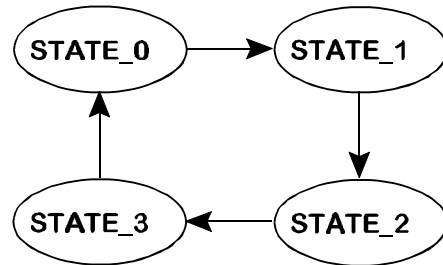


STATE ASSIGNMENTS AND STATECAD

StateCAD allows for four different types of state assignments. In these notes we will look at each of these. We shall use the machine depicted in the diagram to the right. It has no inputs or outputs since we will be concerned here only with the state assignment aspect of this.



We will begin with the BINARY ENCODED state assignment since it is the most “traditional”. In this case, since we have four states we will need two state variables. This gives us four possible combinations, one for each state. StateCAD assigns these combinations sequentially starting with the first state entered. The VHDL created is shown below.

```
ARCHITECTURE BEHAVIOR OF ENCODED IS
    SIGNAL sreg      : std_logic_vector (1 DOWNTO 0);
    SIGNAL next_sreg : std_logic_vector (1 DOWNTO 0);
    CONSTANT STATE0  : std_logic_vector (1 DOWNTO 0) := "00";
    CONSTANT STATE1  : std_logic_vector (1 DOWNTO 0) := "01";
    CONSTANT STATE2  : std_logic_vector (1 DOWNTO 0) := "10";
    CONSTANT STATE3  : std_logic_vector (1 DOWNTO 0) := "11";

BEGIN
    PROCESS (CLK, next_sreg)
    BEGIN
        IF CLK='1' AND CLK'event THEN sreg <= next_sreg;
        END IF;
    END PROCESS;
    PROCESS (sreg)
    BEGIN
        next_sreg<=STATE0;
        CASE sreg IS
            WHEN STATE0 =>
                next_sreg<=STATE1;
            WHEN STATE1 =>
                next_sreg<=STATE2;
            WHEN STATE2 =>
                next_sreg<=STATE3;
            WHEN STATE3 =>
                next_sreg<=STATE0;
            WHEN OTHERS =>
                next_sreg<=STATE0;
        END CASE;
    END PROCESS;
END BEHAVIOR;
```

We have omitted the ENTITY section. It is pretty standard and does not change with assignment. First, the state variables are declared as a two-bit vector (sreg). A second signal, next_sreg, is also defined for reasons to be seen soon. Then four constants are included defining the four states.

There are two PROCESS statements for this machine. The first actually causes the transitions to occur. Specifically IF CLK=' 1' AND CLK' event THEN sreg <= next_sreg; says that on the rising edge of the clock the state variables (sreg) should assume the value of next_sreg.

The second process defines the state transitions. In this simple case, each state is directed to the next. The equations programmed into the PLD are as below.

$$\begin{aligned} \text{sreg_0.D} &= \text{/sreg_0.Q} \\ \text{sreg_1.D} &= \text{sreg_1.Q} * \text{/sreg_0.Q} + \text{/sreg_1.Q} * \text{sreg_0.Q} \end{aligned}$$

(Clock and asynchronous signals have not been included). This will turn out to be a less than optimal state assignment. We can specify our own state assignment in StateCAD by assigning the state variable values in each state. We will change states 2 and 3 so that state 2 is "11" and state 3 is "10". The VHDL generated is as below.

```

ARCHITECTURE BEHAVIOR OF MANUAL IS
    SIGNAL sreg : std_logic_vector (1 DOWNTO 0);
    SIGNAL next_sreg : std_logic_vector (1 DOWNTO 0);
    CONSTANT STATE0 : std_logic_vector (1 DOWNTO 0) := "00";
    CONSTANT STATE1 : std_logic_vector (1 DOWNTO 0) := "10";
    CONSTANT STATE2 : std_logic_vector (1 DOWNTO 0) := "11";
    CONSTANT STATE3 : std_logic_vector (1 DOWNTO 0) := "01";
BEGIN
    PROCESS (CLK, next_sreg)
    BEGIN
        IF CLK=' 1' AND CLK' event THEN
            sreg <= next_sreg;
        END IF;
    END PROCESS;

    PROCESS (sreg)
    BEGIN
        next_sreg<=STATE0;
        CASE sreg IS
            WHEN STATE0 =>
                next_sreg<=STATE1;
            WHEN STATE1 =>
                next_sreg<=STATE2;
            WHEN STATE2 =>
                next_sreg<=STATE3;
            WHEN STATE3 =>
                next_sreg<=STATE0;
                HECK<=' 0';
            WHEN OTHERS =>
                next_sreg<=STATE0;
        END CASE;
    END PROCESS;
END BEHAVIOR;

