```systemverilog
/*
    multiplier_32b is a 32-bit multiplier leveraging both optimization structures: bit-pair
recoding and carry-save addition.
    It accepts M and Q as the multiplication and multiplier, respectively.
*/


module multiplier_32b (
    input [31:0] M, Q,
    output [63:0] result
    );

    // All relevant variations of M for booth augend selection.
    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.
    wire [31:0] negM = -M;
    wire [32:0] Mx2 = {M, 1'b0};
    wire [32:0] negMx2 = {negM, 1'b0};

    // Augends.
    reg [63:0] partial_products [15:0];

    integer i;
    // Perform Booth Augend Selection
    always @(*) begin
        for (i = 0; i < 31; i = i+2) begin
            // Choose variant of M based on partial_products
            case ({Q_shifted[i+2], Q_shifted[i+1], Q_shifted[i]})

                // All values are properly sign-extended.
                3'b000: partial_products[i>>1] = 64'b0; // 0 x M
                3'b001: partial_products[i>>1] = {{32{M[31]}}, M}; // +1 x M
                3'b010: partial_products[i>>1] = {{32{M[31]}}, M}; // +1 x M
                3'b011: partial_products[i>>1] = {{31{Mx2[32]}}, Mx2}; // +2 x M
                3'b100: partial_products[i>>1] = {{31{negMx2[32]}}, negMx2}; // -2 x M
                3'b101: partial_products[i>>1] = {{32{negM[31]}}, negM}; // -1 x M
                3'b110: partial_products[i>>1] = {{32{negM[31]}}, negM}; // -1 x M
                3'b111: partial_products[i>>1] = 64'b0; // 0 x M
                default: partial_products[i>>1] = 64'b0;
            endcase
            // Apply appropriate shift before addition.
            partial_products[i>>1] = partial_products[i>>1] << i;
        end
    end

    // Final operands after reduction process
    wire [63:0] reduced1, reduced2;

    // 16-to-2 CSA reducer.
    CSA_tree_16to2 reduction (.augends(partial_products), .reduced1(reduced1), .reduced2(
reduced2));

    // Final carry-propagate stage, with no carry-in, nor carry-out (result for 32-bit mult.
is 64-bits)
    adder_64b carry_propagate (.cin(1'b0), .x(reduced1), .y(reduced2), .s(result)); // No
cout.

endmodule


`timescale 1ns / 1ps

module multiplier_32b_tb;
    reg [31:0] M, Q;
    wire [63:0] result;

    // Instantiate the multiplier
    multiplier_32b dut (M, Q, Result);

    // Test procedure
    initial begin
        // Test cases
        M = 32'd0; Q = 32'd0; #10; // 0 * 0
        $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
```