CPE166 Lab 3 Part 1
By: Prof. Pang

Lab 3

Part 1: (7, 4) Hamming Code Generator

The key of the Hamming Code is the use of extra parity bits to allow the correction of a single bit error. **Hamming** (**7**,**4**) is a linear error-correcting **code** that encodes four bits of data into seven bits by adding three even parity bits shown in figure 1. This project is to design (7, 4) Hamming Code generator in VHDL.

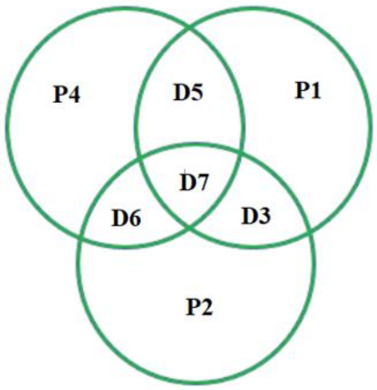

Figure 1. (7, 4) hamming code diagrams

In the above figure, D7, D6, D5 and D3 are input data,

- P4 is the even parity bit of binary data D7, D6 and D5;
- P2 is the even parity bit of binary data D7, D6 and D3;
- P1 is the even parity bit of binary data D7, D5 and D3.

The interface of this design is shown in Table 1.

Table 1. (7, 4) hamming code interface signals

Port Names	Port Direction	Port Size
D7	Input	1 bit
D6	Input	1 bit
D5	Input	1 bit
D3	Input	1 bit
DOUT	Output	7 bits (7 downto 1)

In the final design, D7, D6, D5, D3 are input data, DOUT is the final 7-bit hamming code.

Demo Requirement

You need to demonstrate the final simulation waveform of this design to your lab instructor.

CPE166 Lab 3 Part 1
By: Prof. Pang

Lab Procedure

Step 1. Parity Function

The even parity bit is the parity bit added to the data block. It is used to ensure that the total number of bits (including parity bits) in the message is even for error detection.

For this step, you will practice and review again after learning VHDL functions and program packages in class.

Simplified Sample Syntax

```
function function_name (parameter_list) return type is
        declarations
begin
        sequential statements
end function_name;
```

The following MY_PACK.vhd is used to create a package that can be used. Therefore, in your project, you need to add this source file first.

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;

package MY_PACK is

 function PARITY (D : std_logic_vector)       -- declaration of function

 return std_logic;

end MY_PACK;

package body MY_PACK is

function PARITY (D : std_logic_vector)       -- implementation of function inside the package body

 return std_logic is

 variable TMP : std_logic;

 begin

      TMP := D(0);

      for J in 1 to D'high loop

          TMP := TMP xor D(J);

       end loop;     -- works for any size of D

 return TMP;

 end PARITY;      -- even parity

end MY_PACK;
```

Step 2. Even Parity Bit of 3-bit Input Data

In the following PAR.vhd design, the "use work.MY_PACK.all" statement includes the MY_PACK package. As a result, you can directly call the PARITY function in the PAR.vhd design to obtain the even parity bit value of the three-bit input data.

```
library ieee;

use ieee.std_logic_1164.all;

use work.MY_PACK.all;

entity PAR is

 port(  db: in std_logic_vector(2 downto 0);

           pb: out std_logic);

end PAR;

architecture ARCH of PAR is

begin

        pb <= PARITY(db);

