1. Problem/Objective

For lab 4 the main objective was to learn and implement the Single Error Correction (SEC) technique. Our internal and external memory systems face the difficulties of having soft-error problems. The Single Error Correction algorithm can solve this by detecting the error and then correcting it. We used VHDL to code the SEC to simulate the output, and then project it onto the FPGA board. We were provided the shell code of the components and were to fill out the main parts.

2. Methodology

The paper chosen was "Secure Error Correction Using Multiparty Computation", by M. Raeini and M. Nojoumian. This paper is about how error correction techniques are prominent in their role in certain areas such as information theory, networking, and communication. They state that these techniques are used to detect and correct corrupt data over noisy channels. They want to propose the use of secure error correction by using secure multiparty computation. They find the importance of error correction use over private data.

The first solution was to use MPC protocols in which parties share private inputs by cryptographic primitives. This helps conjure a function without actually revealing the private inputs, the only thing being revealed is the function to all parties. They use the Berlekamp-Welch algorithm for their secure MPC protocol to locate errors. After the use of the Berlekamp-Welch function they use the cryptographic technique known as enrollment protocol to repair the errors.

3. Question(s)

Please, show your work. Don't just give a single value. Explain how you obtained that number.

a) Show how you calculated the check-bits K from data M in the

How we calculated the check-bits K from data M was the formula $2^k - 1 \geq M + K$. Since K = 3 didn't satisfy the equation, we tried K = 4, $2^4 - 1 \geq 8 + 4 = 16 \geq 12$.

b) Briefly describe what would happen when using the SEC algorithm if a check bit has an error rather than a data bit.

If we considered that the error was in check bit 8, then our stored word would become "001111001111". We change C8 from 0 to 1 to be considered an error. When we take data bit positions "0011, 01111, 1010, and 1111" and perform the XOR operation, it gives us a result of "1000", thus identifying that there is an error in position number "1000". Check bit C8 results in error.

4. Program Code

Copy your code here. Please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your code as text, not as a screenshot/image**.

SEC COMPARE:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity sec_compare is
	port(
		k, kp : in std_logic_vector(3 downto 0);
		syndrome : out std_logic_vector(3 downto 0);
		err : out std_logic);
end sec_compare;

architecture behavioral of sec_compare is
begin
	process(k, kp)
	begin
		-- TO DO: Determine if an error has occurred
		if (k = kp) then
			err <= '0';
		else
			err <= '1';
		end if;

	end process;
	-- TO DO: Determine the check bits
	syndrome <= k xor kp;

end behavioral;
```

SEC CORRECTOR:

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

entity sec_corrector is
	port(
		 m : in std_logic_vector(7 downto 0);
		 syndrome : in std_logic_vector(3 downto 0);
		 mp : out std_logic_vector(7 downto 0));
end sec_corrector;

architecture behavioral of sec_corrector is
begin
	process(m,syndrome)
	begin
		-- TO DO: If there is an error, correct the bit
		case syndrome is
			when "0000" =>
				mp <= m;
			when "0001" =>
				mp <= m xor "00000001";
			when "0010" =>
```

```
                    mp <= m xor "00000010";
                when "0011" =>
                    mp <= m xor "00000100";
                when "0100" =>
                    mp <= m xor "00001000";
                when "0101" =>
                    mp <= m xor "00010000";
                when "0110" =>
                    mp <= m xor "00100000";
                when "0111" =>
                    mp <= m xor "01000000";
                when "1000" =>
                    mp <= m xor "10000000";
                when others =>
                    NULL;
        end case;
    end process;

end behavioral;
```

