STEP FPGA drives passive buzzer module

In this section, we will use FPGA to drive the passive buzzer module on the backplane to realize the output of different syllables.

====Hardware description====

Classification of buzzers:

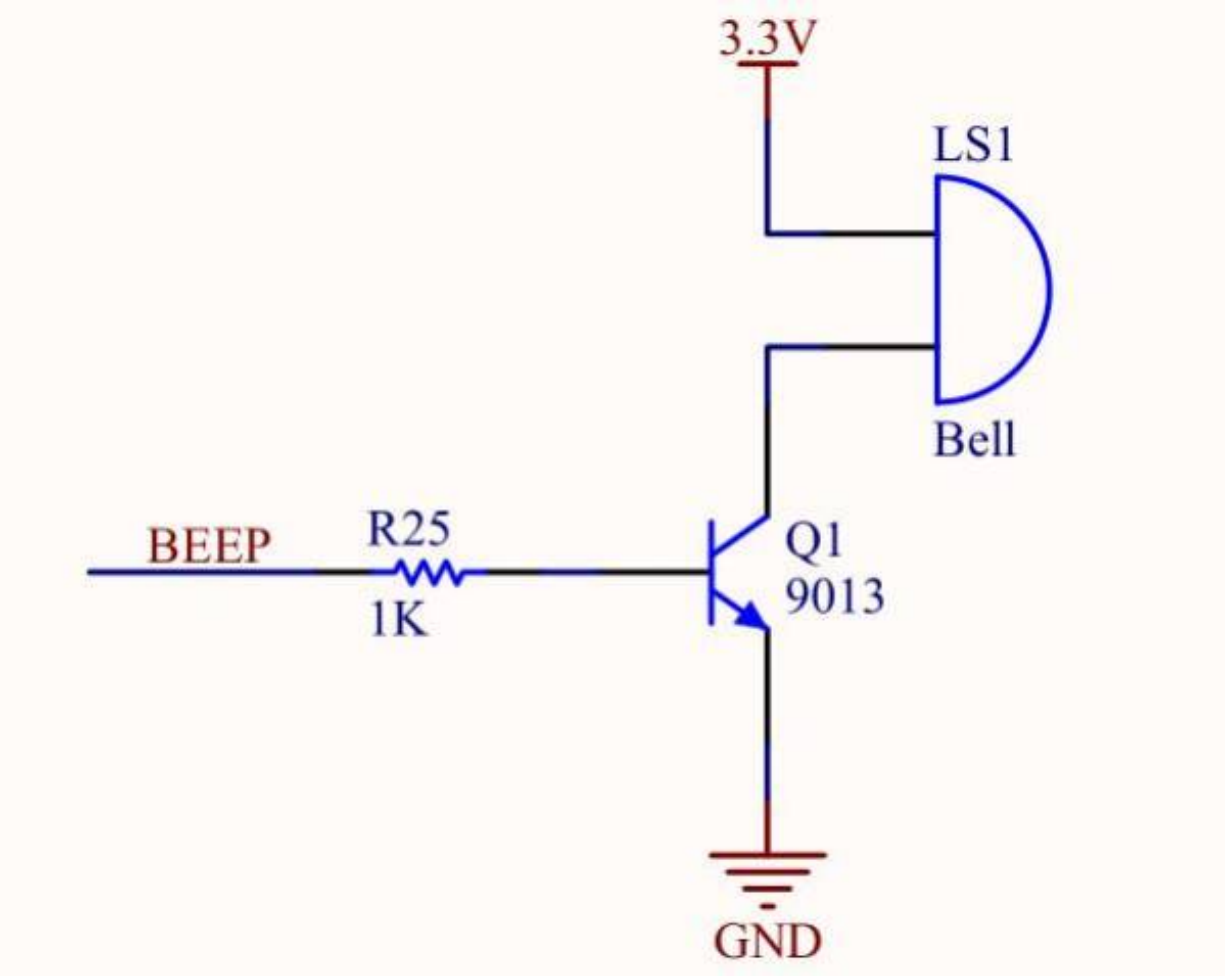
According to their structure, they are mainly divided into piezoelectric buzzers and electromagnetic buzzers:

- The electromagnetic buzzer is composed of an oscillator, an electromagnetic coil, a magnet, a vibrating diaphragm and a shell. After the power is turned on, the audio signal current generated by the oscillator passes through the electromagnetic coil, which causes the electromagnetic coil to generate a magnetic field. The vibrating diaphragm periodically vibrates and sounds under the interaction of the electromagnetic coil and the magnet.
- The piezoelectric buzzer is mainly composed of a multivibrator, a piezoelectric buzzer, an impedance matcher, a resonance box, and a shell. The multivibrator is composed of transistors or integrated circuits. When the power is turned on (1.5 ~ 15V DC working voltage), the multivibrator starts to vibrate and outputs 1.5 ~ 2.5kHZ audio signals. The impedance matcher drives the piezoelectric buzzer. Sound.

There are two types of active buzzer and passive buzzer according to whether it has a signal source:

- An active buzzer only needs to add a rated DC voltage to its power supply terminal, and its internal oscillator can generate a fixed-frequency signal to drive the buzzer to emit sound.
- A passive buzzer can be understood as the same as a horn. It needs to add a constantly changing electrical signal to its power supply terminal to drive the sound.

In this chapter, we will learn about the driver of passive buzzer together. The GPIO port of FPGA or MCU has weak driving ability and cannot directly drive passive buzzer. The commonly used buzzer driving circuit is as follows: buzzer uses NPN transistor (9013) Drive, the transistor is used as a switch. When the base voltage is high, the buzzer is powered on. When the base voltage is low, the buzzer is powered off. FPGA controls the GPIO port to output pulse signals of different frequencies to the base of the transistor. , The buzzer can emit different syllables. The corresponding relationship between different syllables and the buzzer oscillation frequency is as follows: We use the PWM method (the description of PWM, the pulse generator chapter in the quick start has a detailed introduction), use the counter to divide the system clock, and change the counter's Count the final value to achieve the purpose of adjusting the frequency of the PWM signal, and use the PWM signal to control the buzzer circuit.



