```
module skilltest1 (
    input  wire       Clk,
    input  wire       Reset,
    input  wire [3:0] Trigger,
    output wire [3:0] BCD0,
    output wire [3:0] BCD1,
    output wire [3:0] BCD2,
    output wire [3:0] BCD3
);

    // Debouncer and Single pulse input
    reg [3:0] prev_trigger; // Trigger on the previous Clk will update value
    reg trigger_action;     // Single pulse input to update value
    reg [15:0] debouncer_count;

    always @(posedge Clk) begin
        if (Reset) begin
            debouncer_count <= 0;
            prev_trigger <= 0;
            trigger_action <= 0;
        end else begin
            // trigger_action is single pulse, reset on the next Clk
            trigger_action <= 0;

            // When debouncer count is 0 try to sample Trigger
            if (debouncer_count == 0) begin
                // Wait for input (Don't start next count until there is input)
                if (Trigger != 0) begin
                    trigger_action <= 1;
                    prev_trigger <= Trigger;
                    debouncer_count <= debouncer_count + 1;
                end
            end else if (debouncer_count < 1024) begin
                // While count < 1024 increment count
                debouncer_count <= debouncer_count + 1;
            end else begin
                // Wait for button release to start new count
                if (Trigger == 0) begin
                    debouncer_count <= 0;
                end
            end
        end
    end

    // Arithmetic operation
    reg overflow;
    reg [15:0] value;

    always @(posedge Clk) begin
        if (Reset) begin
            overflow <= 0;
            value <= 1;
        end else if (trigger_action && !overflow) begin
            // To update value there must be a pulse and does not overflow
            // Do not update value if it will overflow because assign BCD is asynchronous
            if (prev_trigger[0]) begin
                if (value + 1 > 9999) begin
                    overflow <= 1;
                end else begin
                    value <= value + 1;
                end
            end
            else if (prev_trigger[1]) begin
                if (value + 2 > 9999) begin
                    overflow <= 1;
                end else begin
                    value <= value + 2;
                end
            end
            else if (prev_trigger[2]) begin
                if (value * 2 > 9999) begin
                    overflow <= 1;
                end else begin
                    value <= value * 2;
                end
            end
            else if (prev_trigger[3]) begin
```

```
                if (value * 3 > 9999) begin
                    overflow <= 1;
                end else begin
                    value <= value * 3;
                end
            end
        end
    end

    // Value assignment for each BCD (asynchronous)
    assign BCD0 = (overflow) ? 15 : (value / 1) % 10;
    assign BCD1 = (overflow) ? 15 : (value / 10) % 10;
    assign BCD2 = (overflow) ? 15 : (value / 100) % 10;
    assign BCD3 = (overflow) ? 15 : (value / 1000) % 10;

endmodule
```

```
module debouncer #(
    parameter SAMPLING_RATE = 100000,
    parameter COUNTER_WIDTH = 20
) (
    input  clk,
    input  rst,
    input  data_in,
    output reg data_out
);

    reg [COUNTER_WIDTH-1:0] counter;
    reg stable_val;

    always @(posedge clk) begin
        if (rst) begin
            counter <= 0;
            data_out <= 0;
            stable_val <= 0;
        end else begin
            if (data_in == stable_val) begin
                if (counter == SAMPLING_RATE - 1) begin
                    data_out <= stable_val;
                end else begin
                    counter <= counter + 1;
                end
            end else begin
                counter <= 0;
                stable_val <= data_in;
            end
        end
    end

endmodule
```

```
module single_pulser (
    input clk,
    input rst,
    input in,
    output reg out
);
    reg q;
    always @(posedge clk) begin
        if (rst) begin
            q <= 0;
            out <= 0;
        end else begin
            q <= in;
            out <= in & (~q);
        end
    end
endmodule
```

Operators

Bitwise Operators

a & b	a AND b
a | b	a OR b
a ^ b	a XOR b
~a	NOT a
a ~^ b	a XNOR b

Arithmetic Operators

a + b	a plus b
a - b	a minus b
a * b	a multiplied by b
a / b	a divided by b
a % b	a modulo b
a ** b	a to the power of b

Avoid these 4 operators

Logical Operators

a && b	return true if a and b are true
a || b	return true if a or b are true
!a	not a

Operators

Relational Operators

a < b	a less than b
a > b	a more than b
a <= b	a less than or equal to b
a >= b	a more than or equal to b

Shift Operators

<<	Logical left shift
>>	Logical right shift
<<<	Arithmetic left shift
>>>	Arithmetic right shift

Relational Operators

a === b	a equal b (including x and z)
a !== b	a not equal to b (including x and z)
a == b	a equal to b
a != b	a not equal to b

Other Operators

{a,b}	Concatenate a and b
{b{a}}	Replicate a for b times