside edit box.
Correct declarations should look the following:

BLKJACK.asf

```


//diagram ACTIONS
//diagram DECLARATIONS
Sreg0
//machine DECLARATIONS
variable Ace: boolean;

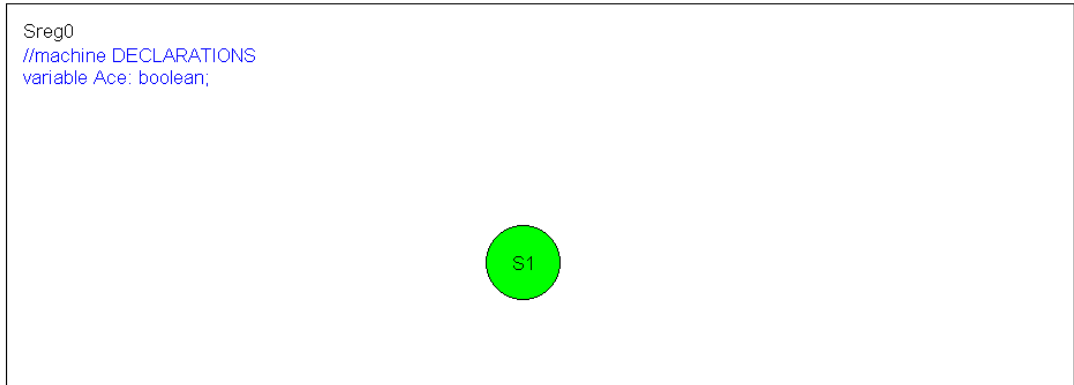
```

 CLK
 NEW_GAME
 NEW_CARD
 CARD[3:0]

6. Adding a Reset State

The first state that will be added will be used to initialize the machine when the new game is started. It will be used to set initial values of all outputs.

- Click  button and place state symbol in the middle of the sheet, below machine declarations (use left mouse button to place state, right mouse button to cancel state mode)



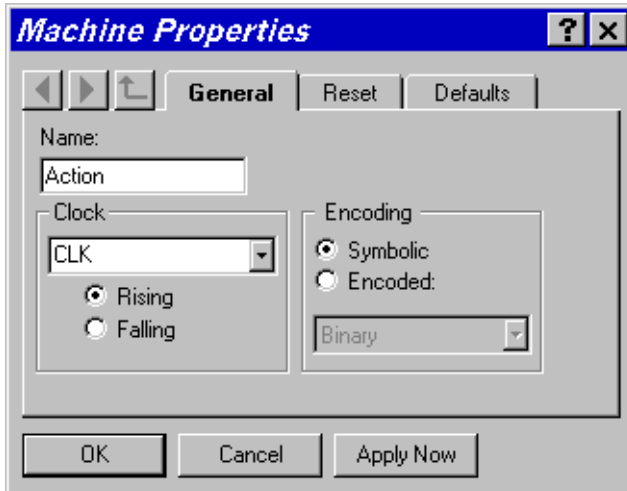
- Double click *S1* state name to open edit box



and replace *S1* by *Start*, then click outside the edit box. The state with new name should look like this:



- Select **FSM | Machines | Sreg0** from the menu to display *Machine Properties* window. This window is used to define global settings of the selected state machine.

