Drink Machine—State Machine Version

The next design is a vending control unit for a soft drink vending machine. The circuit reads signals from a coin-input unit and sends outputs to a change-dispensing unit and a drink-dispensing unit.

Input signals from the coin-input unit are nickel_in (nickel deposited), dime_in (dime deposited), and quarter_in (quarter deposited).

Outputs to the vending control unit are collect (collect coins) to the coin-input unit, nickel_out (nickel change) and dime_out (dime change) to the change-dispensing unit, and dispense (dispense drink) to the drink-dispensing unit.

The price of a drink is 35 cents. The first Verilog description for this design uses a state-machine description style. The description includes the // synopsys directive, which enables Design Compiler to extract an equivalent state machine.
Example A-3  Drink Machine—State Machine Version

'define vend_a_drink (D, dispense, collect) = (IDLE, 2'b11)

module drink_machine(nickel_in, dime_in, quarter_in,
collect, nickel_out, dime_out,
dispense, reset, clk);

parameter IDLE=0, FIVE=1, TEN=2, TWENTY_FIVE=3,
FIFTEEN=4, THIRTY=5, TWENTY=6, OWE_DIME=7;

input nickel_in, dime_in, quarter_in, reset, clk;
output collect, nickel_out, dime_out, dispense;
reg collect, nickel_out, dime_out, dispense;
reg [2:0] D, Q; /* state */
// synopsys state_vector Q
always @ ( nickel_in or dime_in or quarter_in or reset )
begin
  nickel_out     = 0;
dime_out       = 0;
dispense       = 0;
  collect       = 0;

  if ( reset ) D = IDLE;
  else begin
    D = Q;

    case ( Q )
    IDLE:
      if (nickel_in) D = FIVE;
      else if (dime_in) D = TEN;
      else if (quarter_in) D = TWENTY_FIVE;
    FIVE:
      if(nickel_in) D = TEN;
      else if (dime_in) D = FIFTEEN;
      else if (quarter_in) D = THIRTY;

    TEN:
      if (nickel_in) D = FIFTEEN;
      else if (dime_in) D = TWENTY;
      else if (quarter_in) 'vend_a_drink;

    TWENTY_FIVE:
      if (nickel_in) D = THIRTY;
      else if (dime_in) 'vend_a_drink;
      else if (quarter_in) begin
        'vend_a_drink;
        nickel_out = 1;
      dime_out = 1;
      end
end
Example A-3 (continued)  Drink Machine—State Machine Version

FI F T E E N :

if (nickel_in)  D = TWENTY;
else if (dime_in)  D = TWENTY_FIVE;
else if (quarter_in) begin
  'vend_a_drink;
  nickel_out = 1;
end

THIRTY :

if (nickel_in)  'vend_a_drink;
else if (dime_in) begin
  'vend_a_drink;
  nickel_out = 1;
end
else if (quarter_in) begin
  'vend_a_drink;
  dime_out = 1;
  D = OWE_DIME;
end

TWENTY :

if (nickel_in)  D = TWENTY_FIVE;
else if (dime_in)  D = THIRTY;
else if (quarter_in) begin
  'vend_a_drink;
  dime_out = 1;
end

OWE_DIME :

begin
  dime_out = 1;
  D = IDLE;
end

dcase
end
end

always @ (posedge clk ) begin
  Q = D;
end
endmodule
Example A-3 (continued)  Drink Machine—State Machine Version

Drink Machine—Count Nickels Version

This example uses the same design parameters as the preceding example, with the same input and output signals. In this version, a counter counts the number of nickels deposited. This counter is incremented by 1 if the deposit is a nickel, by 2 if it's a dime, and by 5 if it's a quarter.
Example A-4  Drink Machine—Count Nickels Version

module drink_machine(nickel_in,dime_in,quarter_in,collect, nickel_out,dime_out,dispense,reset,clk);

input nickel_in, dime_in, quarter_in, reset, clk;
output nickel_out, dime_out, collect, dispense;

reg nickel_out, dime_out, dispense, collect;
reg [3:0] nickel_count, temp_nickel_count;
reg temp_return_change, return_change;

always @ ( nickel_in or dime_in or quarter_in or
          collect or temp_nickel_count or
          reset or nickel_count or return_change) begin
  nickel_out = 0;
dime_out = 0;
dispense = 0;
collect = 0;
temp_nickel_count = 0;
temp_return_change = 0;

  // Check whether money has come in
  if ( !reset) begin
    temp_nickel_count = nickel_count;
    if (nickel_in)
      temp_nickel_count = temp_nickel_count + 1;
    else if (dime_in)
      temp_nickel_count = tempnickel_count + 2;
    else if (quarter_in)
      temp_nickel_count = temp_nickel_count + 5;
  end

  // correct amount deposited??
  if (temp_nickel_count >= 7) begin
    temp_nickel_count = temp_nickel_count - 7;
    dispense = 1;
collect = 1;
  end

  // return change
  if (return_change || collect) begin
    if (temp_nickel_count >= 2) begin
      dime_out = 1;
      temp_nickel_count = temp_nickel_count - 2;
      temp_return_change = 1;
    end
    if (temp_nickel_count == 1) begin
      nickel_out = 1;
      temp_nickel_count = temp_nickel_count - 1;
    end
  end
end
end
Example A-4 (continued) Drink Machine—Count Nickels Version

always @ (posedge clk) begin
    nickel_count = temp_nickel_count;
    return_change = temp_return_change;
end
endmodule

Carry-Lookahead Adder

This example shows how to build a 32-bit carry-lookahead adder. The adder is built by partitioning the 32-bit input into eight slices of four bits each. The PC module computes propagate and generate values for each of the eight slices.

Propagate (output p from PC) is 1 for a bit position if that position propagates a carry from the next lower position to the next higher position. Generate (output g) is 1 for a bit position if that position generates a carry to the next higher position, regardless of the carry-in from the next lower position.

The carry-lookahead logic reads the carry-in, propagate, and generate information computed from the inputs. It computes the carry value for each bit position. This logic makes the addition operation just an XOR of the inputs and the carry values.