```

It should be observed that this VHDL is identical to the first except for the CONSTANT declarations. The logic implemented for this is as below.

$$\begin{aligned} \text{sreg_1.D} &= \text{/sreg_0.Q} \\ \text{sreg_0.D} &= \text{sreg_1.Q} \end{aligned}$$

This is, obviously, a better state assignment. There is a third method, the ENUMERATED state assignment. The VHDL code generated for this is as below.

```

ARCHITECTURE BEHAVIOR OF ENUMER IS
  TYPE type_sreg IS (STATE0, STATE1, STATE2, STATE3);
  SIGNAL sreg, next_sreg : type_sreg;
BEGIN
  PROCESS (CLK, next_sreg)
  BEGIN
    IF CLK='1' AND CLK'event THEN
      sreg <= next_sreg;
    END IF;
  END PROCESS;

  PROCESS (sreg)
  BEGIN
    next_sreg<=STATE0;
    CASE sreg IS
      WHEN STATE0 =>
        next_sreg<=STATE1;
      WHEN STATE1 =>
        next_sreg<=STATE2;
      WHEN STATE2 =>
        next_sreg<=STATE3;
      WHEN STATE3 =>
        next_sreg<=STATE0;
      WHEN OTHERS =>
        next_sreg<=STATE0;
    END CASE;
  END PROCESS;
END BEHAVIOR;

```

Here there are no constant declarations. Instead, the state variables are declared to be an ENUMERATED type with the possible values of "STATE", "STATE1", etc. The logic implemented is as below.

$$sregSBV_1.D = /sregSBV_1.Q$$

$$sregSBV_0.D = sregSBV_0.Q * /sregSBV_1.Q + /sregSBV_0.Q * sregSBV_1.Q$$

These are the same as for the first design since the state assignment was the same. The only particular advantage to this assignment is that it is only necessary to change the order of the "names" in the type declaration to do a different state assignment.

The last type of state assignment has not been generally practical in the past but with the use of CPLD's and FPGA's has become useful. In this assignment, there is one state variable for each state. We need four state variables for this machine. The VHDL for this is below.

```

ARCHITECTURE BEHAVIOR OF ONEHOT IS
-- State variables for machine sreg
SIGNAL STATE0, next_STATE0, STATE1, next_STATE1, STATE2, next_STATE2,
STATE3, next_STATE3 : std_logic;
BEGIN
PROCESS (CLK, next_STATE0, next_STATE1, next_STATE2, next_STATE3)
BEGIN
    IF CLK='1' AND CLK' event THEN
        STATE0 <= next_STATE0;
        STATE1 <= next_STATE1;
        STATE2 <= next_STATE2;
        STATE3 <= next_STATE3;
    END IF;
END PROCESS;

PROCESS (STATE0, STATE1, STATE2, STATE3)
BEGIN

    IF (( (STATE3='1')) THEN next_STATE0<='1';
    ELSE next_STATE0<='0';
    END IF;

    IF (( (STATE0='1')) THEN next_STATE1<='1';
    ELSE next_STATE1<='0';
    END IF;

    IF (( (STATE1='1')) THEN next_STATE2<='1';
    ELSE next_STATE2<='0';
    END IF;

    IF (( (STATE2='1')) THEN next_STATE3<='1';
    ELSE next_STATE3<='0';
    END IF;

    END PROCESS;
END BEHAVIOR;

