CALIFORNIA STATE UNIVERSITY, FULLERTON

# EGEC 447: INTRODUCTION TO CPS SECURITY

FALL 2023

PROGRAMMING ASSIGNMENT-1

HAMMING ENCODER AND DECODER

## Introduction:

1. What is the purpose of the hamming (7,4) code?

   The hamming 7,4 code is used to take a 4 bit input, and encrypt it into a 7 bit output with 3 additional parity bits that are used to decode and correct all one bit errors in the data. The hamming codes were developed by Richard W. Hamming to automatically correct errors introduced in punch-card readers in 1950.
2. How does each of these works?

   The hamming (7,4) code is a method of encoding and decoding data that can correct single bit errors and detect bit errors. It works by adding three parity bits to the four bit data word forming a seven bit codeword. The parity bits are calculated based on the position of the data bits. Using this scheme, the decoder identifies and corrects any single bit error by comparing the received parity bits with the expected ones. If two bits are flipped, the error can be detected and not corrected since there will be two possible original codewords that differ by two bits.

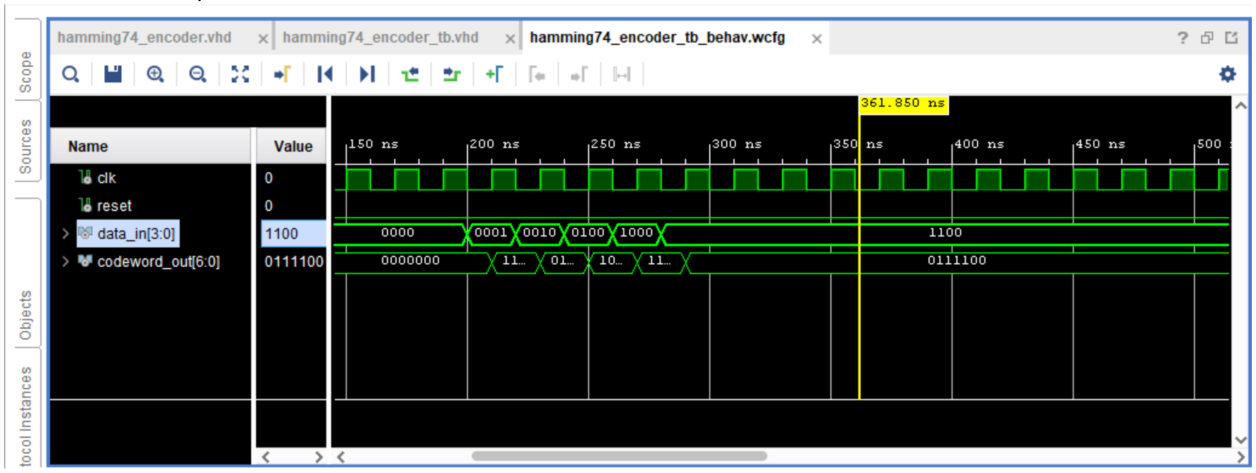
## Procedure

**Encoder:**

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


entity hamming74_encoder is
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          data_in : in STD_LOGIC_VECTOR (3 downto 0);
          codeword_out : out STD_LOGIC_VECTOR (6 downto 0));
end hamming74_encoder;

architecture Behavioral of hamming74_encoder is
   signal input_reg : std_logic_vector (3 downto 0 );
begin
   process(clk, reset)
   begin
      if rising_edge(clk) then
         if reset = '1' then
            codeword_out <= (others => '0');
         elsif reset = '0' then
            input_reg <= data_in;
            codeword_out(6) <= input_reg(3) xor input_reg(2) xor input_reg(0);--P1
            codeword_out(5) <= input_reg(3) xor input_reg(1) xor input_reg(0);--P2
            codeword_out(4) <= input_reg(3);--D1
            codeword_out(3) <= input_reg(2) xor input_reg(1) xor input_reg(0);--P3
            codeword_out(2) <= input_reg(2);--D2
            codeword_out(1) <= input_reg(1);--D3
            codeword_out(0) <= input_reg(0);--D4
          end if;
      end if;
    end process;
end Behavioral;
```

**The Encoder testbench:**

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

entity hamming74_encoder_tb is
end hamming74_encoder_tb;

architecture Behavioral of hamming74_encoder_tb is
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal data_in : std_logic_vector(3 downto 0) := (others => '0');
    signal codeword_out : std_logic_vector(6 downto 0);

     component hamming74_encoder is
        Port ( clk : in STD_LOGIC;
               reset : in STD_LOGIC;
               data_in : in STD_LOGIC_VECTOR (3 downto 0);
               codeword_out : out STD_LOGIC_VECTOR (6 downto 0));
    end component;

begin
    uut: hamming74_encoder port map (
        clk => clk,
        reset => reset,
        data_in => data_in,
        codeword_out => codeword_out
    );

    clk_process :process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    end process;

    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        wait for 100 ns;

        reset <= '0';
        wait for 100 ns;
        -- insert stimulus here
        data_in <= "0001";
        wait for 20 ns;

        data_in <= "0010";
        wait for 20 ns;

        data_in <= "0100";
        wait for 20 ns;

        data_in <= "1000";
        wait for 20 ns;

        data_in <= "1100";
        wait for 20 ns;

        wait;
    end process;

end Behavioral;
```



