Digital Engineering
Lab 4

Y3890959
Y3878784

14th February 2023

Task A: Test Pattern Generation

3.1.1 Gate Level Fault Collapsing

Key: The inputs are X and Y. The output is Z. The outputs that don't match the desired have been highlighted yellow to show that the fault can be detected.

AND Gate

INPUT XY	DESIRED Z	Xs-a-0	Xs-a-1	Ys-a-0	Ys-a-1	Zs-a-0	Zs-a-1
00	0	0	0	0	0	0	1
01	0	0	1	0	0	0	1
10	0	0	0	0	1	0	1
11	1	0	1	0	1	0	1

Looking at the last row in the table above, we've for one test vector '11' being able to test 3 faults (Xs-a-0, Ys-a-0, and Zs-a-0) these faults are equivalent. Looking at the middle two rows with test vectors '01' and '10', the Zs-a-1 fault is being tested twice with the Xs-a-1 and Ys-a-1 faults, so we can say that Zs-a-1 is dominated by Xs-a-1 and Ys-a-1.

Equivalent	Dominant	Dominated
Xs-a-0	Xs-a-1	Zs-a-1
Ys-a-0	Ys-a-1	
Zs-a-0		

The same equivalent, dominant, and dominated rules are applied to the 'OR' and 'NOR' gates below.

OR Gate

INPUT XY	DESIRED Z	Xs-a-0	Xs-a-1	Ys-a-0	Ys-a-1	Zs-a-0	Zs-a-1
00	0	0	1	0	1	0	1
01	1	1	1	0	1	0	1
10	1	0	1	1	1	0	1
11	1	1	1	1	1	0	1

Equivalent	Dominant	Dominated
Xs-a-1	Xs-a-0	Zs-a-0
Ys-a-1	Ys-a-0	
Zs-a-1		

NOR Gate

INPUT XY	DESIRED Z	Xs-a-0	Xs-a-1	Ys-a-0	Ys-a-1	Zs-a-0	Zs-a-1
00	1	1	0	1	0	0	1
01	0	0	0	1	0	0	1
10	0	1	0	0	0	0	1
11	0	0	0	0	0	0	1

Equivalent	Dominant	Dominated
Xs-a-1	Xs-a-0	Zs-a-1
Ys-a-1	Ys-a-0	
Zs-a-0		

3.1.2 Circuit Fault Collapsing

First considering the AND gate with inputs A and B, the equivalent faults are As-a-0, Bs-a-0, and Gs-a-0. We can eliminate Bs-a-0 and Gs-a-0 as only As-a-0 is required to test the rest. Gs-a-1 is dominated by As-a-1 and Bs-a-1, therefore Gs-a-1 can also be eliminated.

Now considering the NOR gate with inputs G and I0, the equivalent faults are Gs-a-1, I0s-a-1, and Js-a-0. Gs-a-1 has already been eliminated, we can further eliminate I0s-a-1 and Js-a-0. Js-a-1 is being dominated by Gs-a-0 and I0s-a-0, therefore we can eliminate Js-a-1 too.

Next considering the OR gate with inputs J and K, the equivalent faults are Js-a-1, Ks-a-1, and Ls-a-1. Js-a-1 has already been eliminated, we can further eliminate both Ks-a-1 and Ls-a-1. Ls-a-0 is dominated by Js-a-0 and Ks-a-0, therefore we can also eliminate Ls-a-0.

Let's consider the AND gate with inputs C and D, the equivalent faults are Cs-a-0, Ds-a-0, and Hs-a-0. We can eliminate Ds-a-0 and Hs-a-0 as only Cs-a-0 is needed to test the other two. Hs-a-1 is dominated by Cs-a-1 and Ds-a-1, so we can further eliminate Hs-a-1.

Now considering the OR gate with inputs H and E, the equivalent faults are Hs-a-1, Es-a-1, and Is-a-1. Hs-a-1 has already been eliminated, we can further eliminate both Es-a-1 and Is-a-1. Is-a-0 is being dominated by Hs-a-0 and Es-a-0, hence we can eliminate Is-a-0.

Next considering the OR gate with inputs I1 and F, the equivalent faults are I1s-a-1, Fs-a-1, and Ks-a-1. Ks-a-1 has already been eliminated, we can also eliminate I1s-a-1 and Fs-a-1. Ks-a-0 is dominated by I1s-a-0 and Fs-a-0, we can therefore eliminate Ks-a-0.

Finally, considering the OR gate with inputs J and K, the equivalent faults are Js-a-1, Ks-a-1, and Ls-a-1. All of these have already been eliminated. Ls-a-0 is dominated by Js-a-0 and Ks-a-0, Ls-a-0 has already been eliminated.