 end ARCH;
```

After including the above two VHDL files into your project, you need to write a testbench file for PAR design. Then, you can run simulations to verify the generation of the even parity signal of the three-bit test data in VHDL.

Step 3. (7, 4) Hamming Code Generator

(7,4) hamming code encodes four bits of data into seven bits by adding three even parity bits. The contents below are for you to review what you have learned in class.

For the four binary data bits D7 D6 D5 D3, use three even parity bits P4 P2 P1 according to the following diagram.

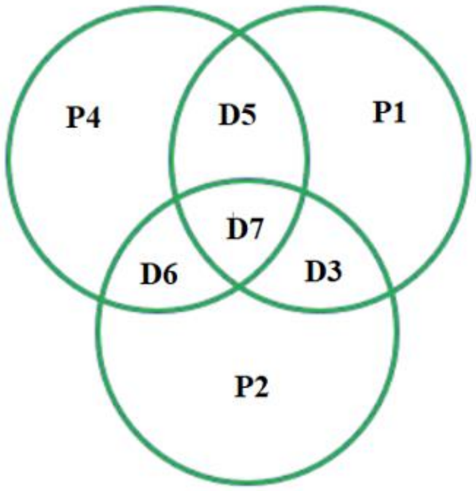

Figure 1. (7, 4) hamming code diagrams

In the above figure,

- P4 is the even parity bit of binary data D7, D6 and D5;
- P2 is the even parity bit of binary data D7, D6 and D3;
- P1 is the even parity bit of binary data D7, D5 and D3.

For the 4-bit input data D7 D6 D5 D3, the final (7, 4) hamming code will be

- D7, D6, D5, P4, D3, P2, P1

Now, it is time to design the hamming.vhd with the following interface signals according to the above

(7, 4) Hamming code generation scheme.

Table 1. (7, 4) hamming code interface signals

Port Names	Port Direction	Port Size
D7	Input	1 bit
D6	Input	1 bit
D5	Input	1 bit
D3	Input	1 bit
DOUT	Output	7 bits (7 downto 1)

In the hamming.vhd design, D7, D6, D5, D3 are input data, DOUT is the final 7-bit hamming code.

DOUT is the concatenation of the following data bits:

- D7, D6, D5, P4, D3, P2, P1
  (bit7, bit6, bit5, bit 4, bit3, bit2, bit 1 of DOUT)

P4, P2, P1 are the internal parity signals used inside hmming.vhd.

In this design, you can use the PAR component in step 1, and create three PAR instances to generate P4, P2 and P1.

The block diagram of the hamming.vhd is shown below.

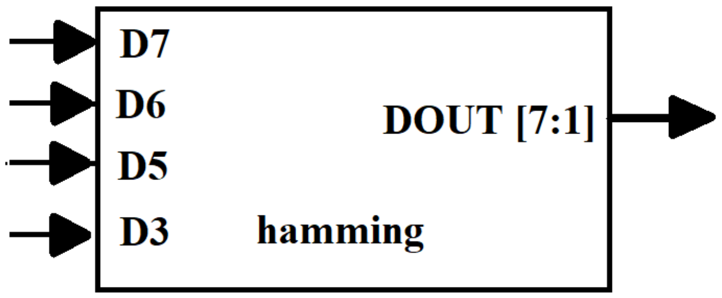

Figure 2. (7, 4) hamming code generator design block diagram

After completing hamming.vhd, you need to write testbench to run simulations.

CPE166 Lab 3 Part 1
By: Prof. Pang

Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample VHDL Codes:

(1). Concatenation Operator (&)

```
architecture arch of  one_design is

signal    a:   std_logic_vector ( 3 downto 0);

signal   b, c, d, e:  std_logic;

begin

       a <= b & c & d & e;  -- Concatenate four single bit signals to create one 4-bit signal

      …

end arch;
```

(2). cir.vhd   and cir_tb.vhd

```
library ieee;

use ieee.std_logic_1164.all;

entity cir is

port(     a: in std_logic;

             b: in std_logic;

            c: in std_logic;

             f: out std_logic

);

end cir;

architecture dataflow of cir is

begin

   f <= ( a and b ) or c;

end dataflow;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;


entity tb_cir is

end tb_cir;


architecture tb of tb_cir is


   component cir

      port (a : in std_logic;

           b : in std_logic;

           c : in std_logic;

           f : out std_logic);

   end component;


   signal a : std_logic;

   signal b : std_logic;

   signal c : std_logic;

   signal f : std_logic;

begin

   dut : cir  port map (

            a => a,

           b => b,

           c => c,

           f => f   );

   stimuli : process

   begin

      a <= '0';
```

```
        b <= '0';

        c <= '0';

        wait for 10 ns;

        a <= '0';

        b <= '1';

        c <= '0';

        wait for 10 ns;

        a <= '1';

        b <= '1';

        c <= '0';

        wait for 10 ns;

        a <= '0';

        b <= '1';

        c <= '1';

        wait for 10 ns;

        a <= '1';

        b <= '1';

        c <= '1';

        wait for 10 ns;

        a <= '0';

        b <= '0';

        c <= '0';

        wait for 10 ns;

        wait;

    end process;

end tb;
```

(3). example.vhd and example_tb.vhd

```
library ieee ;

use ieee.std_logic_1164.all ;


entity example is

  port( a: in std_logic;

        b,c: in std_logic_vector (1 downto 0);

        f:  out std_logic_vector (1 downto 0)

  );

end example;


architecture gates of example is

signal  m, n:  std_logic_vector(1 downto 0);

begin

    m <=  (a & a ) and b;

   -- AND gate comment: m(0) <= a & b(0);

   -- AND gate comment: m(1) <= a & b(1);


    n <=  (a & a ) and c;

    f <=  m or n;

end gates;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;


entity tb_example is

end tb_example;


architecture tb of tb_example is

   component example

       port (a : in std_logic;

            b : in std_logic_vector (1 downto 0);

            c : in std_logic_vector (1 downto 0);

            f : out std_logic_vector (1 downto 0));

   end component;

   signal a : std_logic;

   signal b : std_logic_vector (1 downto 0);

   signal c : std_logic_vector (1 downto 0);

   signal f : std_logic_vector (1 downto 0);

begin

   dut : example

   port map (a => a,

            b => b,

            c => c,

            f => f);

   a <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;

   b <= "00", "01" after 5 ns, "10" after 10 ns, "11" after 20 ns, "10" after 30 ns;

   c <= "11", "10" after 10 ns, "01" after 30 ns, "11" after 40 ns;

end tb;
```