VL405 Design for Testability

Lab Assignment 2

Design and Fault Analysis of Even Parity Generator and Checker Circuits using VHDL

Student Name : Ayan Garg

Roll Number : B23484

School of Computing and Electrical Engineering (SCEE)
Indian Institute Of Technology Mandi
September 7, 2025

Contents

1 Objective

The objective of this experiment is to design and implement even parity generator and checker circuits using VHDL, and to analyze their behavior under stuck-at fault conditions (SA0 and SA1). The experiment aims to verify the correctness of the parity-based error detection mechanism, study the limitations of fault detection, and understand the practical applications of parity codes in digital system reliability.

For this lab, we will assume that there are no stuck-at faults at the inputs of any gate.

2 Theory

2.1 Stuck-At Fault Model

The stuck-at fault model is the most widely used fault model in digital testing. It represents a logical failure in which a signal line (input, output, or internal net) of a circuit is assumed to be permanently fixed at a logic level, irrespective of the actual inputs applied to the circuit. Such faults generally occur due to manufacturing defects like short-circuits, open-circuits, or wear-out effects in integrated circuits.

- **Stuck-at-0 (SA0):** The line is permanently tied to logic '0'.
- **Stuck-at-1 (SA1):** The line is permanently tied to logic '1'.

This abstraction does not refer to a physical short to ground or VDD but instead to the logical behavior of the faulty circuit. For instance, if a line is modeled as SA0, then whenever the circuit attempts to drive that line to '1', the actual response will still be '0', leading to a logic error.

The stuck-at fault model is preferred because:

1. It simplifies fault analysis while still covering a wide range of practical defects.
2. Tests designed for single stuck-at faults often detect multiple other fault types.
3. It allows systematic test generation using Automatic Test Pattern Generation (ATPG).

2.2 Test Pattern Generation

To detect a stuck-at fault, we must apply test vectors that **differentiate the fault-free circuit from the faulty circuit**. The general strategy is:

- **SA0 detection:** Apply an input combination that would force the fault-free output to be '1'. If the output remains '0', the fault is detected.
- **SA1 detection:** Apply an input combination that would force the fault-free output to be '0'. If the output remains '1', the fault is detected.

Thus, an effective test set is one where every node in the circuit is sensitized to both '0' and '1' conditions at least once, ensuring that both SA0 and SA1 faults are observable at the primary outputs.

2.3 Gate-Specific Fault Detection Requirements

2.3.1 3-Input Even Parity Generator

A 3-input even parity generator outputs a parity bit such that the total number of '1's in the inputs plus the parity bit is **even**. The function can be expressed as:

$$P = A \oplus B \oplus C$$

where A, B, C are inputs and P is the parity bit.

- **SA0 at Output (P/0):** To detect this, inputs must be applied such that the correct parity output is '1'. If the output remains stuck at '0', the fault is detected. Example test vectors: 001, 010, 100, 111.
- **SA1 at Output (P/1):** To detect this, inputs must be applied such that the correct parity output is '0'. If the output remains stuck at '1', the fault is detected. Example test vectors: 000, 011, 101, 110.

Table 1: Truth Table for 3-Bit Even Parity Generator

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2.3.2 4-Bit Even Parity Checker

A 4-bit even parity checker verifies whether the received 4-bit word (3 data + 1 parity) contains an **even number of 1's**. Its output, Parity Error Check (PEC), is given by:

$$PEC = A \oplus B \oplus C \oplus P$$

where P is the transmitted parity bit.

- **SA0 at Output (PEC/0):** Apply inputs that should produce '1' (odd number of 1's). If PEC remains at '0', the fault is detected.
- **SA1 at Output (PEC/1):** Apply inputs that should produce '0' (even number of 1's). If PEC remains at '1', the fault is detected.

Table 2: Truth Table for 4-Bit Even Parity Checker

A	B	C	P	PEC
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

2.3.3 Verifying the truth table of a 4-bit Even parity checker circuit

The given design implements a 4-bit even parity checker with fault injection capability. A parity checker is a combinational circuit that determines whether the number of logic '1's in a set of inputs is even or odd. For four data inputs A, B, C, and D, the parity output P is computed using XOR operations:

$$P = (A \oplus B) \oplus (C \oplus D)$$

Here, two intermediate signals are defined:

$$\text{temp1} = A \oplus B, \quad \text{temp2} = C \oplus D$$

and the final parity is obtained as $P = \text{temp1} \oplus \text{temp2}$.