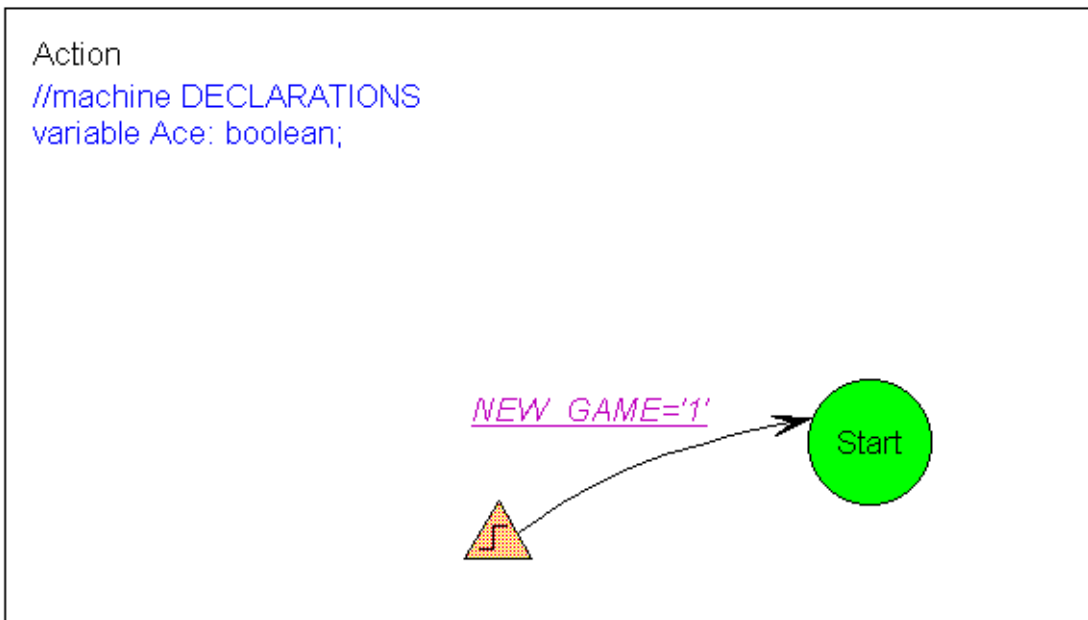


- Replace *Sreg0* with *Action* in the **Name:** box. This sets the name of the state machine signal used to store the current state in VHDL.



- Click on 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 defines that the *NEW_GAME* will be used as machine reset and when it is activated the machine will enter the *Start* state.


Click **OK** button - state diagram should look as below:



NOTE: The reset transition defines a reset condition for documentation purposes. The reset symbol is not a state and it indicates that this transition will be executed regardless of the current state if the condition is met.

7. Adding initialization actions

Once the reset condition is met the machine will enter the Start state. To define the actions that should be performed when this happens we will create an entry action. The entry action is always attached to the top of the state symbol and indicated that the code specified is executed once when the machine enters this state.

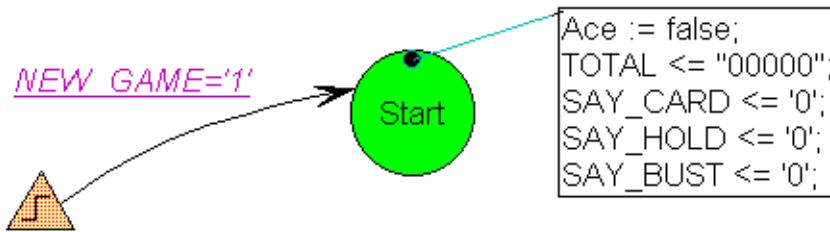
- Click  (Entry Action) button, then position 'dot end' of the mouse pointer over *Start* state and click left mouse button. Type:

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

in the edit box and click right mouse button outside edit box.

Note that you have to use the VHDL syntax and add semicolons after each line. Also note that the assignment operator <= can be used with ports and signals, and the := operator is used with variables.


Defined entry actions are shown below:

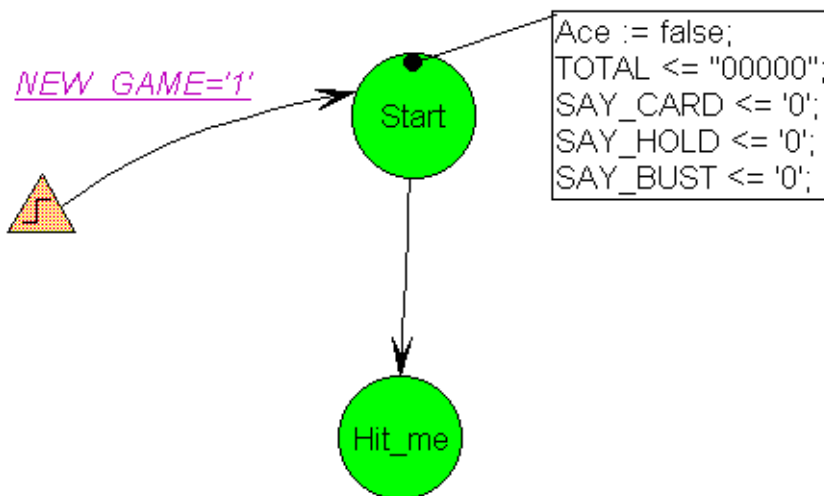
**NOTE:**

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 where the value was not explicitly indicated. In this example SAY_BUST will be '0' in all states unless you add an action SAY_BUST <= '1'.

8. Adding a state to request a card


Once the reset is performed that machine should start with requesting a card. This will be the *Hit_me* state.

- Add *Hit_me* state below *Start* state (use procedure described above for *Start* state)
- Click  (Transition) button, click *Start* state, click *Hit_me* state, then click right mouse button.

