Using AHDL to Design State Machines

Finite state machine is another name for sequential circuits. A two-bit up-down counter can be described as a state machine with one input and two outputs:

There are many ways to design state machines using AHDL. Here are one design for the two-bit up-down counter:

```ahdl
SUBDESIGN two_bit
  (  count[1..0] : OUTPUT;
      clock    : INPUT;
      up       : INPUT;
  )

VARIABLE
  ss : MACHINE WITH STATES(s0, s1, s2, s3);

BEGIN
  ss.clk = clock; % Specify the clock for the state machine %

  CASE ss IS
    WHEN s0 =>
      count[] = B"00";
      IF (up == 1) THEN ss = s1; ELSE ss = s3; END IF;
    WHEN s1 =>
      count[] = B"01";
      IF (up == 1) THEN ss = s2; ELSE ss = s0; END IF;
    WHEN s2 =>
```
count[] = B"10";
IF (up == 1) THEN ss = s3; ELSE ss= s1; END IF;
WHEN s3 =>
  count[] = B"11";
  IF (up == 1) THEN ss = s0; ELSE ss= s2; END IF;
END CASE;

END;

The two-bit up-down counter is a Moore machine — i.e., the outputs of the machine depend only on the current state, and not on the current input. You can design a Moore machine by specifying a bit pattern associated with each state. In this example, we use a state transition table rather than a CASE statement. The count[1..0] outputs are directly associated with bits of the state machine. This means that the count[1..0] outputs will be the outputs of flip-flops, and will not change value until the machine changes states.

SUBDESIGN two_bit
(  count[1..0] : OUTPUT;
  clock : INPUT;
  up    : INPUT;
)

VARIABLE
  ss : MACHINE OF BITS (count[1..0])
  WITH STATES(s0 = B"00",
               s1 = B"01",
               s2 = B"10",
               s3 = B"11");

BEGIN
  ss.clk = clock; % Specify the clock for the state machine%

  TABLE
    % current     current     next    %
    % state       input       state    %
    ss,   up   =>  ss;
    s0,   1    =>  s1;
    s1,   1    =>  s2;
    s2,   1    =>  s3;
    s3,   1    =>  s0;
    s0,   0    =>  s3;
    s1,   0    =>  s0;
    s2,   0    =>  s1;
    s3,   0    =>  s2;

END TABLE;

END;
You can use AHDL to design state machines with asynchronous outputs, also called Mealy machines. Here is an example from your textbook:

![State Machine Diagram]

Here is an AHDL file to implement the design. This example shows how to reset a state machine. When reset goes high, the machine will be reset to the first state in the state machine list; in this case, that will be state A. The reset is done using the clrn and prn inputs to D flip-flops, so the reset is done as soon as reset goes high; it is not necessary to wait for a clock edge.

When in state B, the output will be 0 when the input is 0, and the output will be 1 when the input is 1. The output will change multiple times while in state B if the input changes multiple times. For a Moore machine, the output changes only when the machine switches from one state to another.

```
SUBDESIGN mealy
(
    clock : INPUT;
    reset : INPUT;
    w : INPUT;
    z : OUTPUT;
)

VARIABLE
    ss : MACHINE WITH STATES(A, B);

BEGIN
    ss.clk = clock; % Specify the clock for the state machine%
    ss.reset = reset; % Specify the reset for the state machine%

    CASE ss IS
        WHEN A =>
            if (w == GND) THEN
                z = GND;
                ss = A;
            else
                z = GND;
                ss = B;
            END IF;
        WHEN B =>
            if (w == 1) THEN
                z = B"1";
                ss = B;
```

3
Here is the same system designed using a state transition table:

```
SUBDESIGN mealy
(  
clock : INPUT;
reset : INPUT;
w : INPUT;
z : OUTPUT;
)

VARIABLE
  ss : MACHINE WITH STATES(A, B);

BEGIN
  ss.clk = clock;  % Specify the clock for the state machine %
  ss.reset = reset; % Specify the reset for the state machine %

  TABLE
    % current current current next %
    % state input output state %
    ss, w => z, ss;
    A, 0 => 0, A;
    A, 1 => 0, B;
    B, 0 => 0, A;
    B, 1 => 1, B;

END TABLE;

END;
```

Note that this just describes in a table what the state diagram described in a figure.