To study fault tolerance, the circuit introduces the **stuck-at fault model** through a 3-bit control input, `Ctrl(2:0)`. A stuck-at fault forces an internal node to remain permanently at logic '0' (SA0) or logic '1' (SA1), irrespective of the actual inputs. This allows simulation of common manufacturing defects or physical failures in digital hardware.

The control signal has the following functionality:

- `Ctrl(2) = 0`: Normal operation, no faults injected.
- `Ctrl(2) = 1`: Fault mode enabled.
  - `Ctrl(1) = 0`: Fault applied to `temp1` ($A \oplus B$).
    - `Ctrl(0) = 0`: `temp1` stuck-at-0.
    - `Ctrl(0) = 1`: `temp1` stuck-at-1.
  - `Ctrl(1) = 1`: Fault applied to `temp2` ($C \oplus D$).
    - `Ctrl(0) = 0`: `temp2` stuck-at-0.

* `Ctrl(0) = 1`: `temp2` stuck-at-1.

Thus, the design can be operated in five distinct modes:

1. Normal mode (`Ctrl = 000`)
2. temp1 stuck-at-0 (`Ctrl = 100`)
3. temp1 stuck-at-1 (`Ctrl = 101`)
4. temp2 stuck-at-0 (`Ctrl = 110`)
5. temp2 stuck-at-1 (`Ctrl = 111`)

This fault injection mechanism enables verification of the complete truth table of the parity checker under both fault-free and faulty conditions. By comparing the observed output P with the expected parity result for all input combinations, the behavior of the circuit under different fault scenarios can be thoroughly analyzed.

3 VHDL Implementation

3.1 3-Input Even Parity Generator

3.1.1 3-Input Even Parity Generator Entity and Architecture

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity three_bit_even_parity is
port(
A: in std_logic;
B: in std_logic;
C: in std_logic;
P: out std_logic
);
end entity three_bit_even_parity;

architecture Behavioral of three_bit_even_parity is
signal temp : std_logic;
begin
temp <= A xor B;
P <=  temp xor C;
end Behavioral;
```

Listing 1: 3-Input Even Parity Generator Implementation

3.1.2 3-Input Even Parity Generator Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity three_bit_even_parity_tb is
end three_bit_even_parity_tb;

architecture test of three_bit_even_parity_tb is
component three_bit_even_parity
port(
A: in std_logic;
B: in std_logic;
C: in std_logic;
P: out std_logic
);
end component three_bit_even_parity;

signal test_A: std_logic;
signal test_B: std_logic;
signal test_C: std_logic;
signal test_P: std_logic;

begin
UUT : three_bit_even_parity
```

```
port map(
A=>test_A,
B=> test_B,
C=> test_C,
P=>test_P
);

stimulus_process: process
begin
test_A <= '1';
test_B <= '1';
test_C <= '1';
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at odd function output!" severity error;
else
    report "Odd funtion output is correct" severity note;
end if;

test_A <= '0';
test_B <= '0';
test_C <= '1';
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

test_A <= '1';
test_B <= '0';
test_C <= '1';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

test_A <= '0';
test_B <= '1';
test_C <= '1';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

test_A <= '1';
test_B <= '1';
test_C <= '0';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at odd function output!" severity error;
else
    report "Odd funtion output is correct" severity note;
end if;

test_A <= '0';
test_B <= '0';
test_C <= '0';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

test_A <= '1';
test_B <= '0';
test_C <= '0';
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

test_A <= '0';
test_B <= '1';
test_C <= '0';
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at odd function output!" severity error;
else
    report "odd function output is correct" severity note;
end if;

wait for 10ns;
end process stimulus_process;
```

```
end test;
```

