```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

entity TB_Adder is
end TB_Adder;

architecture behavior of TB_Adder is
    constant N : integer := 64;

    -- DUT ports
    signal TBA, TBB : std_logic_vector(N-1 downto 0) := (others => '0');
    signal TBCin    : std_logic := '0';
    signal TBS      : std_logic_vector(N-1 downto 0);
    signal TBCout   : std_logic;
    signal TBOvfl   : std_logic;

    -- Test-vector file
    constant TestVectorFile : string := "TestVectors/Adder00.tvs";

    -- Timing constants
    constant PreStimTime    : time   := 1 ns;   -- Time to drive 'X' before applying stimuli
    constant PostStimTime   : time   := 300 ns; -- Maximum time to wait for outputs to stabilize
    constant StableTime     : time   := 50 ns;   -- Time window to consider signal stable

    -- Component declaration
    component TestUnit is
        port (
            A, B  : in std_logic_vector (N-1 downto 0);
            S : out std_logic_vector (N-1 downto 0);
            Cin : in std_logic;
            Cout, Ovfl : out std_logic
        );
    end component;

begin
    -- Device Under Test (DUT)
    DUT: TestUnit
        port map (
            A => TBA,
            B => TBB,
            Cin => TBCin,
            S => TBS,
            Cout => TBCout,
            Ovfl => TBOvfl
        );

    -- Stimulus Process with proper propagation delay measurement
    stimulus : process
        -- Simulation variables
        file      tvf : text;
        variable  L, L_parse : line;
        constant  MAXLEN : natural := 2048;
        variable  s : string(1 to MAXLEN);
        variable  vA, vB, vS : std_logic_vector(N-1 downto 0);
        variable  vCin, vCout, vOvfl : std_logic;
        variable  skip_line  : boolean;
        variable  idx        : natural := 0;
        variable  pass       : boolean;
        variable  OUTL       : line;

        -- Timing measurement variables
        variable StartTime, EndTime : time;
        variable PropDelay_S, PropDelay_Cout, PropDelay_Ovfl : time;
        variable MaxPropDelay_S, MaxPropDelay_Cout, MaxPropDelay_Ovfl : time := 0 ns;
        variable MaxPropDelay_Overall : time := 0 ns;

        -- Variables for stability detection
        variable last_S, current_S : std_logic_vector(N-1 downto 0);
        variable last_Cout, current_Cout : std_logic;
        variable last_Ovfl, current_Ovfl : std_logic;
        variable stable_start : time;
        variable outputs_stable : boolean;

        -- Failure counting
        variable total_tests : natural := 0;
        variable failed_tests : natural := 0;

    begin
        -- Initialize max delay tracking
        MaxPropDelay_S := 0 ns;
        MaxPropDelay_Cout := 0 ns;
        MaxPropDelay_Ovfl := 0 ns;
        MaxPropDelay_Overall := 0 ns;

        -- Initialize failure counting
        total_tests := 0;
        failed_tests := 0;

        -- Open test vector file
        file_open(tvf, TestVectorFile, read_mode);

        -- Report file name
        report "Using test vectors from file: " & TestVectorFile;

        -- Loop through every test vector
        while not endfile(tvf) loop
            readline(tvf, L);

            -- Skip blank lines
            if L'length = 0 then
                next;
            end if;
            -- Set skip_line to false to begin
            skip_line := false;

            -- Check if line is too long
            if L'length > MAXLEN then
                report "Input line exceeds MAXLEN=" & integer'image(MAXLEN) severity failure;
            end if;

            s := (others => ' ');
            s(1 to L'length) := L.all;

            -- Check for comments
            for i in s'range loop
                if s(i) > ' ' then
                    -- Set skip line to true if the line is a comment
                    if i < s'high and s(i) = '-' and s(i + 1) = '-' then
                        skip_line := true;
                    end if;
                    exit;
                end if;
            end loop;

            -- Skip line if skip_line is true
            if skip_line then
                next;
            end if;

            -- Rebuild the line to parse values
            L_parse := null;
            write(L_parse, s(1 to L'length));

            -- Parse: A B Cin S Cout Ovfl
            HREAD(L_parse, vA);
            HREAD(L_parse, vB);
            read (L_parse, vCin);
            HREAD(L_parse, vS);
            read (L_parse, vCout);
            read (L_parse, vOvfl);

            -- 1) Drive 'X' for PreStimTime (clear previous state)
            TBA   <= (others => 'X');
            TBB   <= (others => 'X');
            TBCin <= 'X';
            wait for PreStimTime;

            -- 2) Apply inputs and record start time
            TBA   <= vA;
            TBB   <= vB;
            TBCin <= vCin;
            StartTime := now;

            -- 3) Simple and reliable stability detection
            -- Wait for outputs to change from initial state
            wait until TBS'event or TBCout'event or TBOvfl'event;

            -- Now monitor outputs until they stabilize for StableTime duration
            stable_start := now;
            outputs_stable := false;

            while not outputs_stable and (now - StartTime < PostStimTime) loop
                -- Record current output values
                last_S := TBS;
                last_Cout := TBCout;
                last_Ovfl := TBOvfl;

                -- Wait a small delta time
                wait for 1 ns;

                -- Check if outputs have changed
                current_S := TBS;
                current_Cout := TBCout;
                current_Ovfl := TBOvfl;

                if (last_S = current_S) and (last_Cout = current_Cout) and (last_Ovfl = current_Ovfl) then
                    -- Outputs haven't changed in this check
                    if (now - stable_start) >= StableTime then
                        outputs_stable := true;
                    end if;
                else
                    -- Outputs changed, reset stability timer
                    stable_start := now;
                end if;
            end loop;

            if outputs_stable then
                EndTime := stable_start + StableTime; -- When stability was confirmed
            else
                -- Timeout occurred
                EndTime := StartTime + PostStimTime;
                report "Timeout waiting for outputs to stabilize at measurement " & integer'image(idx)
                       severity warning;
            end if;

            -- 4) Calculate ACTUAL propagation delays
            PropDelay_S := EndTime - StartTime;
            PropDelay_Cout := EndTime - StartTime;
            PropDelay_Ovfl := EndTime - StartTime;

            -- Update maximum propagation delays
            if PropDelay_S > MaxPropDelay_S then
                MaxPropDelay_S := PropDelay_S;
            end if;
            if PropDelay_Cout > MaxPropDelay_Cout then
                MaxPropDelay_Cout := PropDelay_Cout;
            end if;
            if PropDelay_Ovfl > MaxPropDelay_Ovfl then
                MaxPropDelay_Ovfl := PropDelay_Ovfl;
            end if;
            if PropDelay_S > MaxPropDelay_Overall then
```

```vhdl
                MaxPropDelay_Overall := PropDelay_S;
            end if;
            if PropDelay_Cout > MaxPropDelay_Overall then
                MaxPropDelay_Overall := PropDelay_Cout;
            end if;
            if PropDelay_Ovfl > MaxPropDelay_Overall then
                MaxPropDelay_Overall := PropDelay_Ovfl;
            end if;

            -- Define pass condition
            pass := (TBS = vS) and (TBCout = vCout) and (TBOvfl = vOvfl);

            -- Count failures
            total_tests := total_tests + 1;
            if not pass then
                failed_tests := failed_tests + 1;
            end if;

            -- 5) Compute pass/fail and assert
            assert pass
                report "Mismatch: i=" & integer'image(idx) &
                    " A=" & to_hstring(TBA) &
                    " B=" & to_hstring(TBB) &
                    " Cin=" & std_logic'image(TBCin) &
                    "  got S=" & to_hstring(TBS) & " Cout=" & std_logic'image(TBCout) & " Ovfl=" & std_logic'image(TBOvfl) &
                    "  exp S=" & to_hstring(vS)  & " Cout=" & std_logic'image(vCout) & " Ovfl=" & std_logic'image(vOvfl)
            severity error;

            -- 6) Print one concise summary line with timing information
            OUTL := null;
            write(OUTL, idx);
            write(OUTL, string'(" A="));               write(OUTL, to_hstring(TBA));
            write(OUTL, string'(" B="));               write(OUTL, to_hstring(TBB));
            write(OUTL, string'(" Cin="));             write(OUTL, TBCin);
            write(OUTL, string'("  |  S="));           write(OUTL, to_hstring(TBS));
            write(OUTL, string'(" Cout="));            write(OUTL, TBCout);
            write(OUTL, string'(" Ovfl="));            write(OUTL, TBOvfl);
            write(OUTL, string'("  Delays(S/Cout/Ovfl)="));
            write(OUTL, PropDelay_S);   write(OUTL, string'("/"));
            write(OUTL, PropDelay_Cout);write(OUTL, string'("/"));
            write(OUTL, PropDelay_Ovfl);
            write(OUTL, string'("  status="));
            if pass then
                write(OUTL, string'("PASS"));
            else
                write(OUTL, string'("FAIL"));
            end if;
            writeline(output, OUTL);

            -- Increase index
            idx := idx + 1;
        end loop;

        -- Report worst-case delays and test summary at the end
        report "Simulation completed: reached end of " & TestVectorFile;
        report "Test Summary: " & integer'image(total_tests - failed_tests) & " passed, " &
                                  integer'image(failed_tests) & " failed out of " &
                                  integer'image(total_tests) & " total tests";
        report "Worst-case propagation delays - S: " & time'image(MaxPropDelay_S) &
               ", Cout: " & time'image(MaxPropDelay_Cout) &
               ", Ovfl: " & time'image(MaxPropDelay_Ovfl);
        report "Overall worst-case propagation delay: " & time'image(MaxPropDelay_Overall);

        -- Close test vector file
        file_close(tvf);
        wait;
    end process;
end architecture;
```