音节名	频率(Hz)	音节名	频率(Hz)	音节名	频率(Hz)
低音1	261.6	中音1	523.3	高音1	1045.5
低音2	293.7	中音2	587.3	高音2	1174.7
低音3	329.6	中音3	659.3	高音3	1318.5
低音4	349.2	中音4	698.5	高音4	1396.9
低音5	392	中音5	784	高音5	1568
低音6	440	中音6	880	高音6	1760
低音7	493.9	中音7	987.8	高音7	1975.5

====Verilog code====

```
// ---------------------------------------------- --------------------
// >>>>>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE < <<<<<<<<<<<<<<<<<<<<<<<
// ------------------------ ----------------------------------------------
// Module: Beeper
/ /
// Author: Step
//
// Description: Beeper
//
// Web: www.stepfapga.com
//
// ---------------------- ----------------------------------------------
// Code Revision History:
// ---------------------------------------------- ----------------------
// Version: |Mod. Date: |Changes Made:
// V1.0 |2016/04/20 |Initial ver
// ---------------------------------- ----------------------------
module Beeper
 (
input                                   clk_in ,                //system clock
input                                   rst_n_in ,      //system reset, low active
input                                   tone_en ,       //buzzer enable signal
input                   [ 4 : 0 ]       tone ,          //buzzer syllable control
output  reg                             piano_out        / /Buzzer control output
) ;
/*
Passive buzzers can emit different syllables, which are related to the frequency of buzzer vib
ration (equal to the frequency of the buzzer control signal).
In order to make the buzzer control signal produce different frequencies, we use the counter t
o count (frequency division) to achieve, different syllable control corresponds to different f
inal counts (frequency division coefficient)
The counter counts and divides the frequency according to the final count value to generate a
 buzzer control signal
*/
reg  [ 15 : 0 ] time_end ;
//According to different syllable control, select the corresponding final count value (frequen
cy division coefficient)
//The frequency of bass 1 is 261.6Hz, and the buzzer control signal period should be 12MHz/26
1.6 Hz = 45871.5,
//Because the buzzer control signal in this design is flipped according to the counter cycle,
 several final values = 45871.5/2 = 22936
// need to count 22936, the counting range is 0 ~ (22936-1) , So time_end = 22935
always @ ( tone )  begin
        case ( tone )
                5'd1 :  time_end =      1 6'd22935 ;    //L1,
                5'd2 :  time_end =      1 6'd20428 ;    //L2,
                5'd3 :  time_end=       1 6'd18203 ;    //L3,
                5'd4 :  time_end =      1 6'd17181 ;    //L4,
                5'd5 :  time_end =      1 6'd15305 ;    //L5,
                5'd6 :  time_end =      1 6'd13635 ;    //L6,
                5'd7 :  time_end =      1 6'd12147 ;    //L7,
                5'd8 :  time_end =      1 6'd11464 ;    //M1,
```

```
                5'd9 :  time_end =      1 6'd10215;     //M2,
                5'd10 :         time_end =      1 6'd9100 ;     //M3,
                5'd11 :         time_end =      1 6'd8589 ;     //M4,
                5'd12 :         time_end =      1 6'd7652 ;     //M5,
                5'd13 :         time_end =      1 6'd6817 ;     //M6,
                5'd14 :         time_end =      1 6'd6073 ;     //M7,
                5'd15 :         time_end =      1 6'd5740 ;     //H1,
                5'd16:  time_end =      1 6'd5107 ;     //H2,
                5'd17 :         time_end =      1 6'd4549 ;     //H3,
                5'd18 :         time_end =      1 6'd4294 ;     //H4,
                5'd19 :         time_end =      1 6 'd3825 ;    //H5,
                5'd20 :         time_end =      1 6'd3408 ;     //H6,
                5'd21 :         time_end =      1 6'd3036 ;     //H7,
                default : time_end=     1 6'd65535 ;
        endcase
end

reg  [ 17 : 0 ] time_cnt ;
//When the buzzer is enabled, the counter counts according to the final count value (division
 coefficient)
always @ ( posedge clk_in or  negedge rst_n_in )  begin
        if ( ! rst_n_in )  begin
                time_cnt <=  1' b0 ;
        end  else  if ( ! tone_en )  begin
                time_cnt <=  1'b0 ;
        end  else  if ( time_cnt >= time_end )  begin
                time_cnt <=  1'b0 ;
        end  else  begin
                time_cnt <= time_cnt +  1'b1 ;
        end
end

//According to the cycle of the counter, flip the buzzer control signal
always @ ( posedge clk_in or  negedge rst_n_in )  begin
        if ( ! Rst_n_in )  begin
                piano_out <=  1'b0 ;
        end  else  if ( time_cnt == time_end )  begin
                piano_out <=  ~ piano_out ;     //buzzer control output flip, flipped twice to
1Hz
        end  else  begin
                piano_out <= piano_out ;
        end
end

endmodule
```

====Summary====

This section mainly explains the different types of buzzers and the driving principles of passive buzzers. You need to create your own projects while mastering them, and generate FPGA configuration file loading tests through the entire design process.
If you are not familiar with the use of Diamond software, please refer to here: Use of Diamond .

====Related Information====

Use STEP-MXO2 second generation buzzer control program: subsequent download connection will be updated
using the STEP-MAX10 buzzer control program: subsequent download link will be updated