Listing 2: 3-Input Even Parity Generator Testbench for Stuck-At Fault Detection

3.2 4-Input Even Parity Generator

3.2.1 4-Input Even Parity Generator Entity and Architecture

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity four_bit_even_parity_checker is
port(
A: in std_logic;
B: in std_logic;
C: in std_logic;
D: in std_logic;
P: out std_logic
);
end entity four_bit_even_parity_checker;

architecture Behavioral of four_bit_even_parity_checker is
signal temp1 : std_logic;
signal temp2 : std_logic;
begin
temp1 <= A xor B;
temp2 <= C xor D;
P <=  temp1 xor temp2;
end Behavioral;
```

Listing 3: 3-Input Even Parity Generator Implementation

3.2.2 4-Input Even Parity Generator Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity four_bit_even_parity_checker_tb is
end four_bit_even_parity_checker_tb;

architecture test of four_bit_even_parity_checker_tb is
component four_bit_even_parity_checker
 port(
A: in std_logic;
B: in std_logic;
C: in std_logic;
D: in std_logic;
P: out std_logic
);
end component four_bit_even_parity_checker;

signal test_A: std_logic;
signal test_B: std_logic;
signal test_C: std_logic;
signal test_D: std_logic;
signal test_P: std_logic;

begin
UUT : four_bit_even_parity_checker
port map(
A=>test_A,
B=> test_B,
C=> test_C,
D=> test_D,
P=> test_P
);

stimulus_process: process
begin
test_A <= '1';
test_B <= '1';
test_C <= '1';
test_D <= '1';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at even parity output!" severity error;
else
    report "Even parity output is correct" severity note;
end if;

test_A <= '1';
test_B <= '1';
test_C <= '0';
test_D <= '1';
```

```
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at even parity output!" severity error;
else
    report "Even parity output is correct" severity note;
end if;

test_A <= '1';
test_B <= '1';
test_C <= '0';
test_D <= '0';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at even parity output!" severity error;
else
    report "Even parity output is correct" severity note;
end if;

test_A <= '1';
test_B <= '0';
test_C <= '0';
test_D <= '0';
wait for 10ns;

if test_P = '0' then
   report "Stuck at 0 fault detected at even parity output!" severity error;
else
    report "Even parity output is correct" severity note;
end if;

test_A <= '0';
test_B <= '0';
test_C <= '0';
test_D <= '0';
wait for 10ns;

if test_P = '1' then
   report "Stuck at 1 fault detected at even parity output!" severity error;
else
    report "Even parity output is correct" severity note;
end if;

wait for 10ns;
end process stimulus_process;
end test;
```

Listing 4: 4-Input Even Parity Generator Testbench for Stuck-At Fault Detection

3.3 Verifying the truth table of a 4-bit even parity checker circuit

3.3.1 Verifying the truth table of a 4-bit even parity checker circuit Entity and Architecture

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity four_bit_parity_set is
    Port (
        A: in  std_logic;
        B: in  std_logic;
        C: in  std_logic;
        D: in  std_logic;
        P: out std_logic;
        Ctrl: in  std_logic_vector(2 downto 0)
    );
end four_bit_parity_set;

architecture Behavioral of four_bit_parity_set is
    signal temp1 : std_logic;
    signal temp2 : std_logic;
begin
    process (A, B, C, D, Ctrl)
    begin
        if Ctrl(2) = '0' then
            temp1 <= A xor B;
            temp2 <= C xor D;
            P <= temp1 xor temp2;
        else
            if Ctrl(1) = '0' then
                if Ctrl(0) = '0' then
                    temp1 <= '0';
                    temp2 <= C xor D;
                    P <= temp1 xor temp2;
                else
                    temp1 <= '1';
                    temp2 <= C xor D;
                    P <= temp1 xor temp2;
                end if;
            else
```

```
                if Ctrl(0) = '0' then
                    temp1 <= A xor B;
                    temp2 <= '0';
                    P <= temp1 xor temp2;
                else
                    temp1 <= A xor B;
                    temp2 <= '1';
                    P <= temp1 xor temp2;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