After this process, we are left with the following 10 faults:

- Node As-a-0
- Node As-a-1
- Node I0s-a-0
- Node Ds-a-1
- Node Es-a-0
- Node I1s-a-0

- Node Bs-a-1
- Node Cs-a-0
- Node Cs-a-1
- Node Fs-a-0

3.1.3 The D-Algorithm

As-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1	D	1					D							
Step 2	D	1					D			0		D’		
Step 3	D	1					D			0		D’	0	D’
Step 4	D	1				0	D		0	0	0	D’	0	D’
Step 5	D	1			0	0	D	0	0	0	0	D’	0	D’
Step 6	D	1	X	0	0	0	D	0	0	0	0	D’	0	D’

As-a-1	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1	D’	1					D’							
Step 2	D’	1					D’		0	0	0	D		
Step 3	D’	1					D’		0	0	0	D	0	D
Step 4	D’	1				0	D’		0	0	0	D	0	D
Step 5	D’	1			0	0	D’	0	0	0	0	D	0	D
Step 6	D’	1	X	0	0	0	D’	0	0	0	0	D	0	D

I_0s-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1							0		1	D	1	D’		
Step 2	0	0					0		1	D	1	D’		
Step 3	0	0			1		0	1	1	D	1	D’		
Step 4	0	0	1	1	1		0	1	1	D	1	D’		
Step 5	0	0	1	1	1		0	1	1	D	1	D’	0	D’
Step 6	X	0	1	1	1		0	1	1	D	1	D’	0	D’

I_0s-a-1 is an undetectable fault. The yellow highlighted cells show the reason for this, with I_1 at ‘1’, K cannot be ‘0’ as this is an ‘OR’ gate.

Ds-a-1	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1			1	D’				D’						
Step 2			1	D’	0			D’	D’	D’	D’			
Step 3			1	D’	0	0		D’	D’	D’	D’		D’	
Step 4			1	D’	0	0		D’	D’	D’	D’	0	D’	D’
Step 5			1	D’	0	0	1	D’	D’	D’	D’	0	D’	D’
Step 6	1	1	1	D’	0	0	1	D’	D’	D’	D’	0	D’	D’

Es-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1					D			0	D	D	D			
Step 2			0	0	D			0	D	D	D			
Step 3			0	0	D	0		0	D	D	D		D	
Step 4			0	0	D	0		0	D	D	D	0	D	D
Step 5			0	0	D	0	1	0	D	D	D	0	D	D
Step 6	1	1	X	0	D	0	1	0	D	D	D	0	D	D

I_1s-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1					0			1	1	1	D			
Step 2			1	1	0			1	1	1	D			
Step 3			1	1	0	0		1	1	1	D		D	
Step 4			1	1	0	0		1	1	1	D	0	D	D
Step 5	X	X	1	1	0	0	X	1	1	1	D	0	D	D

Bs-a-1	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1	1	D’					D’							
Step 2	1	D’					D’		0	0	0	D		
Step 3	1	D’	0	0	0		D’	0	0	0	0	D		
Step 4	1	D’	0	0	0		D’	0	0	0	0	D	0	D
Step 5	1	D’	X	0	0	0	D’	0	0	0	0	D	0	D

Cs-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1			D	1				D						
Step 2			D	1	0			D	D	D	D			
Step 3			D	1	0	0		D	D	D	D		D	
Step 4			D	1	0	0		D	D	D	D	0	D	D
Step 5			D	1	0	0	1	D	D	D	D	0	D	D
Step 6	1	1	D	1	0	0	1	D	D	D	D	0	D	D

Cs-a-1	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1			D’	1				D’						
Step 2			D’	1	0			D’	D’	D’	D’			
Step 3			D’	1	0	0		D’	D’	D’	D’		D’	
Step 4			D’	1	0	0		D’	D’	D’	D’	0	D’	D’
Step 5			D’	1	0	0	1	D’	D’	D’	D’	0	D’	D’
Step 6	1	1	D’	1	0	0	1	D’	D’	D’	D’	0	D’	D’

Fs-a-0	**A**	**B**	**C**	**D**	**E**	**F**	G	H	I	I_0	I_1	J	K	**L**
Step 1						D					0		D	
Step 2						D			0	0	0	0	D	D
Step 3						D	1		0	0	0	0	D	D
Step 4	1	1				D	1		0	0	0	0	D	D
Step 5	1	1			0	D	1	0	0	0	0	0	D	D
Step 6	1	1	X	0	0	D	1	0	0	0	0	0	D	D

List of Detectable Faults

Fault	A	B	C	D	E	F
As-a-0	D	1	X	0	0	0
As-a-1	D’	1	X	0	0	0
Ds-a-1	1	1	1	D’	0	0
Es-a-0	1	1	X	0	D	0
I_1s-a-0	X	X	1	1	0	0
Bs-a-1	1	D’	X	0	0	0
Cs-a-0	1	1	D	1	0	0
Cs-a-1	1	1	D’	1	0	0
Fs-a-0	1	1	X	0	0	D

3.1.4 Merging Test Patterns

Fault	A	B	C	D	E	F
As-a-0	1	1	X	0	0	0
As-a-1	0	1	X	0	0	0
Ds-a-1	1	1	1	0	0	0
Es-a-0	1	1	X	0	1	0
I1s-a-0	X	X	1	1	0	0
Bs-a-1	1	0	X	0	0	0
Cs-a-0	1	1	1	1	0	0
Cs-a-1	1	1	0	1	0	0
Fs-a-0	1	1	X	0	0	1

By considering ‘D’ to be ‘1’ and ‘D” to be 0, we can merge the two rows highlighted green together, and the two rows highlighted yellow together.

