Code:

```
module alu(clk, a, b, c, opcode, zf, sf, cf); input clk;
input [3:0] a, b;
output reg [3:0] c;
input [2:0] opcode;
output reg zf, sf, cf;
reg [1:0] bit_state, temp_sum;
reg temp_add_carry, temp_sub_carry;

parameter [3:0] opcode_reset = 3'b000, opcode_xor = 3'b001,
                                                                    opcode_add = 3'b010,
                                                                    opcode_and = 3'b011,
                                                                    opcode_sub = 3'b100;

parameter [1:0] bit_state_0 = 2'b00,
                                                                    bit_state_1 = 2'b01,
                                                                    bit_state_2 = 2'b10,
                                                                    bit_state_3 = 2'b11;


parameter bit_0 = 1'b0, bit_1 = 1'b1;

parameter [1:0] bit_10 = 2'b10,
                                                                    bit_11 = 2'b11;



always @(posedge clk)
begin
   if (opcode == opcode_reset)
   begin
          bit_state = bit_state_0;
   end

   else
   begin
          case(opcode)
                                opcode_xor: if(bit_state == bit_state_0) begin
                                                                                        c[0] = a[0] ^ b[0];

                                                                bit_state = bit_state_1;
           end
           else if(bit_state == bit_state_1)
          begin
               c[1] = a[1] ^ b[1];


                bit_state = bit_state_2;
          end
             else if(bit_state == bit_state_2)
```

```
            begin
                  c[2] = a[2] ^ b[2];

                  bit_state = bit_state_3;
            end

            else if(bit_state == bit_state_3)
            begin
                   c[3] = a[3] ^ b[3];

                  if (c[3] == bit_1) sf = bit_1;
                  else sf = bit_0;
                  cf = bit_0;
                        if (c[0] == bit_0 && c[1] == bit_0 &&
                        c[2] == bit_0 && c[3] == bit_0) zf = bit_1; else zf = bit_0;

                                          bit_state = bit_state_0;
                                                end


   opcode_add: if(bit_state == bit_state_0;
                        begin
              temp_sum = a[0] + b[0];
           if (temp_sum == bit_10)



begin
                  c[0] = bit_0;

temp_add_carry = bit_1;
                  end
                        else if (temp_sum == bit_11)
                  begin
                  c[0] = bit_1;

temp_add_carry = bit_1;
                   end
                   else
 begin
    c[0] = temp_sum;

temp_add_carry = bit_0;
   end

   bit_state = bit_state_1;
   end

  else if(bit_state == bit_state_1)
  begin
   temp_sum = a[1] + b[1] + temp_add_carry; if (temp_sum == bit_10)
   begin
   c[1] = bit_0;
```

```
temp_add_carry = bit_1;
   end
   else if (temp_sum == bit_11)
   begin
   c[1] = bit_1;

temp_add_carry = bit_1;
   end
   else
   begin
   c[1] = temp_sum;

temp_add_carry = bit_0;
   end

   bit_state = bit_state_2;
   end


   else if(bit_state == bit_state_2)
   begin
   temp_sum = a[2] + b[2] + temp_add_carry; if (temp_sum == bit_10)
   begin
   c[2] = bit_0;

temp_add_carry = bit_1;
   end
   else if (temp_sum == bit_11)
   begin
   c[2] = bit_1;

temp_add_carry = bit_1;
end
else
begin
c[2] = temp_sum;

temp_add_carry = bit_0;
end

bit_state = bit_state_3;
end

else if(bit_state == bit_state_3)
begin
temp_sum = a[3] + b[3] + temp_add_carry; if (temp_sum == bit_10)
begin
c[3] = bit_0;

temp_add_carry = bit_1;
end
else if (temp_sum == bit_11)
```

```
begin
    c[3] = bit_1;

    temp_add_carry = bit_1;
    end
    else
    begin
    c[3] = temp_sum;

    temp_add_carry = bit_0;
    end

    if (c[3] == bit_1) sf = bit_1;
    else sf = bit_0;
    if (temp_add_carry == bit_1) cf = bit_1; else cf = bit_0;
    if (c[0] == bit_0 && c[1] == bit_0 && c[2] == bit_0 && c[3] == bit_0) zf = bit_1;
    else zf = bit_0;

    bit_state = bit_state_0;
    end
     opcode_and: if(bit_state == bit_state_0) begin
    c[0] = a[0] & b[0];
     bit_state = bit_state_1;
      end

    else if(bit_state == bit_state_1)
    begin
    c[1] = a[1] & b[1];

    bit_state = bit_state_2;
    end

    else if(bit_state == bit_state_2)
    begin
    c[2] = a[2] & b[2];

    bit_state = bit_state_3;
    end

    else if(bit_state == bit_state_3)
    begin
      c[3] = a[3] & b[3];

     if (c[3] == bit_1) sf = bit_1;
    else sf = bit_0;
    cf = bit_0;
    if (c[0] == bit_0 && c[1] == bit_0 && c[2] == bit_0 && c[3] == bit_0) zf = bit_1;
     else zf = bit_0;

    bit_state = bit_state_0;
    end
opcode_sub: if(bit_state == bit_state_0)
begin
```

```
c[0] = a[0] ^ b[0];

if (a[0] == bit_0 && b[0] == bit_1) temp_sub_carry = bit_1;
else temp_sub_carry = bit_0;

bit_state = bit_state_1;
end

else if(bit_state == bit_state_1)
begin
 if (temp_sub_carry == bit_1) c[1] = ~(a[1] ^ b[1]); else c[1] = a[1] ^ b[1];

if (a[1] == bit_0 && b[1] == bit_1) temp_sub_carry = bit_1;
else temp_sub_carry = bit_0;

bit_state = bit_state_2;
end

else if(bit_state == bit_state_2)
begin
if (temp_sub_carry == bit_1) c[2] = ~(a[2] ^ b[2]); else c[2] = a[2] ^ b[2];

if (a[2] == bit_0 && b[2] == bit_1) temp_sub_carry = bit_1;
else temp_sub_carry = bit_0;

bit_state = bit_state_3;
end

   else if(bit_state == bit_state_3)
   begin
   if (temp_sub_carry == bit_1) c[3] = ~(a[3] ^ b[3]); else c[3] = a[3] ^ b[3];

   if (a[3] == bit_0 && b[3] == bit_1) temp_sub_carry = bit_1;
   else temp_sub_carry = bit_0;

   if (c[3] == bit_1) sf = bit_1;
   else sf = bit_0;
   if (temp_sub_carry == bit_1) cf = bit_1;
   else cf = bit_0;
   if (c[0] == bit_0 && c[1] == bit_0 && c[2] == bit_0 && c[3] == bit_0) zf = bit_1;
   else zf = bit_0;

   bit_state = bit_state_0;
   end

   endcase
    end
   end
   endmodule
```