**Explanation:**
The encoder is designed to take a 4-bit input every rising clock edge. The encoder then calculates the parity bits using logic gates and then produces a 7-bit encoded output with the parity bits calculated and inserted at the right places.

**Decoder:**

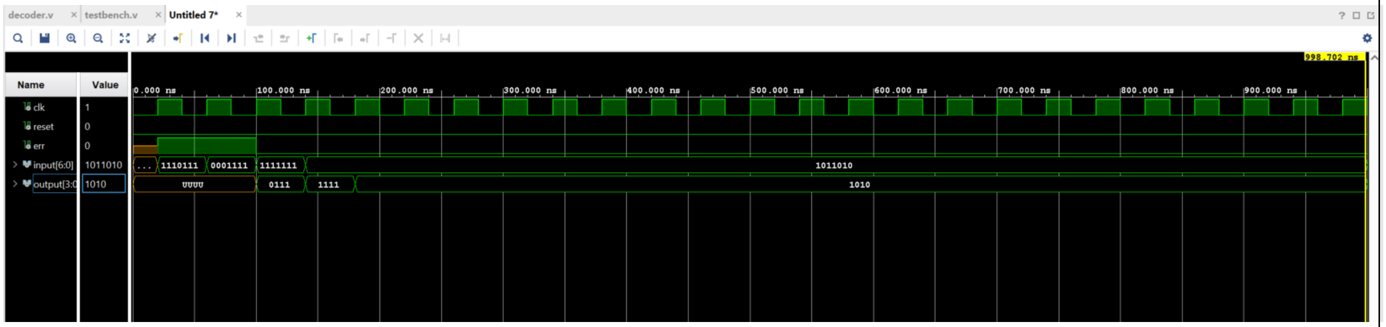
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity hamming_decoder is
   Port (
      input : in  STD_LOGIC_VECTOR (6 downto 0);
      output : out  STD_LOGIC_VECTOR (3 downto 0);
      clk: in std_logic;
      err: out std_logic;
      reset: in std_logic
   );
end hamming_decoder;

architecture Behavioral of hamming_decoder is
      signal input_reg : std_logic_vector(6 downto 0);

begin
   process(clk,reset)
      variable C : std_logic_vector(2 downto 0);
      variable corrected_output : std_logic_vector(3 downto 0);
      variable err_store : std_logic;
   begin
      if (rising_edge(clk)) then
         if reset = '0' then
            input_reg <= input;
            err_store := '0';
            C(0) := input_reg(6) xor input_reg(4) xor input_reg(2) xor input_reg(0) ;
            C(1) := input_reg(5) xor input_reg(4) xor input_reg(1) xor input_reg(0) ;
            C(2) := input_reg(3) xor input_reg(2) xor input_reg(1) xor input_reg(0) ;


            case C is
               when "000" =>
                  corrected_output := input_reg(4) & input_reg(2) & input_reg(1) & input_reg(0);
                  err_store := '0';
               when "001" =>
```

```
                    corrected_output := input_reg(4) & input_reg(2) & input_reg(1) & input_reg(0);
                    err_store := '0';
                  when "010" =>
                    corrected_output := input_reg(4) & input_reg(2) & input_reg(1) & input_reg(0);
                    err_store := '0';
                  when "011" =>
                    corrected_output := not input_reg(4) & input_reg(2) & input_reg(1) &
input_reg(0);
                    err_store := '1';
                  when "100" =>
                    corrected_output := input_reg(4) & input_reg(2) & input_reg(1) & input_reg(0);
                    err_store := '0';
                  when "101" =>
                    corrected_output := input_reg(4) & not input_reg(2) & input_reg(1) &
input_reg(0);
                    err_store := '1';
                  when "110" =>
                    corrected_output := input_reg(4) & input_reg(2) & not input_reg(1) &
input_reg(0);
                    err_store := '1';
                  when "111" =>
                    corrected_output := input_reg(4) & input_reg(2) & input_reg(1) & not
input_reg(0);
                    err_store := '1';
                  when others =>
                    -- Handle unexpected cases
                    corrected_output := "UUUU";
                    err_store := '1';
                end case;