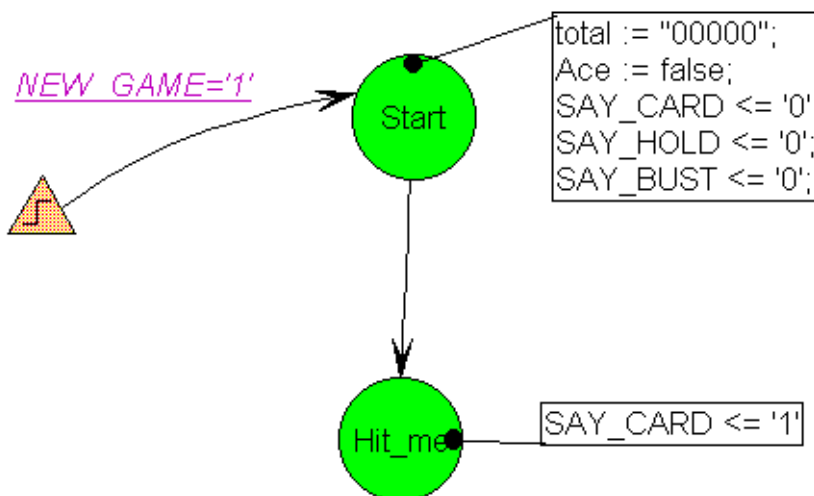


NOTE: Unconditional transition from *Start* state to *Hit_me* state has been

defined. This transition is 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'. Note that this signal should remain active all the time while in this state. For that reason we have to use state action and not entry action. State actions are executed on each clock cycle as long as the machine remains in this state.
- Click  (State Action) button, then position 'dot end' of the mouse pointer over *Hit_me* state and click left mouse button. Type
`SAY_CARD <= '1'`

in the edit box and click right mouse button outside edit box. State diagram after this operation is shown below:



9. HDL Code Generation

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

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

- Press **HDL Code Generation** from the Synthesis menu to generate HDL code; answer **Yes** when asked if you want to view generated code.

It is essential that you can relate the code that is being generated to the elements on the drawing. State editor uses standard templates to create

the VHDL design and understanding the structure of the VHDL is very helpful when debugging the design when you use 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 IEEE, METAMOR and SYNOPSYS libraries.

```
library IEEE;
use IEEE.std_logic_1164.all;

-- SYNOPSYS library declaration
library SYNOPSYS;
use SYNOPSYS.std_logic_arith.all;
use SYNOPSYS.std_logic_unsigned.all;

library METAMOR;
use METAMOR.ATTRIBUTES.all;
```

Entity declaration section lists all ports defined on 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;
        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 this type for every machine on your diagram.
- ◇ objects and actions common to all machines.

```
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).

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

Reset definition section

```
begin

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

Machine action description section

```
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

```
-- signal assignment statements for combinatorial outputs
SAY_CARD <= '1' when (Action = Hit_me) else
    '0';

SAY_HOLD <= '0';

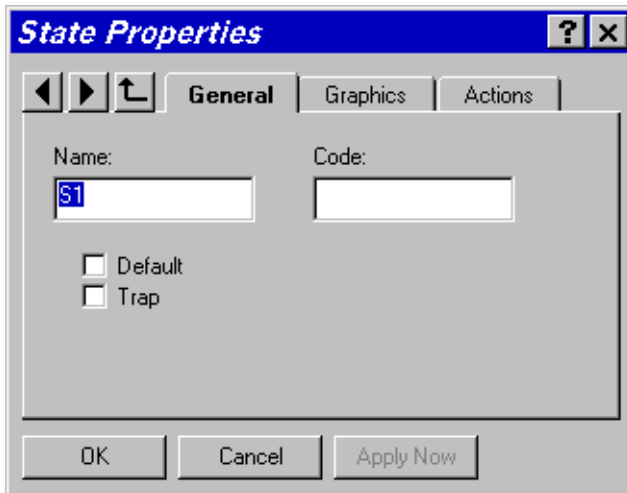
SAY_BUST <= '0';


