VL405 Design for Testability

Lab Assignment 3

Design and Fault Analysis using VHDL

Student Name : Ayan Garg

Roll Number : B23484

School of Computing and Electrical Engineering (SCEE)
Indian Institute Of Technology Mandi
September 15, 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 stuck-at faults in the 4-to-16 decoder, we must apply test vectors that **differentiate the fault-free circuit from the faulty circuit**. The general principle is:

- **SA0 detection:** Apply a test vector that forces the fault-free node/output to be logic '1'. If the faulty circuit produces a '0' instead, the fault is detected.
- **SA1 detection:** Apply a test vector that forces the fault-free node/output to be logic '0'. If the faulty circuit produces a '1' instead, the fault is detected.

For the given assignment, we consider faults on input lines w, z, and output line $D3$. The following test vectors are sufficient to detect the respective stuck-at faults:

- **Input z:**
  - z stuck-at-0: Apply $wxyz = 1101$.
    Fault-free output: $D_{13} = 1$.
    Faulty output: $D_{12} = 1$.
  - z stuck-at-1: Apply $wxyz = 1100$.
    Fault-free output: $D_{12} = 1$.
    Faulty output: $D_{13} = 1$.
- **Input w:**
  - w stuck-at-0: Apply $wxyz = 1100$.
    Fault-free output: $D_{12} = 1$.
    Faulty output: $D_4 = 1$.
  - w stuck-at-1: Apply $wxyz = 0100$.
    Fault-free output: $D_4 = 1$.
    Faulty output: $D_{12} = 1$.
- **Output $D3$:**
  - $D3$ stuck-at-0: Apply $wxyz = 0011$.
    Fault-free output: $D_3 = 1$.
    Faulty output: $D_3 = 0$.
  - $D3$ stuck-at-1: Apply $wxyz = 0000$.
    Fault-free output: $D_0 = 1$, $D_3 = 0$.
    Faulty output: $D_0 = 1$, $D_3 = 1$.

Summary of Test Vectors:

Table 1: Test Vectors for Stuck-at Fault Detection

Fault site	**Test Vector (wxyz)**	**Fault-free output**	**Faulty output**
z SA0	1101	D_{13}	D_{12}
z SA1	1100	D_{12}	D_{13}
w SA0	1100	D_{12}	D_4
w SA1	0100	D_4	D_{12}
$D3$ SA0	0011	$D_3 = 1$	$D_3 = 0$
$D3$ SA1	0000	$D_0 = 1, D_3 = 0$	$D_0 = 1, D_3 = 1$

2.3 Gate-Specific Fault Detection Requirements

2.3.1 Input z Stuck-at Faults

The input z drives both 3→8 decoders in the hierarchical 4→16 design. If z is stuck, the effective decoded index shifts by ± 1 within each active half of the decoder.

- **SA0 at z:** Whenever $z = 1$, the circuit instead behaves as if $z = 0$. Thus, the selected output line is shifted down by one. Example: For $wxyz = 1101$ the correct output is D_{13}, but under SA0 on z the output becomes D_{12}.

- **SA1 at z:** Whenever $z = 0$, the circuit behaves as if $z = 1$. Thus, the selected output line is shifted up by one. Example: For $wxyz = 1100$ the correct output is D_{12}, but under SA1 on z the output becomes D_{13}.

Table 2: Example Test Vectors for Stuck-at Fault Detection in 4→16 Decoder

Fault	**Test Vector (wxyz)**	**Faulty vs. Fault-Free Output**
z SA0	1101	$D_{13} \rightarrow D_{12}$
z SA1	1100	$D_{12} \rightarrow D_{13}$
w SA0	1100	$D_{12} \rightarrow D_4$
w SA1	0100	$D_4 \rightarrow D_{12}$
$D3$ SA0	0011	$D_3 = 1 \rightarrow$ None (all low)
$D3$ SA1	0000	$D_0 = 1 \rightarrow (D_0, D_3)$

2.3.2 Input w Stuck-at Faults

The input w selects between the lower (D0–D7) and upper (D8–D15) halves. If w is stuck, the activation swaps between halves, shifting the decoded index by ± 8.

- **SA0 at w:** When $w = 1$, the circuit instead behaves as if $w = 0$. The intended high line from D8–D15 is replaced by its alias in D0–D7. Example: For $wxyz = 1100$, the correct output is D_{12}, but under SA0 on w the output becomes D_4.
- **SA1 at w:** When $w = 0$, the circuit instead behaves as if $w = 1$. The intended high line from D0–D7 is replaced by its alias in D8–D15. Example: For $wxyz = 0100$, the correct output is D_4, but under SA1 on w the output becomes D_{12}.