3.1.5 List of Reduced Test Patterns

From the previous step, we can reduce the test patterns to 7 tests:

Test	A	B	C	D	E	F	L
Ds-a-1	1	1	1	0	0	0	0
As-a-1	0	1	0	0	0	0	1
Cs-a-0	1	1	1	1	0	0	1
Es-a-0	1	1	0	0	1	0	1
Bs-a-1	1	0	0	0	0	0	1
Cs-a-1	1	1	0	1	0	0	0
Fs-a-0	1	1	0	0	0	1	1

3.1.6 Fault Simulation

Pattern	A	B	C	D	E	F	G	H	I	I_0	I_1	J	K	L
As-a-1	s-a-1					s-a-1	s-a-1			s-a-1	s-a-1	s-a-0	s-a-1	s-a-1
Bs-a-1		s-a-1					s-a-1			s-a-1		s-a-0		s-a-0
Cs-a-0			s-a-0	s-a-0				s-a-0	s-a-0					
Cs-a-1			s-a-1		s-a-1			s-a-1	s-a-1					
Ds-a-1				s-a-1	s-a-1			s-a-1	s-a-1					
Es-a-0					s-a-0				s-a-0					
Fs-a-0						s-a-0							s-a-0	s-a-0

Task B: Built-In Self-Test (BIST)

3.2.1 Memory Content File

```
memory_initialization_radix=2;
memory_initialization_vector=
1110000,
0100001,
1111001,
1100101,
1000001,
1101000,
1100011;
```

3.2.2 Testbench VHDL

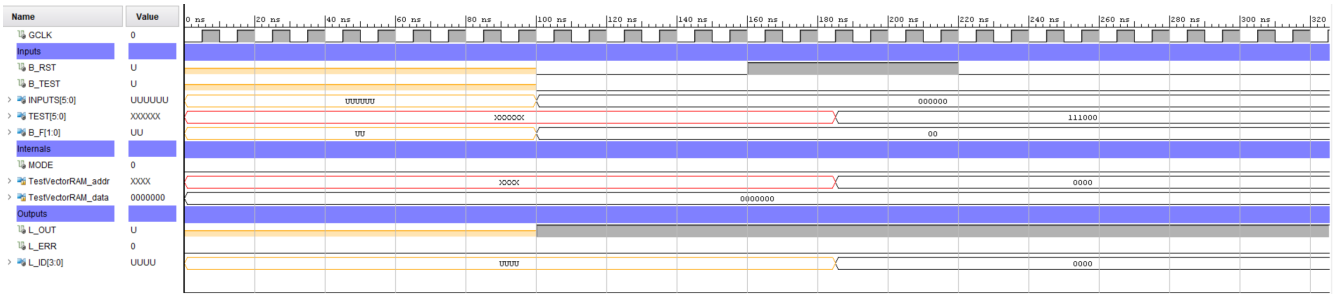
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bist_tb is
end bist_tb;

architecture Behavioral of bist_tb is

-- Instantiating the signals
signal GCLK, B_RST, B_TEST, L_OUT, L_ERR : STD_LOGIC;
signal INPUTS : STD_LOGIC_VECTOR (5 downto 0);
signal B_F : STD_LOGIC_VECTOR (1 downto 0);
signal L_ID : STD_LOGIC_VECTOR (3 downto 0);

constant clk_period : time := 10ns;

begin

UUT : entity work.top_level
    -- Mapping the signals to for the UUT
    PORT MAP (
        GCLK => GCLK,
        B_RST => B_RST,
        B_TEST => B_TEST,
        L_OUT => L_OUT,
        L_ERR => L_ERR,
        INPUTS => INPUTS,
        B_F => B_F,
        L_ID => L_ID
    );
```

```
clk_process : process
begin
    GCLK <= '0';
    wait for clk_period/2;
    GCLK <= '1';
    wait for clk_period/2;
end process;

--  TEST STRATEGY
--      GLOBAL RESET
--      Test 1: No faults test with two manual inputs.
--      Test 2: No faults test with memory stored vectors.
--      Test 3: Cycle stored vectors for each 3 faults.
test : process
begin
    wait for 100ns;
    wait until falling_edge(GCLK);

    -- GLOBAL RESET
    B_RST <= '0';
    B_TEST <= '0';
    B_F <= "00";
    INPUTS <= "000000";
    wait for clk_period*6;
    B_RST <= '1';
    wait for clk_period*6;
    B_RST <= '0';
    wait for clk_period*12;

    -- Test 1:
    --  Initialise B_F=00 (no fault) and test with 000000 inputs
    --  then 111111 inputs.
    B_F <= "00";
    B_TEST <= '0';
    INPUTS <= "000000";
    wait for clk_period*12;
    --  Reset
    B_RST <= '1';
    wait for clk_period*6;
    B_RST <= '0';
    wait for clk_period*12;

    B_F <= "00";
    B_TEST <= '0';
    INPUTS <= "111111";
    wait for clk_period*12;
    --  Reset
    B_RST <= '1';
    wait for clk_period*6;
    B_RST <= '0';
    wait for clk_period*12;