end blkjack_arch;
```

10. Receiving and Handling of the New Card

Once the 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 next portion which performs this action.

- Add new state below *Hit_me* state
- Rename this new state to *Got_it* using *State Properties*:
 - ◊ click right mouse button inside state bubble, but not the state name
 - ◊ select **Properties** in the local menu
 - ◊ type *Got_it* in **Name:** box



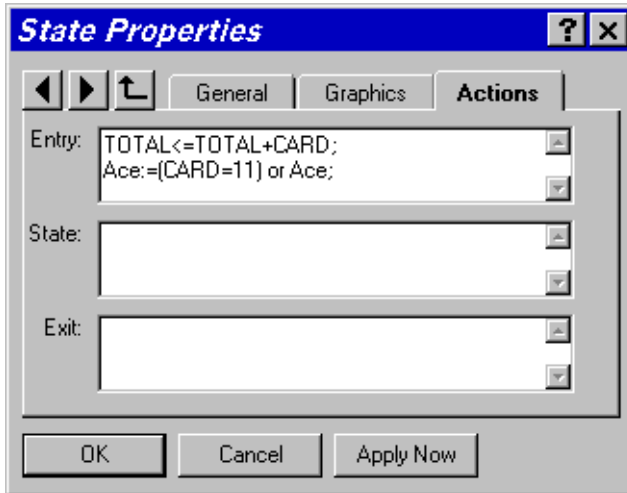
- ◇ click **OK**
- Add transition from *Hit_me* to *Got_it* state
- Add condition to the recently drawn transition:
 - ◇ click  (Condition) button
 - ◇ click the transition line
 - ◇ type `NEW_CARD='1'` in the edit box
 - ◇ click right mouse button outside edit box

NOTES:

- ⇒ state machine remains in *Hit_me* state until `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 default value '0' was defined in the reset state.
- ⇒ all actions are performed on rising edge of the `CLK` signal. This is also defined in Machine properties and can be changed if desired.
- Once the card was received, the total score needs to be updated. Also the card will be checked if it was 11 and if so the ACE flag will be set.
- Since these action need to be executed only once when entering the state an entry action will be used.
- To assign entry actions to the state, you can also use *Actions* card in *State Properties* window:
 - ◇ click right mouse button inside state bubble
 - ◇ select **Properties** in the local menu
 - ◇ click **Actions** tab
 - ◇ type

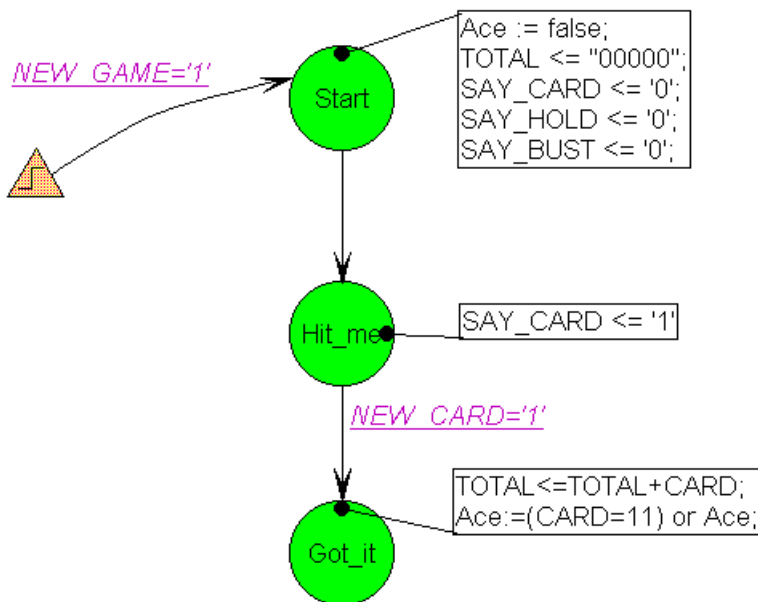