                output <= corrected_output;
                err <= err_store;
              elsif (reset = '1') then
                output <= "0000";
                err <= '0';
              end if;
  end if;
end process;
end architecture Behavioral;
```

**The decoder testbench:**

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

entity hamming74_encoder_tb is
end hamming74_encoder_tb;

architecture Behavioral of hamming74_encoder_tb is
  signal clk : std_logic := '0';
  signal reset : std_logic;
  signal err : std_logic := '0';
  signal input: std_logic_vector(6 downto 0);
  signal output: std_logic_vector(3 downto 0);


  component hamming_decoder is
  Port(
    input: in std_logic_vector(6 downto 0);
    output: out std_logic_vector(3 downto 0);
    clk: in std_logic;
    err: out std_logic;
    reset: in std_logic
  );
end component;

begin
    decoder_inst : hamming_decoder
    port map (
      input => input,
      output => output,
      clk => clk,
      err => err,
      reset => reset
  );

  clk_process : process
  begin
      clk <= '0';
      wait for 20 ns;
      clk <= '1';
      wait for 20 ns;
  end process;
```

```
    uut: process
    begin
        reset <= '0';
        wait until rising_edge(clk);
        input <= "1110111";
        wait for 40 ns;
        input <= "0001111";
        wait for 40 ns;
        input <= "1111111";
        wait for 40 ns;
        input <= "1011010";

        wait;
    end process;

end Behavioral;
```



**Explanation:** The decoder takes a 7-bit encoded input along with an error signal, clock and reset. At every rising clock edge, a 7-bit codeword is decoded. The decoding process first stores the input parity bits in a variable. This variable is used to calculate the parity matrix. A case statement compares the syndrome bits and depending on their value it corrects all one-bit errors using the hamming (7,4) calculations using Xor and not gates. Once these errors have been corrected the final output is produced, along with an error signal that signifies if any incorrect bits were found and corrected. This way the decoder module is successfully able to error check and decode the input to a 4-bit output. As we can see in the above screenshot we are able to decode successfully and use the error signal to detect the presence of any errors in the databits, the error signal stays low for parity bit errors.

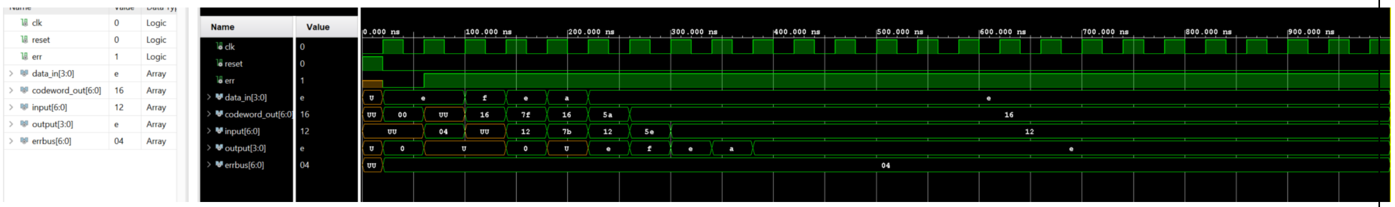
**The Testbench/Final Instantiation:**

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

entity hamming74_encoder_tb is
end hamming74_encoder_tb;

architecture Behavioral of hamming74_encoder_tb is
  signal clk : std_logic := '0';
  signal reset : std_logic;
  signal err : std_logic := '0';
  signal data_in: std_logic_vector(3 downto 0);
  signal codeword_out: std_logic_vector(6 downto 0);
  signal input: std_logic_vector(6 downto 0);
  signal output: std_logic_vector(3 downto 0);
  signal errbus: std_logic_vector(6 downto 0);


  component hamming74_encoder is
    Port(
      clk: in std_logic;
      reset: in std_logic;
      data_in: in std_logic_vector(3 downto 0);
      codeword_out: out std_logic_vector(6 downto 0)
    );
  end component;
  component hamming_decoder is
  Port(
    input: in std_logic_vector(6 downto 0);
    output: out std_logic_vector(3 downto 0);
    clk: in std_logic;
    err: out std_logic;
    reset: in std_logic
  );
end component;

begin
  encoder_inst : hamming74_encoder
    port map (
      clk => clk,
      reset => reset,
      data_in => data_in,
      codeword_out => codeword_out
    );
    decoder_inst : hamming_decoder
    port map (
        input => input,
        output => output,
        clk => clk,
        err => err,
        reset => reset
);