Listing 5: 3-Input Even Parity Generator Implementation

3.3.2 Verifying the truth table of a 4-bit even parity checker circuit Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity four_bit_parity_set_tb is
end four_bit_parity_set_tb;

architecture test of four_bit_parity_set_tb is
component four_bit_even_parity_checker
 port(
A: in std_logic;
B: in std_logic;
C: in std_logic;
D: in std_logic;
P: out std_logic
Ctrl: std_logic_vector(2 downto 0);
);
end component four_bit_even_parity_checker;
    type ctrl_array_t is array (natural range <>) of std_logic_vector(2 downto 0);
    constant CTRL_VEC : ctrl_array_t := (
        "000",
        "100",
        "101",
        "110",
        "111"
    );

    function expected_p (
        a, b, c, d : std_logic;
        ctrl : std_logic_vector(2 downto 0)
    ) return std_logic is
        variable t1_nom, t2_nom : std_logic;
        variable t1_eff, t2_eff : std_logic;
    begin
        t1_nom := a xor b;
        t2_nom := c xor d;

        if ctrl(2) = '0' then
            t1_eff := t1_nom;
            t2_eff := t2_nom;
        else
            if ctrl(1) = '0' then
                if ctrl(0) = '0' then
                    t1_eff := '0';
                else
                    t1_eff := '1';
                end if;
                t2_eff := t2_nom;
            else
                t1_eff := t1_nom;
                if ctrl(0) = '0' then
                    t2_eff := '0';
                else
                    t2_eff := '1';
                end if;
            end if;
        end if;

        return t1_eff xor t2_eff;
    end function;

    function bstr(b : std_logic) return string is
    begin
        if b = '1' then return "1"; else return "0"; end if;
    end function;

begin
    uut : entity work.four_bit_parity_set
        port map (
            A    => A,
            B    => B,
            C    => C,
            D    => D,
```

```
            P    => P,
            Ctrl => Ctrl
        );

    stim : process
        constant STEP : time := 10 ns;
        variable expP : std_logic;
        variable vec  : std_logic_vector(3 downto 0);
    begin
        for j in CTRL_VEC'range loop
            Ctrl <= CTRL_VEC(j);
            wait for STEP;

            for i in 0 to 15 loop
                vec := std_logic_vector(to_unsigned(i, 4));
                A <= vec(3);
                B <= vec(2);
                C <= vec(1);
                D <= vec(0);

                wait for STEP;

                expP := expected_p(A, B, C, D, Ctrl);

                assert P = expP
                report "Mismatch: Ctrl=" &
                       bstr(Ctrl(2)) & bstr(Ctrl(1)) & bstr(Ctrl(0)) &
                       " ABCD=" & bstr(A) & bstr(B) & bstr(C) & bstr(D) &
                       " DUT_P=" & bstr(P) & " EXP_P=" & bstr(expP)
                severity error;
            end loop;
        end loop;