Table 3: Test Vectors for w Stuck-at-0 (Global)

Input (wxyz)	**Fault-Free Output**	**Faulty Output (w=0)**
1000	D_8	D_0
1001	D_9	D_1
1010	D_{10}	D_2
1011	D_{11}	D_3
1100	D_{12}	D_4
1101	D_{13}	D_5
1110	D_{14}	D_6
1111	D_{15}	D_7

Table 4: Test Vectors for w Stuck-at-1 (Global)

Input (wxyz)	**Fault-Free Output**	**Faulty Output (w=1)**
0000	D_0	D_8
0001	D_1	D_9
0010	D_2	D_{10}
0011	D_3	D_{11}
0100	D_4	D_{12}
0101	D_5	D_{13}
0110	D_6	D_{14}
0111	D_7	D_{15}

2.3.3 Output $D3$ Stuck-at Faults

The output line $D3$ should be active only for input $wxyz = 0011$. Stuck-at faults on $D3$ either suppress it or cause it to spuriously assert alongside other outputs.

- **SA0 at $D3$:** When $wxyz = 0011$, no line is activated (the one-hot property is lost).
- **SA1 at $D3$:** $D3$ is always high. For input 0011, behavior is correct; for all others, $D3$ is erroneously high along with the correct line.

Table 5: Test Vectors for $D3$ Stuck-at-0

Input (wxyz)	**Fault-Free Output**	**Faulty Output**
0011	D_3	None (all low)

Table 6: Test Vectors for $D3$ Stuck-at-1

Input (wxyz)	**Fault-Free Output**	**Faulty Output**
0000	D_0	D_0, D_3
0001	D_1	D_1, D_3
0010	D_2	D_2, D_3
0011	D_3	D_3
0100	D_4	D_4, D_3
0101	D_5	D_5, D_3
0110	D_6	D_6, D_3
0111	D_7	D_7, D_3
1000	D_8	D_8, D_3
1001	D_9	D_9, D_3
1010	D_{10}	D_{10}, D_3
1011	D_{11}	D_{11}, D_3
1100	D_{12}	D_{12}, D_3
1101	D_{13}	D_{13}, D_3
1110	D_{14}	D_{14}, D_3
1111	D_{15}	D_{15}, D_3

2.3.4 Verification of the 4→16 Decoder with Fault Injection

The design implements a hierarchical 4→16 decoder with fault injection on input lines (z, w) and output ($D3$). The VHDL architecture allows selection of fault sites using control signals, and the testbench exhaustively applies all 16 input combinations under fault-free and faulty modes.

- In normal mode, exactly one line among D0–D15 is active (one-hot).
- Under input faults (z, w), aliasing effects occur: ± 1 shifts (for z) or ± 8 shifts (for w).
- Under output fault ($D3$), the one-hot property is broken: either missing activation (SA0) or double activation (SA1).

By comparing the observed output with the expected truth table, all injected faults are detectable. This methodology ensures comprehensive verification of the decoder's fault behavior and validates the fault models described.

3 VHDL Implementation

3.1 3×8 Decoder with Enable

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity decoder_3x8 is
    port (
        Ctrl1, Ctrl2, Ctrl3 : in std_logic;
        Enable : in std_logic;
        Y : out std_logic_vector (7 downto 0)
    );
end entity decoder_3x8;

architecture behavioral of decoder_3x8 is
    signal sel : std_logic_vector(2 downto 0);
begin
    sel <= Ctrl3 & Ctrl2 & Ctrl1;