```
TOTAL<=TOTAL+CARD;
Ace:=(CARD=11) or Ace;
```

 in **Entry** box

**NOTE:**

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

State diagram should now look the following:



11. Analyzing the Total Score

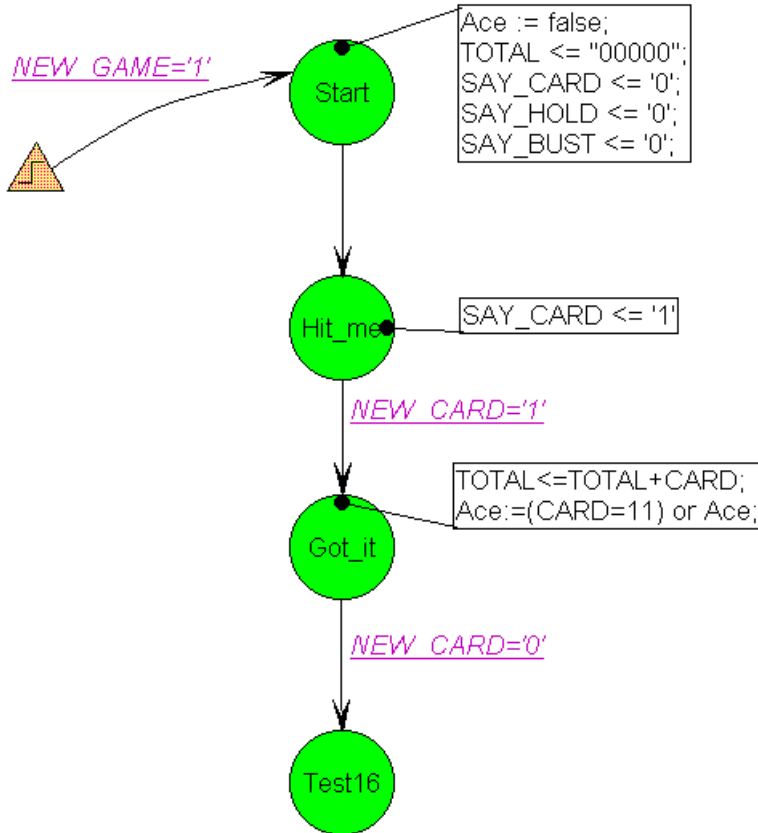
Once the total score is updated the machine has to wait for the SAY_CARD signal back to '0'. Then the total score will be tested. If it is less than 17 then machine should request a new card and go back to the Hit_me state.

- Add *Test16* state below *Got_it* state
- Add transition from *Got_it* state to *Test16* state
- Add `NEW_CARD='0'` condition to the recently defined transition

NOTE:

This prevents state machine from using the same card more than once.

After completing this step, state diagram looks like this

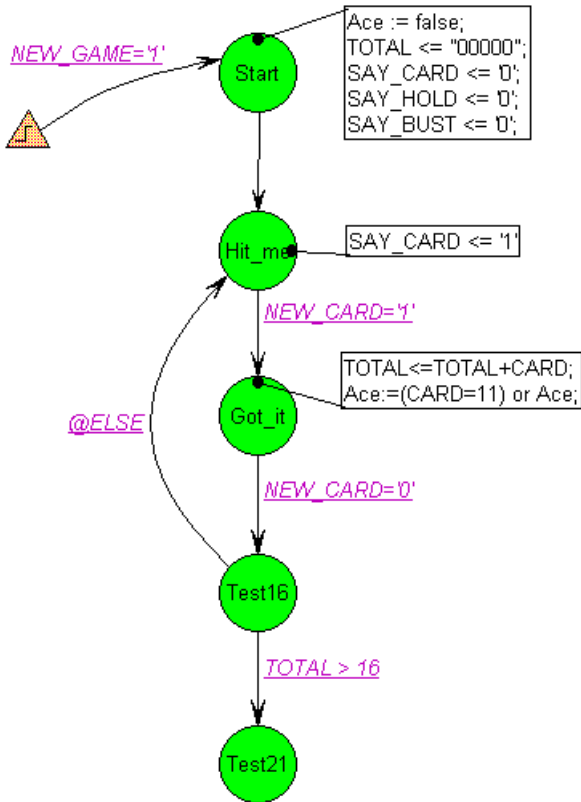


- Add *Test21* state below *Test16* state
- Add transition from *Test16* state to *Test21* state with `TOTAL > 16` condition
- Add transition from *Test16* state to *Hit_me* state with `@ELSE` condition

NOTE:

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

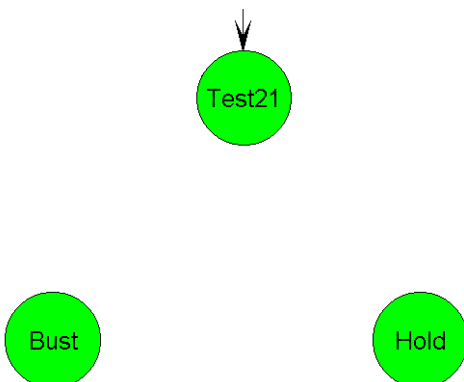
State diagram with 5 states should look the following:



12. Testing the Final Score

Once the machine gets to the Test 21 state the total score is 17 or more. Now the machine has to test if it did not exceed 21. If it did not it will flag the SAY_HOLD signal if it has more than 21 the SAY_BUST signal should be flagged and the machine has lost the game. Before flagging the SAY_BUST signal the machine can check if one of the previous cards was an ACE and then it may decrement the total value by 10 to recover from the bust situation.


- Add two final states: *Bust* and *Hold*.



- Add transition from *Test21* state to *Bust* state and from *Test21* state to *Hold* state
- Add $TOTAL < 22$ condition to the *Test21* \Rightarrow *Hold* transition
- Add $SAY_HOLD \leq '1'$ state action to *Hold* state
- Add transition from *Test21* state to *Test16* state with *Ace* condition

NOTE:

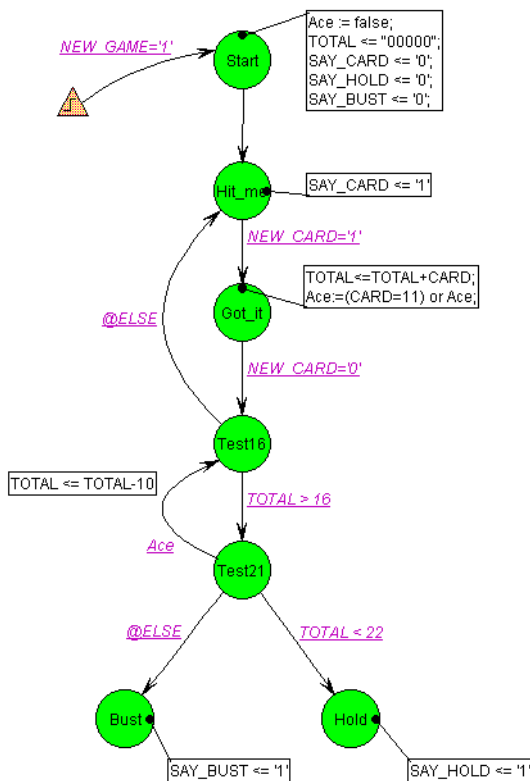
Since *Ace* is Boolean variable, no relational operators are required in the condition text

- Add $TOTAL \leq TOTAL-10$ action to the recently defined transition (use  button, click transition line and type action text)

