```
1   /*
2      Non-restoring 32-bit division with quotient placed in Z[31:0] and remainder places in
    Z[63:32].
3      Operations takes 32 cycles to complete.
4   */
5
6   module DIV(
7
8       input [31:0] Q, // Dividend
9       input [31:0] M, // Divisor
10      input clk, resetn,
11
12      output [31:0] quotient,
13      output [31:0] remainder
14
15  );
16     reg [63:0] AQ_reg;
17     integer count;
18     wire [31:0] M_signed = (M[31]) ? -M : M;
19     wire [31:0] Q_signed = (Q[31]) ? -Q : Q;
20
21     always @(posedge clk) begin
22
23        if (~resetn) begin
24
25           AQ_reg = 64'b0;
26           count = 0;
27
28        end
29
30        else if (count == 0) begin
31
32           count = count + 1;
33
34           AQ_reg = {32'b0, Q_signed};
35
36        end
37
38        else if (count >= 1 && count <= 32) begin
39
40           count = count + 1;
41
42           AQ_reg = AQ_reg << 1;
43
44           if (AQ_reg[63] == 1'b0) begin
45
46              AQ_reg[63:32] = AQ_reg[63:32] - M_signed;
47
48           end
49
50           else begin
51
52              AQ_reg[63:32] = AQ_reg[63:32] + M_signed;
53
54           end
55
56           AQ_reg[0] = (AQ_reg[63] == 1'b0) ? 1'b1 : 1'b0;
57
58        end
59
60        else if (AQ_reg[63] == 1) begin
61
62           AQ_reg[63:32] = AQ_reg[63:32] + M_signed;
63
64        end
65
66     end
67
68
69
70     assign quotient =  (M[31] ^ Q[31]) ? -AQ_reg[31:0] : AQ_reg[31:0];
71
72     assign remainder =   AQ_reg[63:32];
73
74  endmodule
75
```

```
76   module DIV_tb;
77
78      // Declare inputs as reg type
79      reg [31:0] Q;
80      reg [31:0] M;
81      reg clk, resetn;
82
83      // Declare outputs as wire type
84      wire [31:0] quotient;
85      wire [31:0] remainder;
86
87      // Instantiate the DIV module
88      DIV uut (
89        .Q(Q),
90        .M(M),
91        .clk(clk),
92        .resetn(resetn),
93        .quotient(quotient),
94        .remainder(remainder)
95      );
96
97      // Clock generation
98      always begin
99        clk = 0;
100       forever #5 clk = ~clk;
101     end
102
103     initial begin
104         resetn = 0;
105         @ (posedge clk);
106         resetn = 1;
107
108         // Test Cases
109         run_test(32'd38, 32'd6, 32'd6, 32'd2);        // 38 / 6 = 6 remainder 2
110         run_test(32'd100, 32'd25, 32'd4, 32'd0);      // 100 / 25 = 4 remainder 0
111         run_test(32'b1, 32'd50, 32'd0, 32'd1);        // 1 / 50 = 0 remainder 1
112         run_test(32'd0, 32'd10, 32'd0, 32'd0);        // 0 / 10 = 0 remainder 0
113         run_test(-32'd38, 32'd6, -32'd6, 32'd2);      // -38 / 6 = -6 remainder 2
114         run_test(32'd38, -32'd6, -32'd6, 32'd2);      // 38 / -6 = -6 remainder 2
115         run_test(-32'd38, -32'd6, 32'd6, 32'd2);      // -38 / -6 = 6 remainder 2
116
117         $display("Testbench completed successfully" );
118
119         $stop;
120     end
121
122     task run_test(input [31:0] Q_in, input [31:0] M_in, input [31:0] expected_quotient , input
     [31:0] expected_remainder );
123       begin
124         resetn = 0;
125         @ (posedge clk);
126         resetn = 1;
127         Q = Q_in;
128         M = M_in;
129
130         #340; // Wait for calculation (34 clock cycles)
131
132         @ (posedge clk); @ (posedge clk);
133
134         if (quotient !== expected_quotient || remainder !== expected_remainder) begin
135             $display("Test Failed: Q=%d, M=%d -> Expected Quotient=%d, Remainder=%d but got
    Quotient=%d, Remainder=%d" ,
136                      Q_in, M_in, expected_quotient , expected_remainder , quotient, remainder);
137         end
138
139         @ (posedge clk);
140     end
141     endtask
142
143  endmodule
144
145
146
147
148
149
```

```
150
151
152
153
154
155
156
157
158
159
160
161
162
```