給GPT敘述

利用i_left_button 和i_right_button 按鈕實現對打並顯示分數
分數分為left_score跟right_score
狀態 :reserve(進入預備狀態) ,counter_is_counting_left(i_left_button 有打到LED左移) ,
counter_is_counting_right(i_right_button 有打到counter右移),
left_win(counter_is_counting_right狀態i_right_button提早按或是沒打到左贏顯示分數),
right_win(counter_is_counting_left狀態i_left_button提早按或是沒打到右贏顯示分數),
left_ready_serve(i_left_button 按下預備狀態改為左邊預備發球再按一次i_left_button 狀態變成counter_is_counting_right),
right_ready_serve(i_right_button 按下預備狀態改為右邊預備發球再按一次i_right_button 狀態變成counter_is_counting_left)

最初始的程式碼

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

entity ping_pong_game is
    port (
        i_clk         : in std_logic;
        i_rst         : in std_logic;
        i_left_button  : in std_logic;
        i_right_button : in std_logic;
        left_score     : out std_logic_vector(7 downto 0);
        right_score    : out std_logic_vector(7 downto 0);
        o_led         : out std_logic_vector(7 downto 0)
    );
end ping_pong_game;

architecture behavioral of ping_pong_game is
    type state_type is (reserve, counter_is_counting_left, counter_is_counting_right,
                        left_win, right_win, left_ready_serve, right_ready_serve);
    signal state          : state_type;
    signal led_position   : std_logic_vector(7 downto 0) := "00000001"; -- LED初始位置
    signal left_score_reg : unsigned(7 downto 0) := (others => '0');
    signal right_score_reg : unsigned(7 downto 0) := (others => '0');
begin
```

```
-- 输出当前的分数
left_score <= std_logic_vector(left_score_reg);
right_score <= std_logic_vector(right_score_reg);
o_led <= led_position;

process(i_clk, i_rst)
begin
    if i_rst = '0' then
        state <= reserve;
        left_score_reg <= (others => '0');
        right_score_reg <= (others => '0');
        led_position <= "00000001";
    elsif rising_edge(i_clk) then
        case state is
            when reserve =>
                if i_left_button = '1' then
                    state <= left_ready_serve;
                elsif i_right_button = '1' then
                    state <= right_ready_serve;
                end if;

            when left_ready_serve =>
                if i_left_button = '1' then
                    state <= counter_is_counting_right;
                    led_position <= "10000000"; -- 右边开始
                end if;

            when right_ready_serve =>
                if i_right_button = '1' then
                    state <= counter_is_counting_left;
                    led_position <= "00000001"; -- 左边开始
                end if;

            when counter_is_counting_left =>
                if i_left_button = '1' then
                    if led_position = "00000001" then
                        state <= counter_is_counting_right;
                        led_position <= led_position srl 1; -- 左移
                    else
                        state <= right_win; -- 提早按
                    end if;
                elsif led_position = "10000000" then -- LED已经右移到最右
                    state <= right_win;
                else
                    led_position <= led_position sll 1; -- 右移
                end if;

            when counter_is_counting_right =>
                if i_right_button = '1' then
                    if led_position = "10000000" then
                        state <= counter_is_counting_left;
                        led_position <= led_position sll 1; -- 右移
                    else
                        state <= left_win; -- 提早按
```

```
                end if;
            elsif led_position = "00000001" then -- LED已经左移到最左
                state <= left_win;
            else
                led_position <= led_position srl 1; -- 左移
            end if;

        when left_win =>
            left_score_reg <= left_score_reg + 1;
            state <= reserve; -- 返回预备状态
            led_position <= "00000001"; -- 重置LED

        when right_win =>
            right_score_reg <= right_score_reg + 1;
            state <= reserve; -- 返回预备状态
            led_position <= "00000001"; -- 重置LED

        when others =>
            null;
      end case;
    end if;
  end process;

end behavioral;
```

剛開始我將程式分為三大類led_move_state ,counter (位移的) ,count_score (一開始全部寫在一起要除錯也看得很花)

接著修改
1.加上除頻
2.加上mealy機才能上發球狀才不會一直被誤觸
3.分數並沒有確實顯示出來被存在left_score_reg 和right_score_reg 裡面要修改合併進led_position

位移方式改為 " **count <= count(6 downto 0) & '0';** "的方式運算會快一點因此我造著這個骨架從新打了一個也將分數顯示改為
" **(left_score(0)&left_score(1)&left_score(2)&left_score(3)) & right_score; "**
" (left_score(0)&left_score(1)&left_score(2)&left_score(3)) & right_score; "

剛開始給GPT的指令方向很重要而且要很完整,錯了很難導正,還有因為程式碼GPT打得都很精簡所以沒有考慮的都會變成bug,它不會對信號做檢查缺乏穩定性很容易會出現bug,對GPT敘述時可以對著狀態圖描述成文字會讓指令更清晰完整,然後再加上而外的規則生成的程式碼就會更符合預期