    -- Reset back to 0, not necessary but
    -- helps with reducing distractions
    -- when reading the other waveforms
    INPUTS <= "000000";
```

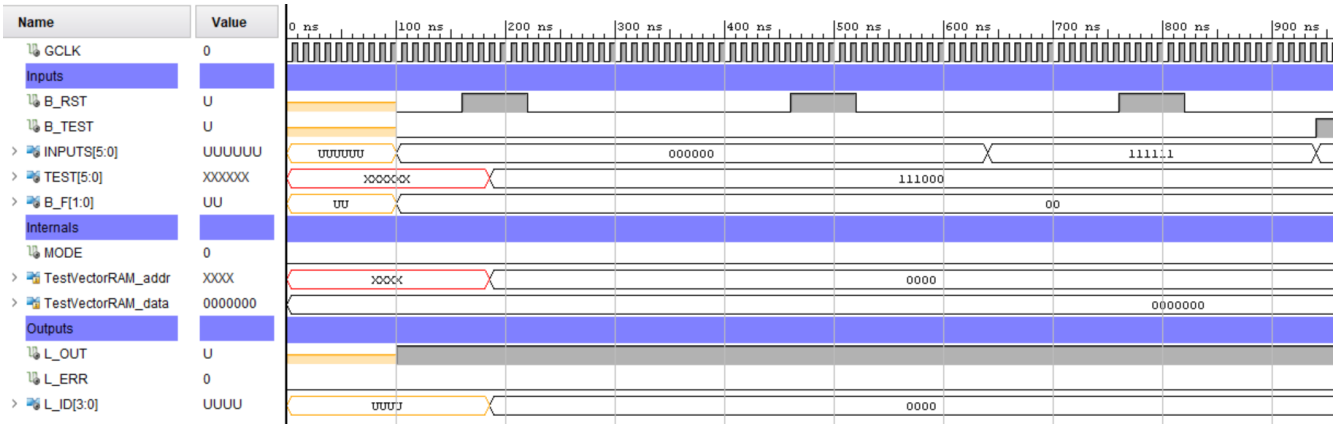
```
        -- Test 2:
        --  Initialise B_F=00 (no fault)
        B_F <= "00";
        B_TEST <= '1';
        wait for clk_period*6;
        B_TEST <= '0';
        wait for clk_period*12;
        --  Reset
        B_RST <= '1';
        wait for clk_period*6;
        B_RST <= '0';
        wait for clk_period*12;

        -- Test 3:
        --  Initialise B_F=01 (Es-a-1)
        B_F <= "01";
        B_TEST <= '1';
        wait for clk_period*6;
        B_TEST <= '0';
        wait for clk_period*12;
        --  Reset
        B_RST <= '1';
        wait for clk_period*6;
        B_RST <= '0';
        wait for clk_period*12;

        --  Initialise B_F=10 (Hs-a-0)
        B_F <= "10";
        B_TEST <= '1';
        wait for clk_period*6;
        B_TEST <= '0';
        wait for clk_period*12;
        --  Reset
        B_RST <= '1';
        wait for clk_period*6;
        B_RST <= '0';
        wait for clk_period*12;

        --  Initialise B_F=11 (Fs-a-0)
        B_F <= "11";
        B_TEST <= '1';
        wait for clk_period*6;
        B_TEST <= '0';
        wait for clk_period*12;
        --  Reset
        B_RST <= '1';
        wait for clk_period*6;
        B_RST <= '0';
        wait for clk_period*12;

        wait;
    end process;