NOTE:

Transition actions allow avoiding redundant states. If you did not assign this action to a transition than another state would be created to execute $TOTAL \leq TOTAL-10$ action.

- Add $@ELSE$ condition to the *Test21* \Rightarrow *Bust* transition
 - Add $SAY_BUST \leq '1'$ state action to *Bust* state
- State diagram with all states, transitions, conditions, and actions should look like this



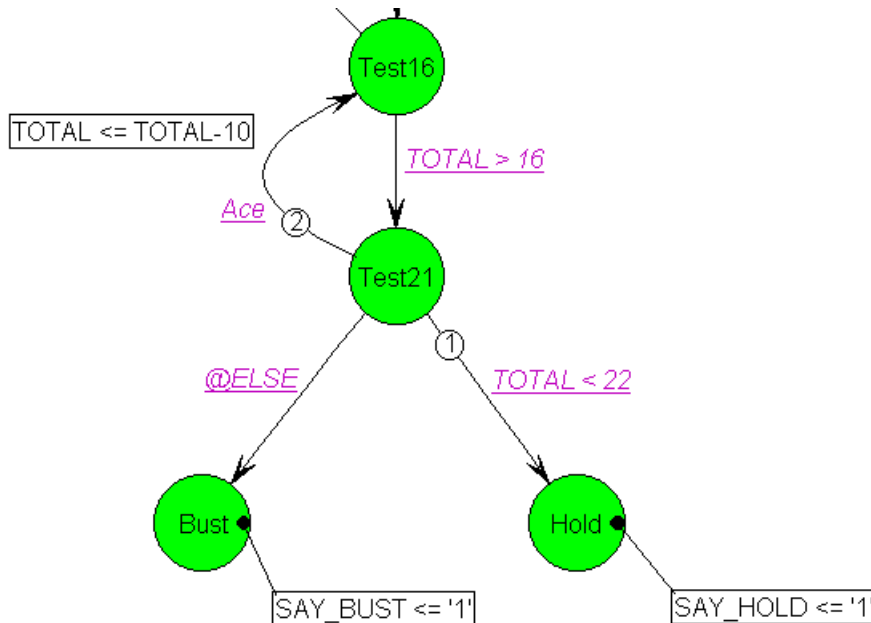
13. Assigning Condition Priorities

- NOTE that in *Test21* state both $TOTAL < 22$ and *Ace* conditions can be met at the same time. In that case behavior of the machine depends on condition

evaluation sequence. To avoid confusion, you should assign different priorities to transitions. Because our machine should first check if TOTAL is less than 22, assign priority **1** to *Test21* ⇒ *Hold* transition and priority **2** to *Test21* ⇒ *Test16* transition.

To change transition priority:

- ◇ click transition line
- ◇ click right mouse button
- ◇ select **Priority** from the local menu
- ◇ select required priority level



NOTE:

Transitions without displayed priority are executed last

14. Selecting State Encoding

When state machines are compiled into logic the current state is stored in a register (series of flip-flops). The values each state is assigned in the register may have an effect in the reliable behavior of the state machine. Incorrect selection of state encoding can create hazard conditions where a glitch in the combinatorial logic may cause the state machine to enter a wrong state. The recommended encoding type for all FPGAs is One-Hot which assigns values to each state so that only one bit of a register is active at any 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.

In this tutorial a binary encoding will be used for simplicity of analysis in the simulator.

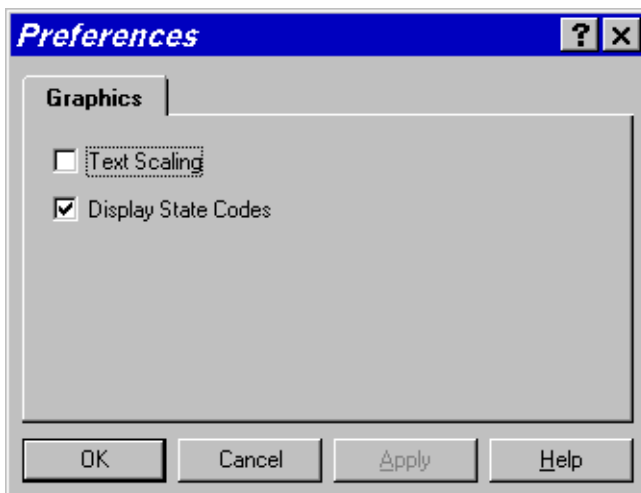
- Select **FSM | Machines | Action** and select binary encoding in *Machine Properties* window



NOTE:


Symbolic encoding option allows synthesis tool to select state encoding method. Metamor XVHDL defaults to binary encoding if the codes are not assigned any values.