    process(sel, Enable)
    begin
        if Enable = '1' then
            case sel is
                when "000" => Y <= "00000001";
                when "001" => Y <= "00000010";
                when "010" => Y <= "00000100";
                when "011" => Y <= "00001000";
                when "100" => Y <= "00010000";
                when "101" => Y <= "00100000";
                when "110" => Y <= "01000000";
                when "111" => Y <= "10000000";
                when others => Y <= (others => '0');
            end case;
        else
            Y <= (others => '0');
        end if;
    end process;
end architecture behavioral;
```

Listing 1: 3-to-8 Decoder with Enable

3.2 Top-level 4×16 Decoder with Fault Injection

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

entity test is
    port(
        sao : in std_logic;  -- output stuck-at enable
        CtrlA : in std_logic_vector(1 downto 0); -- input fault selector
        CtrlB : in std_logic_vector(2 downto 0); -- fault type
```

```
        CtrlO : in std_logic_vector(4 downto 0); -- output fault control

        x, y, z, w : in std_logic;               -- inputs
        OutA : out std_logic_vector (15 downto 0) -- outputs
    );
end test;

architecture Structural of test is
    signal fx1, fx2, fy1, fy2, fz1, fz2, fw1, fw2 : std_logic;
    signal fx, fy, fz, fw : std_logic;
    signal nfw1 : std_logic;

    signal mux_out : std_logic_vector(1 downto 0);
    signal mux_in_sig : std_logic;

    signal dec_low, dec_high : std_logic_vector(7 downto 0);
    signal out_int : std_logic_vector(15 downto 0);
begin
    ------------------------------------------------------------------
    -- Fault input selector
    ------------------------------------------------------------------
    process(x, y, z, w, CtrlA)
    begin
        case CtrlA is
            when "00" => mux_in_sig <= x; fy <= y; fz <= z; fw <= w;
            when "01" => mux_in_sig <= y; fx <= x; fz <= z; fw <= w;
            when "10" => mux_in_sig <= z; fx <= x; fy <= y; fw <= w;
            when others => mux_in_sig <= w; fx <= x; fy <= y; fz <= z;
        end case;
    end process;

    uut_mux: entity work.mux_special
        port map(in_sig => mux_in_sig, sel => CtrlB, out_sig => mux_out);

    -- Assign mux output to faulted inputs
    process(CtrlA, mux_out, x, y, z, w)
    begin
        fx1 <= x; fx2 <= x;
        fy1 <= y; fy2 <= y;
        fz1 <= z; fz2 <= z;
        fw1 <= w; fw2 <= w;

        case CtrlA is
            when "00" => fx1 <= mux_out(0); fx2 <= mux_out(1);
            when "01" => fy1 <= mux_out(0); fy2 <= mux_out(1);
            when "10" => fz1 <= mux_out(0); fz2 <= mux_out(1);
            when others => fw1 <= mux_out(0); fw2 <= mux_out(1);
        end case;
    end process;

    nfw1 <= not fw1;

    ------------------------------------------------------------------
    -- 4x16 Decoder using two 3x8 decoders
    ------------------------------------------------------------------
    dec_low_inst: entity work.decoder_3x8
        port map(Ctrl1 => fx1, Ctrl2 => fy1, Ctrl3 => fz1, Enable => nfw1, Y => dec_low);

    dec_high_inst: entity work.decoder_3x8
        port map(Ctrl1 => fx2, Ctrl2 => fy2, Ctrl3 => fz2, Enable => fw2, Y => dec_high);

    out_int <= dec_low & dec_high;

    ------------------------------------------------------------------
    -- Output stuck-at fault injection
    ------------------------------------------------------------------
    process(out_int, sao, CtrlO)
        variable temp : std_logic_vector(15 downto 0);
    begin
        temp := out_int;
        if sao = '1' then
            case CtrlO(4 downto 1) is
                when "0000" => temp(0)  := CtrlO(0);
                when "0001" => temp(1)  := CtrlO(0);
                when "0010" => temp(2)  := CtrlO(0);
                when "0011" => temp(3)  := CtrlO(0);
                when "0100" => temp(4)  := CtrlO(0);
                when "0101" => temp(5)  := CtrlO(0);
                when "0110" => temp(6)  := CtrlO(0);
                when "0111" => temp(7)  := CtrlO(0);
                when "1000" => temp(8)  := CtrlO(0);
                when "1001" => temp(9)  := CtrlO(0);
                when "1010" => temp(10) := CtrlO(0);
                when "1011" => temp(11) := CtrlO(0);
                when "1100" => temp(12) := CtrlO(0);
                when "1101" => temp(13) := CtrlO(0);
                when "1110" => temp(14) := CtrlO(0);
                when "1111" => temp(15) := CtrlO(0);
                when others => null;
            end case;
        end if;
        OutA <= temp;
    end process;
end Structural;
```