end Behavioral;
```

3.2.3 Behavioural Simulation

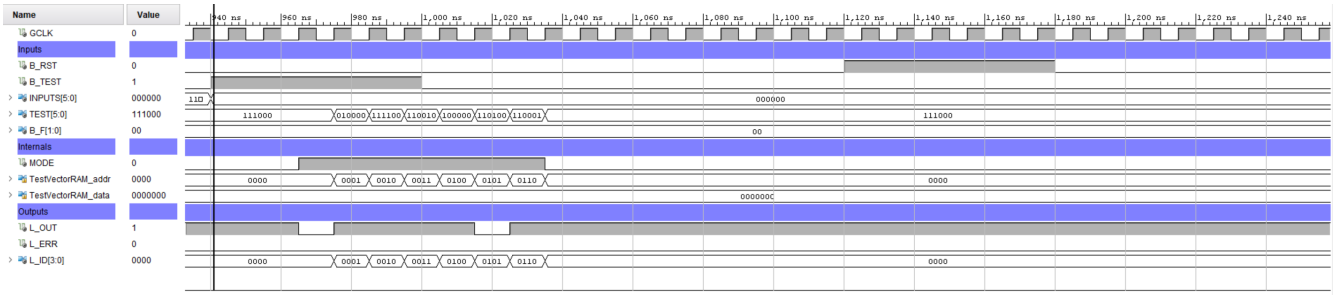
Waveform 1: Global Reset & Initialisation



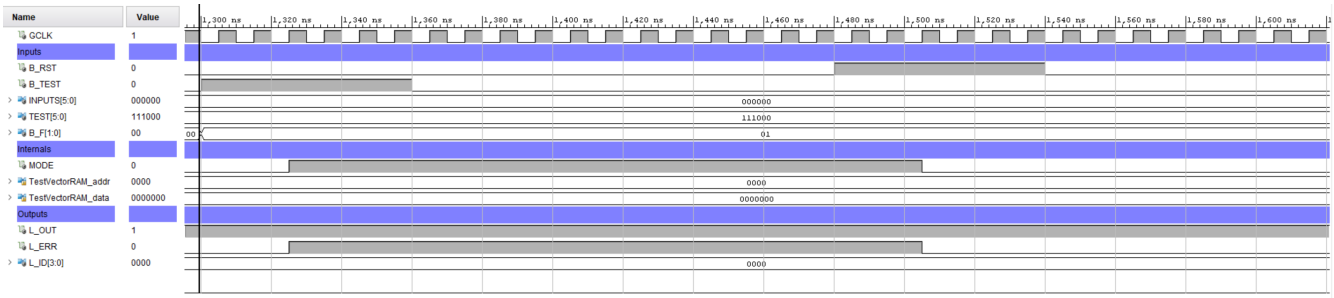
Waveform 2: Test 1



Waveform 3: Test 2

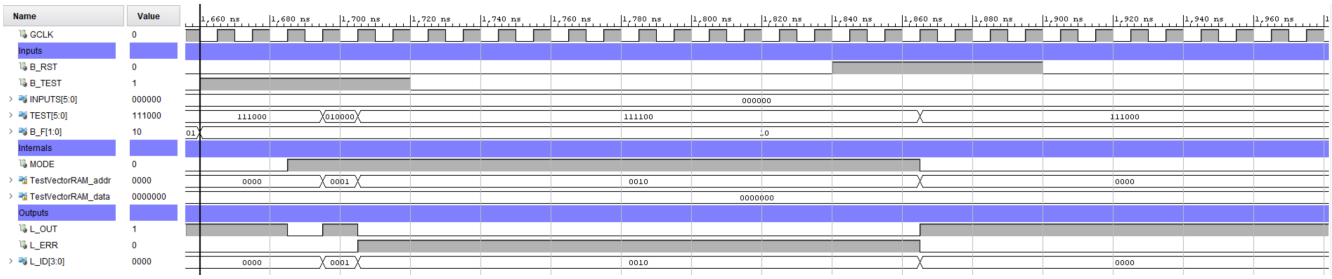


Waveform 4: Test 3



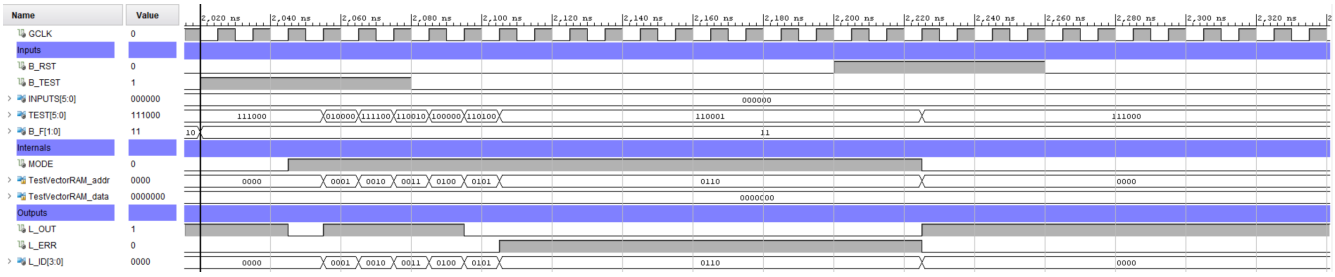
We can see 'L_ERR' turning high as the Es-a-1 fault is detected by the first test vector in memory, the one that detects the Ds-a-1 fault (111000). This verifies that the tests are correct as the table in section 3.1.6 shows Ds-a-1 can verify Es-a-1.

Waveform 5: Test 4



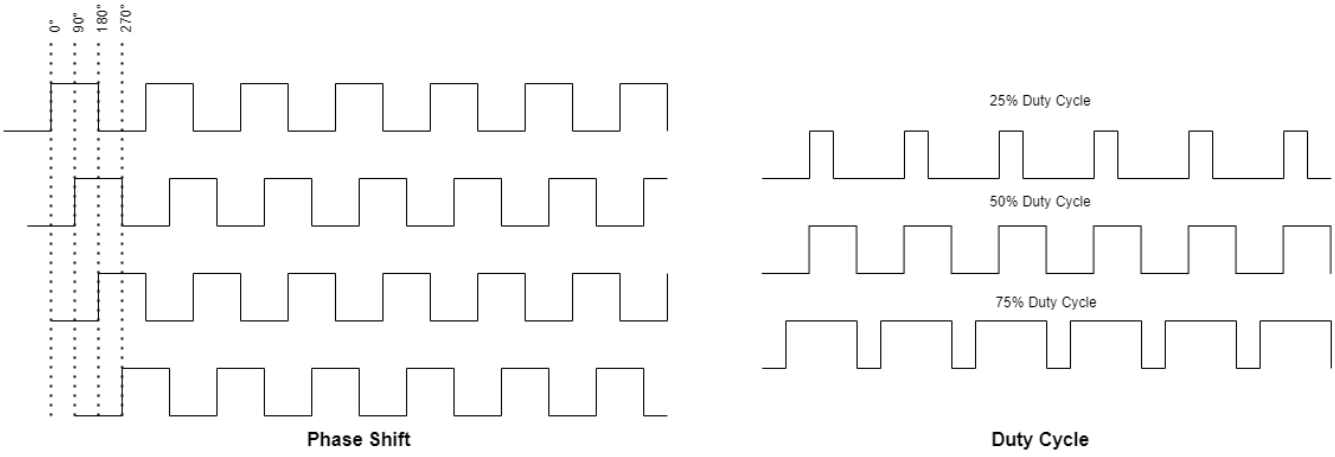
Here we can see 'L_ERR' going high as the Hs-a-0 fault is detected by the third test vector in memory, the one that detects the Cs-a-0 fault (111100). Again, this verifies that the tests are correct as the table in section 3.1.6 shows Cs-a-0 can verify the Hs-a-0 fault.