-- Clock
clk_process : process
begin
        clk <= '0';
        wait for 20 ns;
        clk <= '1';
        wait for 20 ns;
end process;

err_bus: process(clk, reset)
begin
    if (rising_edge(clk)) then
        if reset = '1' then
            errbus <= "0000000";
        elsif reset ='0' then
            input <= codeword_out xor errbus;
        end if;
    end if;
end process;



stimu: process
begin
    reset <= '1';
    wait until rising_edge(clk);
    reset <= '0';
    errbus <= "0000000";
    data_in <= "1110";

    wait for 80 ns;
    data_in <= "1111";

    wait for 40 ns;
    data_in <= "1110";
```

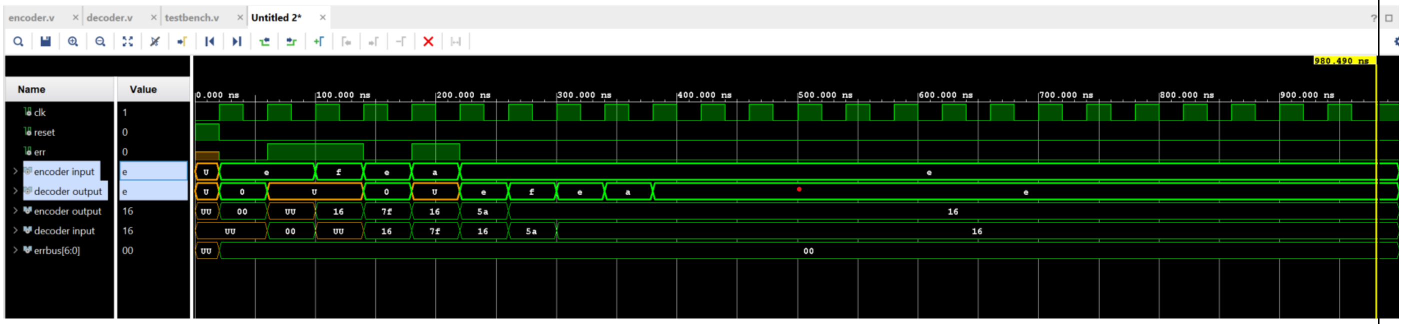
```
        wait for 40 ns;
        data_in <= "1010";
        wait for 40 ns;
        data_in <= "1110";

        wait;
    end process;

end Behavioral;
```

**Explanation:** the final test bench connects the encoder and decoder modules with an error bus in between the output of the encoder and input of the decoder. The encoder output is xor-ed with the error bus to inject 1-bit errors. The functionality of the system is tested. We test multiple inputs and see that the encoder and decoder is functioning as expected. The simulation graph shows that it takes 3-cycles for an input codeword to be encoded, error injected and decoded. Hence, we are able to verify from the waveform that the input data and output data corelate as expected.

The results of the complete system with a bit error injected in the error bus:



The system when testing codewords with no error injected:



**The system is thus able to successfully encode, decode and correct all 1-bit errors injected by the error bus. The renamed labels on the above graph show the encoder input, encoder output, decoder input /output, error bus and error signal.**

**Conclusion:**

1. What did you learn or discover as you completed this assignment?

   We were able to successfully design a system that is capable of encoding a 4-bit message before sending it to the decoder with/without injected error to verify the functionality of the decoder. This helped strengthen our RTL logic design skills. We were also able to gain a better understanding of Hamming code and its process of encoding, decoding and single bit error correction. The project also helped us get a better understanding of timing the input and output registers of the encoding and decoding components such that the system is running asynchronously.
2. If you were to redo this assignment, what would you do differently?

   If We were to redo the assignment, we would try to minimize the delay between encoding and decoding so that the system is better synchronized to providing faster outputs at the rising clock edge. A top module that instantiates both components together with XOR gates between both components would be able to improve the overall design. We would also consider increasing the input and output capability such that we can encode/decode large messages with error correction.

## References

1. https://www.youtube.com/watch?v=2BI7wvmdFE8&t=1s
2. https://en.wikipedia.org/wiki/Hamming(7,4)
3. https://faculty-web.msoe.edu/johnsontimoj/EE3921/files3921/Book_FreeRangeVHDL.pdf

**EGCP 447 Programming assignment-1**
**Teamwork Contribution Sheet**

| | |
|---|---|
| I understand that providing false information could result in me failing the course.<br>Preparer's Signature (sign the initial): | |
| Team Members' Name | Kaavya varshitha raman shantha |
| Description of Job | Designing, Simulating and creating the encoder, verifying the encoder output with the correct hamming encoding format |
| Team Members' Name | Ashwin Koshy John |
| Description of Job | Designing Simulating and creating the decoder, verifying the decoder output with the correct hamming decoding format and error correction |
| Team Members' Name | Reddy Sowmya Vangumalla |
| Description of Job | Designing and simulating a testbench that connects the encoder and decoder with an error correcting bus that injects an error to verify the output. |