        report "All parity and stuck-at tests completed successfully." severity note;
        wait;
    end process;
end architecture;
```

Listing 6: 4-Input Even Parity Generator Testbench for Stuck-At Fault Detection

4 Simulation Results

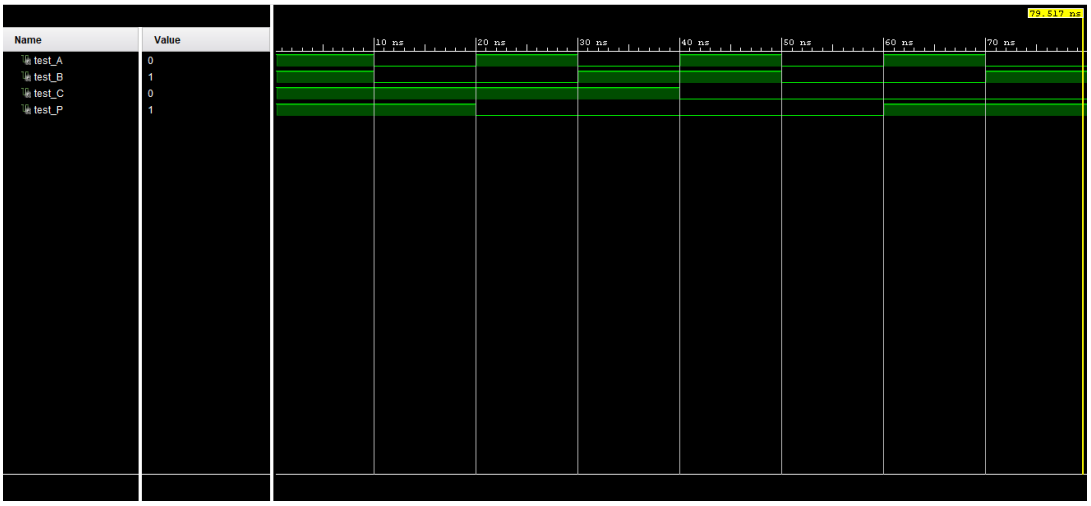

Figure 1: 3-Input Even Parity Generator Simulation Waveform for various inputs

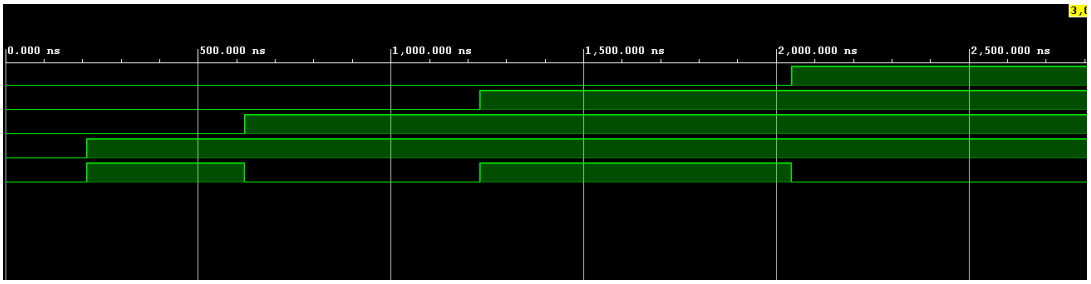

Figure 2: 4-Input Even Parity Generator Simulation Waveform for various inputs

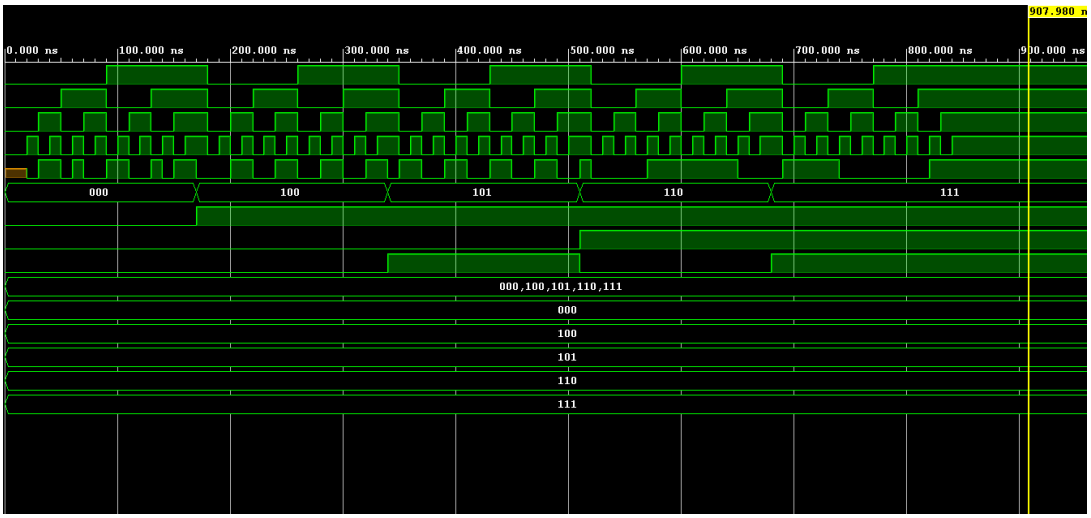

Figure 3: Verifying a 4-bit even parity checker

5 Conclusion

In this experiment, the design and verification of even parity generator and checker circuits were carried out using VHDL, with an emphasis on fault modeling through the stuck-at fault abstraction. Both 3-bit and 4-bit parity circuits were implemented, and their functional correctness was first established by exhaustively validating truth tables against simulation waveforms. The use of XOR-based logic confirmed the fundamental principle of parity coding—that the number of logical '1's in a code word must remain even under correct transmission.

The second part of the study introduced single stuck-at faults (SA0 and SA1) on internal nodes of the 4-bit parity checker. By controlling the injection of faults using a dedicated `Ctrl` input, the circuit could be operated in five distinct modes, covering both fault-free and faulty conditions. Simulation results showed that faults introduced at intermediate nodes (`temp1` and `temp2`) directly impacted the final parity output, making them observable at the primary output. This confirmed the testability of the design and validated the sufficiency of the applied test patterns to sensitize all nodes.