Waveform 5: Test 5



The signal 'L_ERR' goes high as the Fs-a-0 fault is detected by the last test vector in memory, the one that detects the Fs-a-0 fault (110001).

3.3.1 Phase Shift and Duty Cycle in a Clock Signal

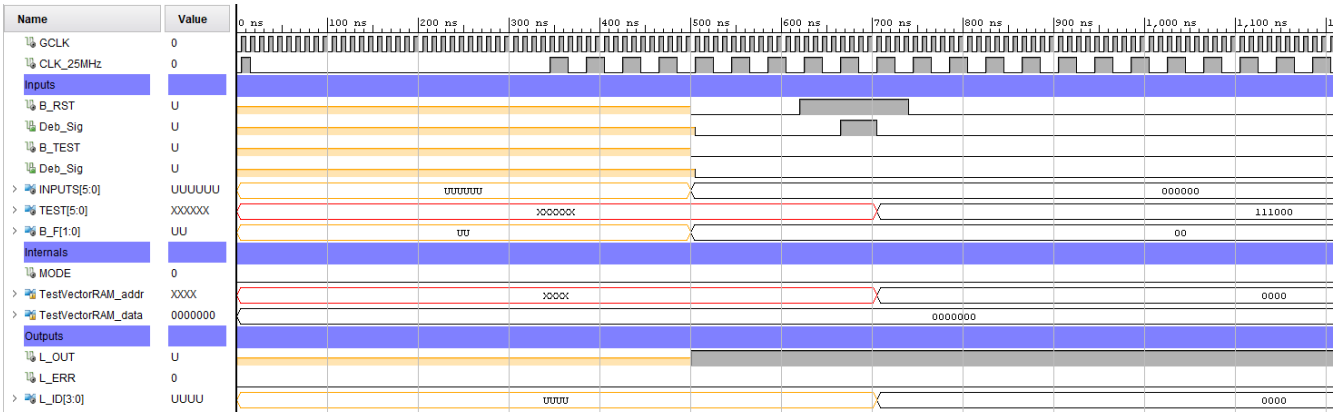


The phase shift of a clock signal refers to the displacement of the signal in time relative to some reference point such as a rising edge. For example, if we have a clock signal with a period of 1 microsecond and a phase shift of 90°, the clock signal will be shifted by 0.5 microseconds relative to the rising edge reference point. The diagram above shows phase shifts of 0°, 90°, 180° and 270°.

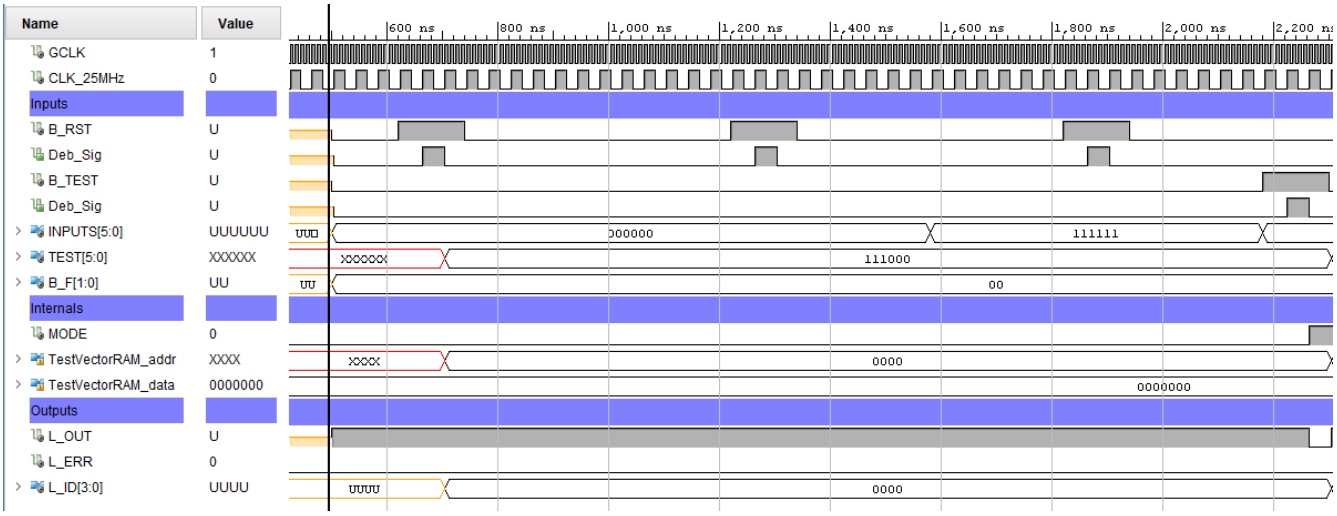
