```
1   /*
2      multiplier_32b is a 32-bit multiplier leveraging both optimization structures: bit-pair
    recoding and carry-save addition.
3      It accepts M and Q as the multiplication and multiplier, respectively.
4   */
5
6
7   module multiplier_32b (
8      input [31:0] M, Q,
9      output [63:0] result
10     );
11
12     // All relevant variations of M for booth augend selection.
13     wire [32:0] Q_shifted = {Q, 1'b0}; // Left shifted Q by 1 such that the i-1 Booth check
    is valid with i = 0.
14     wire [31:0] negM = -M;
15     wire [32:0] Mx2 = {M, 1'b0};
16     wire [32:0] negMx2 = {negM, 1'b0};
17
18     // Augends.
19     reg [63:0] partial_products [15:0];
20
21     integer i;
22     // Perform Booth Augend Selection
23     always @(*) begin
24        for (i = 0; i < 31; i = i+2) begin
25           // Choose variant of M based on partial_products
26           case ({Q_shifted[i+2], Q_shifted[i+1], Q_shifted[i]})
27
28              // All values are properly sign-extended.
29              3'b000: partial_products[i>>1] = 64'b0; // 0 x M
30              3'b001: partial_products[i>>1] = {{32{M[31]}}, M}; // +1 x M
31              3'b010: partial_products[i>>1] = {{32{M[31]}}, M}; // +1 x M
32              3'b011: partial_products[i>>1] = {{31{Mx2[32]}}, Mx2}; // +2 x M
33              3'b100: partial_products[i>>1] = {{31{negMx2[32]}}, negMx2}; // -2 x M
34              3'b101: partial_products[i>>1] = {{32{negM[31]}}, negM}; // -1 x M
35              3'b110: partial_products[i>>1] = {{32{negM[31]}}, negM}; // -1 x M
36              3'b111: partial_products[i>>1] = 64'b0; // 0 x M
37              default: partial_products[i>>1] = 64'b0;
38           endcase
39           // Apply appropriate shift before addition.
40           partial_products[i>>1] = partial_products[i>>1] << i;
41        end
42     end
43
44     // Final operands after reduction process
45     wire [63:0] reduced1, reduced2;
46
47     // 16-to-2 CSA reducer.
48     CSA_tree_16to2 reduction (.augends(partial_products), .reduced1(reduced1), .reduced2(
    reduced2));
49
50     // Final carry-propagate stage, with no carry-in, nor carry-out (result for 32-bit mult.
    is 64-bits)
51     adder_64b carry_propagate (.cin(1'b0), .x(reduced1), .y(reduced2), .s(result)); // No
    cout.
52
53  endmodule
54
55
56  `timescale 1ns / 1ps
57
58  module multiplier_32b_tb;
59      reg [31:0] M, Q;
60      wire [63:0] result;
61
62      // Instantiate the multiplier
63      multiplier_32b dut (M, Q, Result);
64
65      // Test procedure
66      initial begin
67          // Test cases
68          M = 32'd0; Q = 32'd0; #10; // 0 * 0
69          $display("M=%d, Q=%d, result=%d", M, Q, result);
70
```

```
71            M = 32'd15; Q = 32'd10; #10; // 15 * 10
72            $display("M=%d, Q=%d, result=%d", M, Q, result);
73
74            M = -32'd15; Q = 32'd10; #10; // -15 * 10
75            $display("M=%d, Q=%d, result=%d", M, Q, result);
76
77            M = 32'd15; Q = -32'd10; #10; // 15 * -10
78            $display("M=%d, Q=%d, result=%d", M, Q, result);
79
80            M = -32'd15; Q = -32'd10; #10; // -15 * -10
81            $display("M=%d, Q=%d, result=%d", M, Q, result);
82
83            M = 32'h7FFFFFFF; Q = 32'h00000001; #10; // Largest positive * 1
84            $display("M=%d, Q=%d, result=%d", M, Q, result);
85
86            M = 32'h80000000; Q = 32'd1; #10; // Smallest negative * 1
87            $display("M=%d, Q=%d, result=%d", M, Q, result);
88
89            M = 32'hFFFFFFFF; Q = 32'hFFFFFFFF; #10; // -1 * -1
90            $display("M=%d, Q=%d, result=%d", M, Q, result);
91
92            $stop;
93        end
94    endmodule
95
96
```