- Select **Tools | Preferences** and check **Display State Codes** option

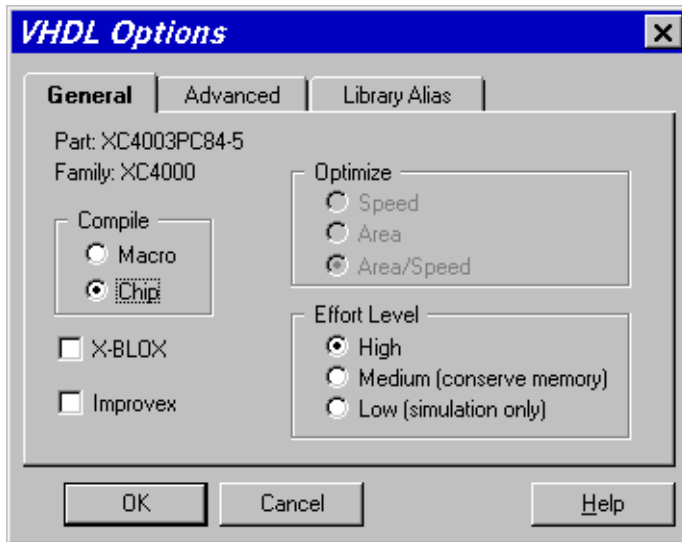


- Save the diagram

15. Compiling the Design

- Save the diagram
- Select **Project | Add to project** in *State Editor* Menu to add *BLKJACK.ASF* diagram as top-level document in your project. This will add the top level document  **BLKJACK.ASF** in *Project Manager* hierarchy navigator.

- Select **Synthesis | Options**



and deselect **X-BLOX** and **Improvex** options to speed up the synthesis process.

- Select **Synthesis | Synthesize** option (or click  button)

16. Locating and Correcting Errors

- Note that [Signal 'TOTAL' is not readable as it has mode OUT.](#) error is reported every time *TOTAL* appears on the right side of signal assignment. This is because the output ports cannot be used for reading. Instead the bidirectional port should be used.
- To correct this error, click right mouse button on *TOTAL* port symbol, select **Properties** and change port type to **Bidirectional**
- Restart synthesis - this time it should complete without errors
- Note that state codes appeared inside state bubbles
- Select **Synthesis | View Report** and check how many flip-flops were used for synthesis.

NOTE: If **Improvex** is used during synthesis, synthesis report also includes mapping information.

Synthesis report is very important, because it shows how much resources are used by the design. Making some changes on the diagram may drastically increase or decrease the number of flip-flops and CLBs.

17. Functional Simulation

- Minimize *State Editor*
- Click  button in *Project Manager* design flow section

- Select **Signal | Add Signal** and double-click on CLK, NEW_GAME, NEW_CARD, (CARD0,CARD3), (TOTAL0,TOTAL4), SAY_BUST, SAY_CARD, SAY_HOLD, and (ACTION2,ACTION0) signals to select them for observation in Waveform Viewer

NOTE: ACTION bus represents the state variable of Action machine. It enables monitoring of the current machine state.

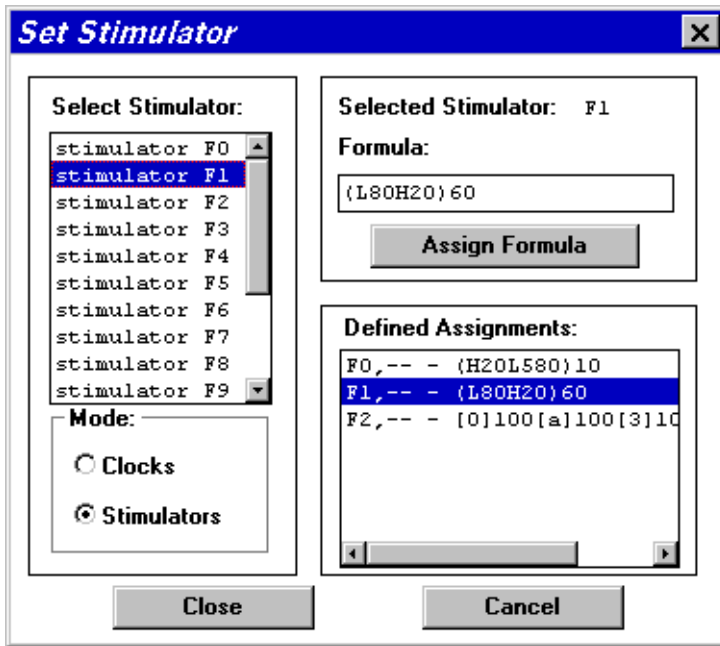
- Selected signals should look like on the picture below

i	CLK	
i	NEW_GAME	
i	NEW_CARD	
i	CARD0	*
i	CARD1	+
i	CARD2	+
i	CARD3	+
o	TOTAL0	*
o	TOTAL1	+
o	TOTAL2	+
o	TOTAL3	+
o	TOTAL4	+
o	SAY_BUST	
o	SAY_CARD	