SEC FUNC:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sec_func is
    port(
        m : in std_logic_vector(7 downto 0);
        k : out std_logic_vector(3 downto 0));
end sec_func;

architecture behavioral of sec_func is

begin
    -- TO DO: Create the function that obtains the k bits from the data-in m
    k(3) <= (m(7) xor m(6) xor m(5) xor m(4));
    k(2) <= (m(7) xor m(3) xor m(2) xor m(1));
    k(1) <= (m(6) xor m(5) xor m(3) xor m(2) xor m(0));
    k(0) <= (m(6) xor m(4) xor m(3) xor m(1) xor m(0));

end behavioral;
```

SEC:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity sec is
	port(
		data_in : in std_logic_vector(11 downto 0);
		 syndrome : out std_logic_vector(3 downto 0);
		 data_out : out std_logic_vector(7 downto 0);
		 err : out std_logic);
end sec;

architecture struct of sec is
	component sec_func
		port(
			m : in std_logic_vector(7 downto 0);
			  k : out std_logic_vector(3 downto 0));
	end component;

	component sec_compare
		port(
			k, kp : in std_logic_vector(3 downto 0);
			  syndrome : out std_logic_vector(3 downto 0);
			  err : out std_logic);
	end component;

	component sec_corrector
		port(
			m : in std_logic_vector(7 downto 0);
			syndrome : in std_logic_vector(3 downto 0);
			mp : out std_logic_vector(7 downto 0));
	end component;

	signal s_m: std_logic_vector(7 downto 0);
	signal s_mp: std_logic_vector(7 downto 0);
	signal s_k : std_logic_vector(3 downto 0);
	signal s_kp : std_logic_vector(3 downto 0);
	signal s_syndrome: std_logic_vector(3 downto 0);
	signal s_err: std_logic;

begin
	-- TO DO: Separate data-in to m and k
	s_m <= (data_in(11 downto 8) & data_in(6 downto 4) & data_in(2));
	s_k <= (data_in(7) & data_in(3) & data_in(1 downto 0));

	-- TO DO: Send m through f to get k'
	Map_Func: sec_func port map(
		m => s_m,
		k => s_kp);

	-- TO DO: Compare k and k', get error signal
```

```
    Map_Compare: sec_compare port map(
            k => s_k,
            kp => s_kp,
            syndrome => s_syndrome,
            err => s_err);

    -- TO DO: Correct data
    Map_Corrector: sec_corrector port map(
            m => s_m,
            syndrome => s_syndrome,
            mp => s_mp);

    -- TO DO: Set output signals
    syndrome <= s_syndrome;
    data_out <= s_mp;
    err <= s_err;


end struct;
```

5. Test Bench

Copy your test bench code here. Again, please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your test bench code as text, not as a screenshot/image**.

SEC FUNC TB:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY sec_func_func_tb_vhd_tb IS
END sec_func_func_tb_vhd_tb;

ARCHITECTURE behavior OF sec_func_func_tb_vhd_tb IS

    COMPONENT sec_func
    PORT(
            m : IN std_logic_vector(7 downto 0);
            k : OUT std_logic_vector(3 downto 0));
    END COMPONENT;

    SIGNAL m :  std_logic_vector(7 downto 0);
    SIGNAL k :  std_logic_vector(3 downto 0);

BEGIN

    uut: sec_func PORT MAP(
            m => m,
```

```
            k => k
    );


-- *** Test Bench - User Defined Section ***
  tb : PROCESS
  BEGIN
    m <= "00111001";
            wait for 20 ns;
            assert k = "0111" report "Incorrect check bits" severity error;
  END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

SEC TB:

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

entity sec_tb is
end sec_tb;

architecture behavior of sec_tb is

   -- component declaration for the unit under test (uut)

   component sec
   port(
      data_in : in  std_logic_vector(11 downto 0);
      syndrome : out  std_logic_vector(3 downto 0);
      data_out : out  std_logic_vector(7 downto 0);
      err : out  std_logic
     );
   end component;


  --inputs
  signal data_in : std_logic_vector(11 downto 0) := (others => '0');

     --outputs
  signal syndrome : std_logic_vector(3 downto 0);
  signal data_out : std_logic_vector(7 downto 0);
  signal err : std_logic;

begin

     -- instantiate the unit under test (uut)
  uut: sec port map (
```

```
        data_in => data_in,
        syndrome => syndrome,
        data_out => data_out,
        err => err
      );

  -- stimulus process
  stim_proc: process
  begin
            -- TO DO: Create the test bench here
            data_in <= "001101001111";
            wait for 20 ns;
            data_in <= "001101101111";
            wait for 20 ns;
            data_in <= "001101001110";
            wait for 20 ns;

            -- end
            assert false report "end of simulation" severity failure;
  end process;

end;
```

Output Waveform: