

Lab 5

Tasks:

a. Truth Table

D3	D2	D1	D0	PAR	EI4	EI3	EI2	EI1	EI0	ERROR	Q3	Q2	Q1	Q0
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	0	0	1	0	0	0	1
0	1	0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	1	0	1	0	0	1	0	0	1	0	0	0	1
0	0	0	1	1	0	0	0	1	0	1	0	0	0	1
1	1	1	1	0	0	0	0	0	1	1	0	0	0	1

b. Clock

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

```
entity Clock is port (
    CLK : inout std_logic := '0'
);
end Clock;
```

```
architecture dataflow of Clock is
begin
    CLK <= (not CLK) after 50 ns;
end dataflow;
```

c. Sender Register

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

```
entity Sender is port (
    CLK : in std_logic;
    D : in std_logic_vector(3 downto 0);
    Q : inout std_logic_vector(3 downto 0)
);
end Sender;
```

```
architecture dataflow of Sender is
begin
    Q <= D when rising_edge(CLK) else
```

```
    Q;  
end dataflow;
```

d. Receiver Register

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
entity Receiver is port (  
    CLK, EN : in std_logic;  
    D : in std_logic_vector(3 downto 0);  
    Q : out std_logic_vector(3 downto 0) := "0000"  
);  
end Receiver;  
  
architecture dataflow of Receiver is  
begin  
    Q <= "0001" when (EN = '1') else  
        D;  
end dataflow;
```

e. Error Injection

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
entity Error_Injector is port (  
    EI : in std_logic_vector(4 downto 0);  
    D : in std_logic_vector(3 downto 0);  
    PAR : in std_logic;  
    D_out : out std_logic_vector(3 downto 0);  
    PAR_out : out std_logic  
);  
end Error_Injector;  
  
architecture dataflow of Error_Injector is  
begin  
    PAR_out <= PAR xor EI(4);  
    D_out(3) <= EI(3) xor D(3);  
    D_out(2) <= EI(2) xor D(2);  
    D_out(1) <= EI(1) xor D(1);  
    D_out(0) <= EI(0) xor D(0);  
end dataflow;
```

f. Parity Generator

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Parity_Generator is port (
    D : in std_logic_vector(3 downto 0);
    PAR : out std_logic
);
end Parity_Generator;

architecture dataflow of Parity_Generator is
begin
    PAR <= D(3) xor D(2) xor D(1) xor D(0);
end dataflow;

```

g. Parity Checker

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Parity_Checker is port (
    D : in std_logic_vector(3 downto 0);
    PAR : in std_logic;
    ERR : out std_logic := '0'
);
end Parity_Checker;

architecture dataflow of Parity_Checker is
begin
    ERR <= PAR xor D(3) xor D(2) xor D(1) xor D(0);
end dataflow;

```

h. Structural Design

```

-----
-- Structural Design (Top Level)
-----

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Lab5 is port (
    D : in std_logic_vector(3 downto 0);
    CLK : inout std_logic;
    EI : in std_logic_vector(4 downto 0);
    Q : out std_logic_vector(3 downto 0);
    ERROR : inout std_logic
);
end Lab5;

```

architecture structural of Lab5 is

```
    signal Gen_PAR, Inj_PAR : std_logic;  
    signal Send_Q, Inj_D : std_logic_vector(3 downto 0);
```

component Clock is port (

```
    CLK : inout std_logic  
);
```

end component;

component Sender is port (

```
    CLK : in std_logic;  
    D : in std_logic_vector(3 downto 0);  
    Q : inout std_logic_vector(3 downto 0)  
);
```

end component;

component Receiver is port (

```
    CLK, EN : in std_logic;  
    D : in std_logic_vector(3 downto 0);  
    Q : out std_logic_vector(3 downto 0)  
);
```

end component;

component Error_Injector is port (

```
    EI : in std_logic_vector(4 downto 0);  
    D : in std_logic_vector(3 downto 0);  
    PAR : in std_logic;  
    D_out : out std_logic_vector(3 downto 0);  
    PAR_out : out std_logic  
);
```

end component;

component Parity_Generator is port (

```
    D : in std_logic_vector(3 downto 0);  
    PAR : out std_logic  
);
```

end component;

component Parity_Checker is port (

```
    D : in std_logic_vector(3 downto 0);  
    PAR : in std_logic;  
    ERR : out std_logic  
);
```

end component;

begin

```
    C1: Clock port map (CLK => CLK);  
    C2: Sender port map (CLK => CLK, D => D, Q => Send_Q);
```

```

    C3: Parity_Generator port map (D => Send_Q, PAR => Gen_PAR);
    C4: Error_Injector port map (EI => EI, D => Send_Q, PAR => Gen_PAR, D_out => Inj_D,
    PAR_out => Inj_PAR);
    C5: Parity_Checker port map (PAR => Inj_PAR, D => Inj_D, ERR => ERROR);
    C6: Receiver port map (CLK => CLK, D => Inj_D, EN => ERROR, Q => Q);
end structural;

```

i. Test Bench

```

library IEEE;
library STD;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;

```

```

ENTITY sim IS
END sim;

```

```

ARCHITECTURE behavioral OF sim IS
--Component Declaration for the Unit Under Test (UUT)
COMPONENT Lab5
PORT(
    D : in std_logic_vector(3 downto 0);
    CLK : inout std_logic;
    EI : in std_logic_vector(4 downto 0);
    Q : out std_logic_vector(3 downto 0);
    ERROR : inout std_logic
);
END COMPONENT;

```

```

-- Test bench stimuli signals to be generated
-- to drive the input signals of the design under test
-- in this case the DUT is the 3-input AND gate
signal D : std_logic_vector(3 downto 0);
signal EI : std_logic_vector(4 downto 0);
signal CLK : std_logic;
-- Test bench signals to represent the output
signal ERROR : std_logic;
signal Q : std_logic_vector(3 downto 0);

```

```

-- begin of test bench action
BEGIN
-- Instantiate the Unit Under Test (UUT)
-- and connect stimuli signals of the test bench
-- to the UUT pins.
 uut: Lab5 PORT MAP (
    D => D,
    CLK => CLK,
    EI => EI,

```

```

Q => Q,
ERROR => ERROR
);
-- Stimulus Process example
-- We use 100 ns pulse width for this example
process
begin
    Report "Test begins...";
-- Stimulus inputs
-----
D <= "1000"; EI <= "00000"; wait for 75 ns;
    Assert(D = "1000")
        Report "Test Failed on D-Vector: 1000";
    Assert(EI = "00000")
        Report "Test Failed on EI-Vector: 00000";
    Assert(ERROR = '0')
        Report "Test Failed on ERROR-Vector: 0";
    Assert(Q = "1000")
        Report "Test Failed on Q-Vector: 1000";
wait for 25 ns;
-----
D <= "0100"; EI <= "00000"; wait for 75 ns;
    Assert(D = "0100")
        Report "Test Failed on D-Vector: 0100";
    Assert(EI = "00000")
        Report "Test Failed on EI-Vector: 00000";
    Assert(ERROR = '0')
        Report "Test Failed on ERROR-Vector: 0";
    Assert(Q = "0100")
        Report "Test Failed on Q-Vector: 0100";
wait for 25 ns;
-----
D <= "0010"; EI <= "00000"; wait for 75 ns;
    Assert(D = "0010")
        Report "Test Failed on D-Vector: 0010";
    Assert(EI = "00000")
        Report "Test Failed on EI-Vector: 00000";
    Assert(ERROR = '0')
        Report "Test Failed on ERROR-Vector: 0";
    Assert(Q = "0010")
        Report "Test Failed on Q-Vector: 0010";
wait for 25 ns;
-----
D <= "0001"; EI <= "00000"; wait for 75 ns;
    Assert(D = "0001")
        Report "Test Failed on D-Vector: 0001";
    Assert(EI = "00000")
        Report "Test Failed on EI-Vector: 00000";