Listing 2: 4-to-16 Decoder with Fault Injection

3.3 MUX for Fault Injection

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

entity mux_special is
    Port (
        in_sig : in std_logic;
        sel    : in std_logic_vector(2 downto 0);
        out_sig : out std_logic_vector(1 downto 0)
    );
end mux_special;

architecture Behavioral of mux_special is
begin
    process(in_sig, sel)
    begin
        case sel is
            when "000" => out_sig <= in_sig & in_sig; -- pass-through
            when "001" => out_sig <= in_sig & '0';    -- lower=in, upper=0
            when "010" => out_sig <= '0' & in_sig;    -- lower=0, upper=in
            when "011" => out_sig <= "00";            -- both stuck-0
            when "100" => out_sig <= in_sig & '1';    -- lower=in, upper=1
            when "101" => out_sig <= '1' & in_sig;    -- lower=1, upper=in
            when "110" => out_sig <= "11";            -- both stuck-1
            when others => out_sig <= in_sig & in_sig;
        end case;
    end process;
end Behavioral;
```

Listing 3: MUX for Fault Injection

4 Simulation Results

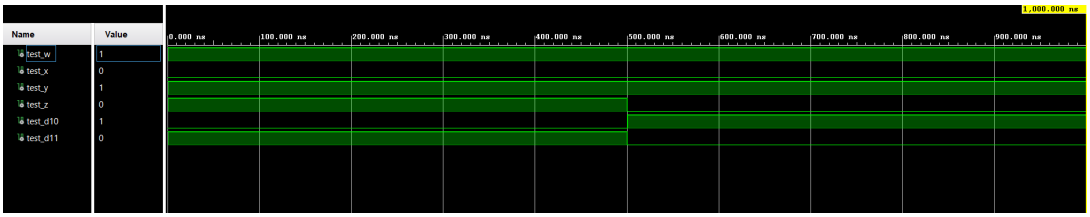

Figure 1: Stuck at fault at input Z

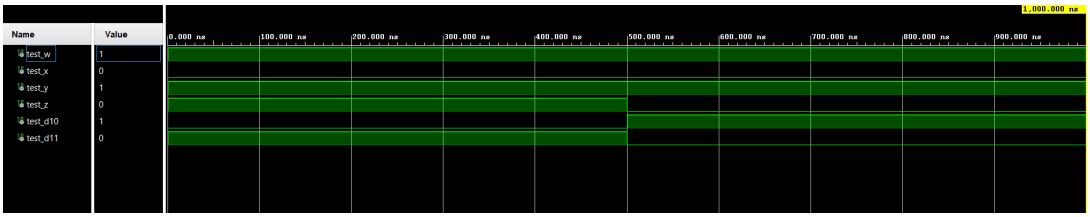

Figure 2: Stuck at fault at input W

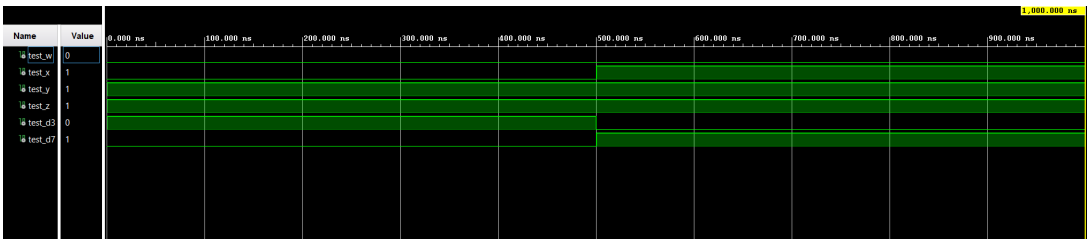

Figure 3: Stuck at fault at input D3

5 Conclusion

In this experiment, a 4-to-16 line decoder was successfully designed and verified using VHDL. The design was implemented hierarchically with two 3-to-8 decoders controlled by the most significant input bit, thereby reducing complexity and improving modularity.

In addition to the normal operation, fault injection capability was introduced through the stuck-at fault model. By integrating a special multiplexer and control signals, both input and output nodes could be forced into stuck-at-0 (SA0) or stuck-at-1 (SA1) conditions. This allowed the decoder to be tested under fault scenarios, demonstrating how manufacturing defects or physical errors may alter circuit behavior.

The truth tables and test vectors confirmed correct operation of the fault-free circuit as well as the ability of the design to simulate and detect faults. Therefore, this experiment not only reinforced the fundamentals of combinational logic design in VHDL but also highlighted the importance of Design-for-Testability (DFT) techniques in ensuring robust and reliable digital systems.