The duty cycle of a clock signal is the percentage of time that the signal is in the 'high' state. It is the ratio of the time that the signal spends in the 'high' state to the total period of the signal. For example, if we have a clock signal with a period of 1 microsecond and a duty cycle of 50%, the signal will be high for 0.5 microseconds and low for the remaining 0.5 microseconds. The diagram above shows duty cycles of 25%, 50% and 75%.

3.3.2 Behavioural Simulation

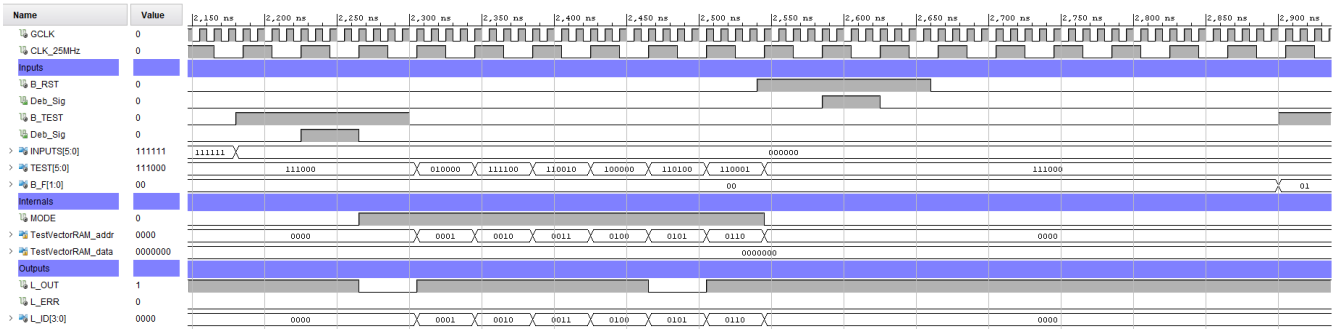
Waveform 1: Global Reset & Initialisation



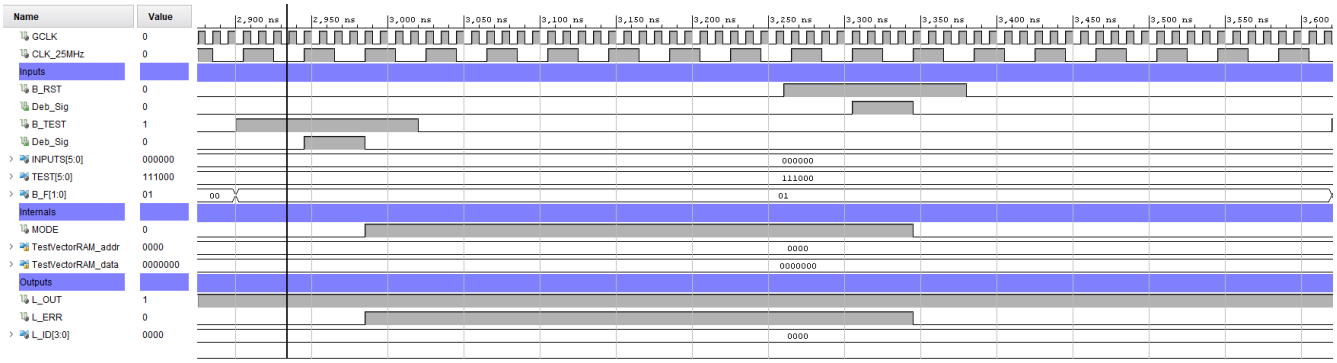
Waveform 2: Test 1



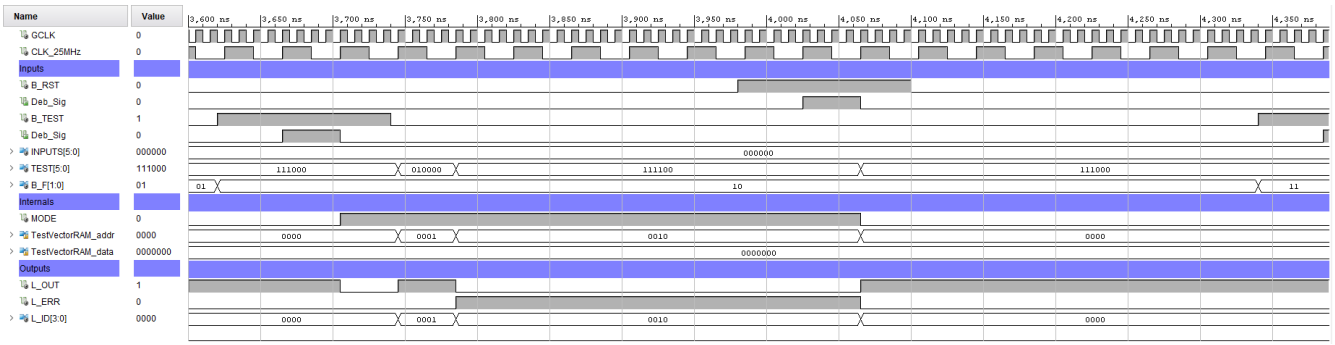
Waveform 3: Test 2



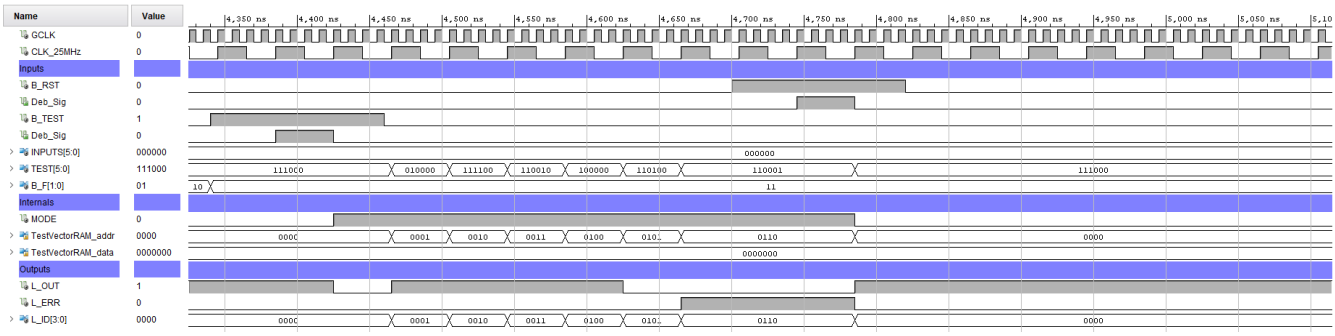
Waveform 4: Test 3



Waveform 5: Test 4

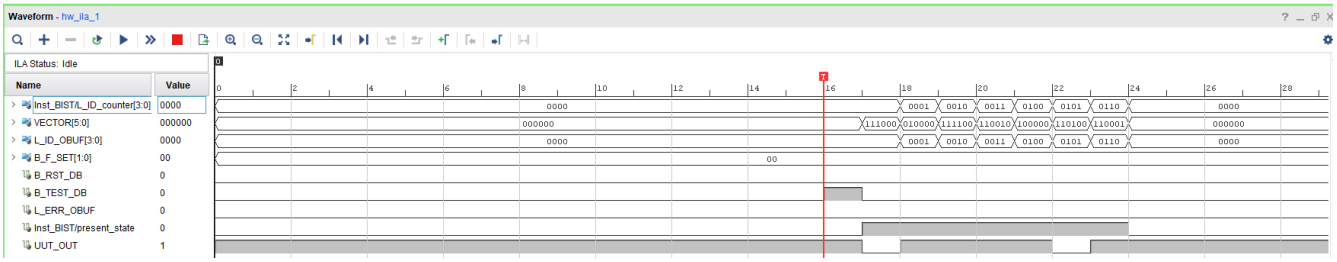