o	SAY_HOLD	
l	ACTION2	+
l	ACTION1	+
l	ACTION0	*

NOTE: To ensure correct display of bus values, check if '*' (asterisk) is displayed by the bus element with index 0. In asterisk is not displayed there, select all bus members by clicking their names while holding **Shift** key and select **Signal | Bus | Direction**

- Select **Utilities | View | Hex Buses**
- Select **Stimulator | Add Stimulators** and assign **B0** stimulator to CLK input

- Click **Formula** button in *Stimulator Selection* window and define F0, F1, and F2 stimulators:



- ◇ Assign `(H20L580)10` formula to **F0** stimulator
 - ◇ Assign `(L80H20)10` formula to **F1** stimulator
 - ◇ Assign `[0]100[a]100[3]100[8]100` formula to **F2** stimulator
 - ◇ Click **Close**
 - Attach **F0** stimulator to NEW_GAME input, **F1** stimulator to NEW_CARD input, and **F2** stimulator to CARD input bus
 - While holding **Shift** key select CARD and TOTAL buses
 - Select **Signal | Bus | Display Decimal**
- Signal names in Waveform Viewer should look like on the picture below

i	CLK	B0
i	NEW_GAME	F0
i	NEW_CARD	F1
B	CARD0 (dec)#4	F2
B	TOTAL0 (dec)#5	
o	SAY_BUST	
o	SAY_CARD	
o	SAY_HOLD	
B	ACTION2 (hex)#3	

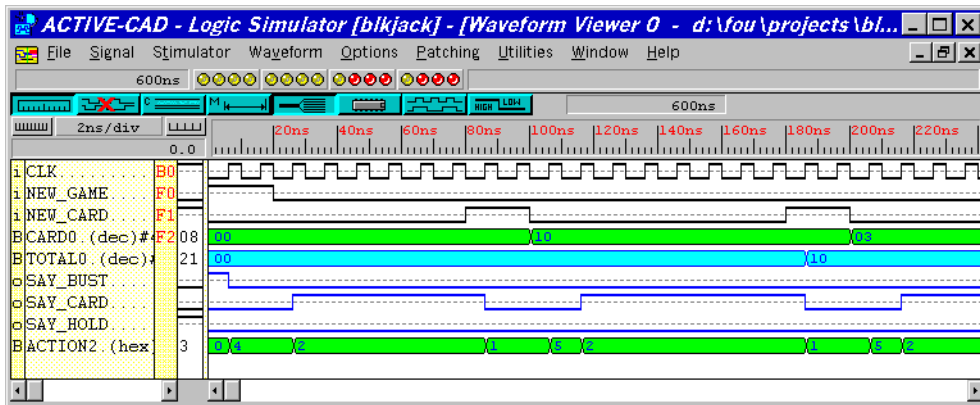
- Simulate using **Step** or **Long** button until machine signals SAY_HOLD or SAY_BUST.

NOTE:

Remember that ACTION bus value is equal to current state code.

You can change sequence of cards by editing **F2** stimulator formula.

Sample simulation results are shown below.



Conclusion

As presented in this tutorial State Editor is a very powerful tool for creating control logic in VHDL. The graphical diagrams can be used for better documentation of the design and to generate efficient and reliable synthesis results.

Quiz

The design implementation presented in this tutorial has a logical flaw. Please try to find a sequence of events that will cause it to work incorrectly. To find out the answer you can send email to support@aldec.com.

State Editor Resources

The complete documentation of the State Editor is provided on the Foundation CD in the file ACTIVE\DOC\SYNTOOLS.PDF.

For additional sample designs, documentation and updates to the State Editor please consult the ALDEC Website at www.aldec.com.