```

```
    Assert(ERROR = '0')
        Report "Test Failed on ERROR-Vector: 0";
    Assert(Q = "0001")
        Report "Test Failed on Q-Vector: 0001";
wait for 25 ns;
```

```
-----
D <= "1000"; EI <= "10000"; wait for 75 ns;
    Assert(D = "1000")
        Report "Test Failed on D-Vector: 1000";
    Assert(EI = "10000")
        Report "Test Failed on EI-Vector: 10000";
    Assert(ERROR = '1')
        Report "Test Failed on ERROR-Vector: 1";
    Assert(Q = "0001")
        Report "Test Failed on Q-Vector: 0001";
wait for 25 ns;
```

```
-----
D <= "0100"; EI <= "01000"; wait for 75 ns;
    Assert(D = "0100")
        Report "Test Failed on D-Vector: 0100";
    Assert(EI = "01000")
        Report "Test Failed on EI-Vector: 01000";
    Assert(ERROR = '1')
        Report "Test Failed on ERROR-Vector: 1";
    Assert(Q = "0001")
        Report "Test Failed on Q-Vector: 0001";
wait for 25 ns;
```

```
-----
D <= "0010"; EI <= "00100"; wait for 75 ns;
    Assert(D = "0010")
        Report "Test Failed on D-Vector: 0010";
    Assert(EI = "00100")
        Report "Test Failed on EI-Vector: 00100";
    Assert(ERROR = '1')
        Report "Test Failed on ERROR-Vector: 1";
    Assert(Q = "0001")
        Report "Test Failed on Q-Vector: 0001";
wait for 25 ns;
```

```
-----
D <= "0001"; EI <= "00010"; wait for 75 ns;
    Assert(D = "0001")
        Report "Test Failed on D-Vector: 0001";
    Assert(EI = "00010")
        Report "Test Failed on EI-Vector: 00010";
    Assert(ERROR = '1')
        Report "Test Failed on ERROR-Vector: 1";
    Assert(Q = "0001")
        Report "Test Failed on Q-Vector: 0001";
```

```

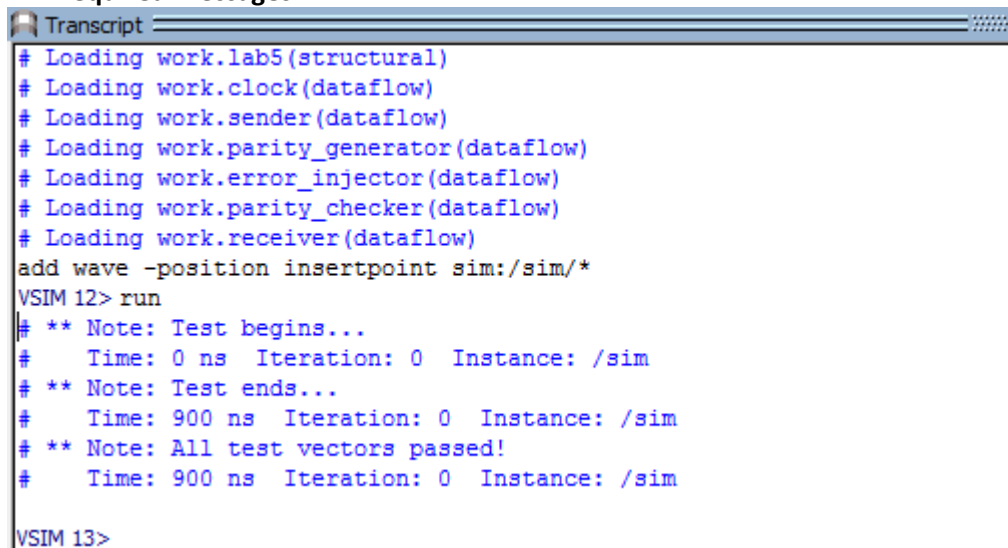
wait for 25 ns;
-----
D <= "1111"; EI <= "00001"; wait for 75 ns;
  Assert(D = "1111")
    Report "Test Failed on D-Vector: 1111";
  Assert(EI = "00001")
    Report "Test Failed on EI-Vector: 00001";
  Assert(ERROR = '1')
    Report "Test Failed on ERROR-Vector: 1";
  Assert(Q = "0001")
    Report "Test Failed on Q-Vector: 0001";
wait for 25 ns;
-----
  Report "Test ends...";
  Report "All test vectors passed!";
wait for 50 ns;
end process;

end;

```

j. Screenshots

i. Required Messages



```

Transcript
# Loading work.lab5 (structural)
# Loading work.clock (dataflow)
# Loading work.sender (dataflow)
# Loading work.parity_generator (dataflow)
# Loading work.error_injector (dataflow)
# Loading work.parity_checker (dataflow)
# Loading work.receiver (dataflow)
add wave -position insertpoint sim:/sim/*
VSIM 12> run
# ** Note: Test begins...
#   Time: 0 ns   Iteration: 0   Instance: /sim
# ** Note: Test ends...
#   Time: 900 ns Iteration: 0   Instance: /sim
# ** Note: All test vectors passed!
#   Time: 900 ns Iteration: 0   Instance: /sim
VSIM 13>

```

ii. Vector Simulation



Conclusions:

- a. I have a much better understanding of how the components work and the overall design structure. It was a good experience to be able to put all the knowledge we have learned into one bigger project. I also learned how to write reports and asserts for a test bench to inform the user if there were any discrepancies.
- b. Starting the project was a daunting task because it seems like a lot of work to finish and has no ideal starting point. After trying to build the components one at a time, it becomes easier when you start to see how they all come together.

Writing the test bench was a difficult task as well. There are not many examples in the book of how to write them and the initial process of connecting them with the components was difficult as well. But once you start figuring out where the components connect to, it makes the rest of the project much easier to complete.