Waveform 5: Test 5

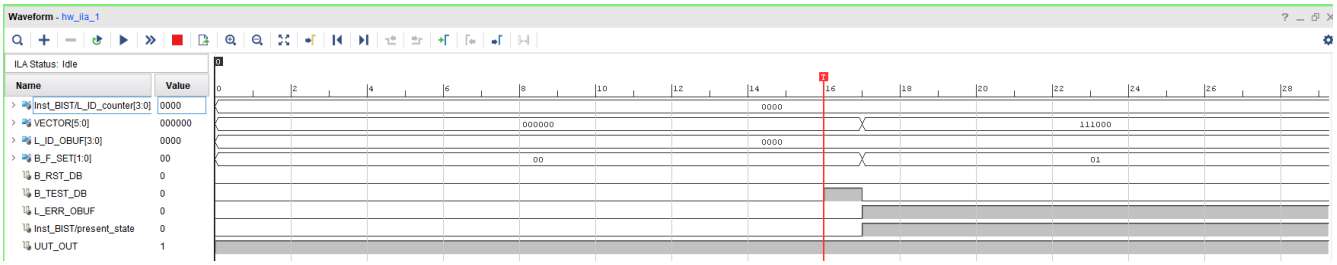


3.3.3 ILA Simulation

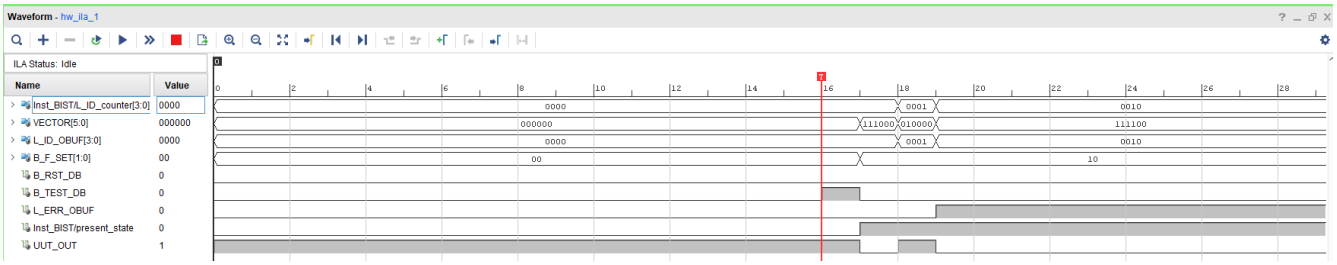
Waveform 1: Fault Free Cycle



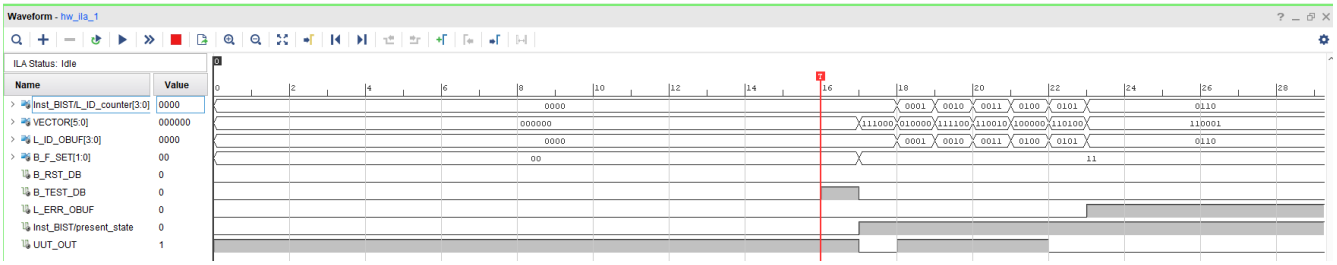
Waveform 2: Es-a-1 Fault



Waveform 3: Hs-a-0 Fault



Waveform 4: Fs-a-0 Fault



3.3.4 Nyquist Theorem and the ILA

The Nyquist theorem states that a signal must be sampled with a rate of at least twice the highest frequency of the signal. However, with an internal logic analyser (ILA), digital signals are sampled instead of analogue ones.

Digital signals are represented by discrete ‘high’ and ‘low’ voltage states, therefore, the required sampling rate is not dependent on the highest frequency of the signal, but instead is dependent on the rate of level change of the signals.