```

The logic produced by this is shown below.

```

state0.D = state3.Q
state1.D = state0.Q
state2.D = state1.Q
state3.D = state2.Q

```

The advantage of this state assignment is not obvious in this example. The advantages which may be gained are:

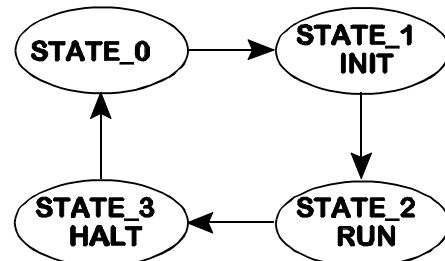
- ❑ In more complex machines, the equations for the next state typically have many product terms. CPLD's are more efficient (space-wise) when the number of product terms can be kept to four or less. One-hot assignment equations almost always have fewer product terms and may, for this reason, be more efficient in device utilization than encoded assignments.
- ❑ For this simple machine, it was very easy to figure out the optimum encoded state assignment. In a more complex machine, the optimum assignment (if any) may be hard to find. With one-hot, there is no such thing as a non-optimum assignment. The design

will always be the same.

Now, all of these machines have been unrealistic in that they have no outputs (in fact, it was necessary to add one before WARP2 would compile them!). Let's consider a more practical example.

This machine has the same states and transitions but has three outputs, INIT, RUN, and HALT.

The VHDL code generated by StateCAD using the one-hot assignment is shown below.



```
ARCHITECTURE BEHAVIOR OF ONEHOT IS
  SIGNAL STATE0, next_STATE0, STATE1, next_STATE1, STATE2, next_STATE2,
  STATE3, next_STATE3 : std_logic;
BEGIN
  PROCESS (CLK, next_STATE0, next_STATE1, next_STATE2, next_STATE3)
  BEGIN
    IF CLK=' 1' AND CLK' event THEN
      STATE0 <= next_STATE0;
      STATE1 <= next_STATE1;
      STATE2 <= next_STATE2;
      STATE3 <= next_STATE3;
    END IF;
  END PROCESS;
  PROCESS (STATE0, STATE1, STATE2, STATE3)
  BEGIN
    IF (( (STATE3=' 1' ))) THEN next_STATE0<=' 1' ;
    ELSE next_STATE0<=' 0' ;
    END IF;

    IF (( (STATE0=' 1' ))) THEN next_STATE1<=' 1' ;
    ELSE next_STATE1<=' 0' ;
    END IF;

    IF (( (STATE1=' 1' ))) THEN next_STATE2<=' 1' ;
    ELSE next_STATE2<=' 0' ;
    END IF;

    IF (( (STATE2=' 1' ))) THEN next_STATE3<=' 1' ;
    ELSE next_STATE3<=' 0' ;
    END IF;

    IF (( (STATE3=' 1' ))) THEN HALT<=' 1' ;
    ELSE HALT<=' 0' ;
    END IF;

    IF (( (STATE1=' 1' ))) THEN INIT<=' 1' ;
    ELSE INIT<=' 0' ;
    END IF;

    IF (( (STATE2=' 1' ))) THEN RUN<=' 1' ;
    ELSE RUN<=' 0' ;
    END IF;
  END PROCESS; END BEHAVIOR;
```

The logic generated by this module is listed below.

```
state0.D = halt.Q  
init.D = state0.Q  
run.D = init.Q  
halt.D = run.Q
```

What happened to states 1, 2, and 3? VHDL reported the following:

```
Aliasing next_state0 to halt  
Aliasing next_state2 to init  
Aliasing next_state3 to run
```

```
Note: Removed unneeded node 'state3'.  
Note: Removed unneeded node 'state1'.  
Note: Removed unneeded node 'state2'.
```

The output signals and the three states were the same so the states were removed. The end result is that this machine requires four macrocells. Using the other encoded state assignments would have required five (2 state variables, three outputs). This is further incentive for using the one-hot state assignment.