APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL version) Project XII PWM demo based on ZYNQ FPGA resources

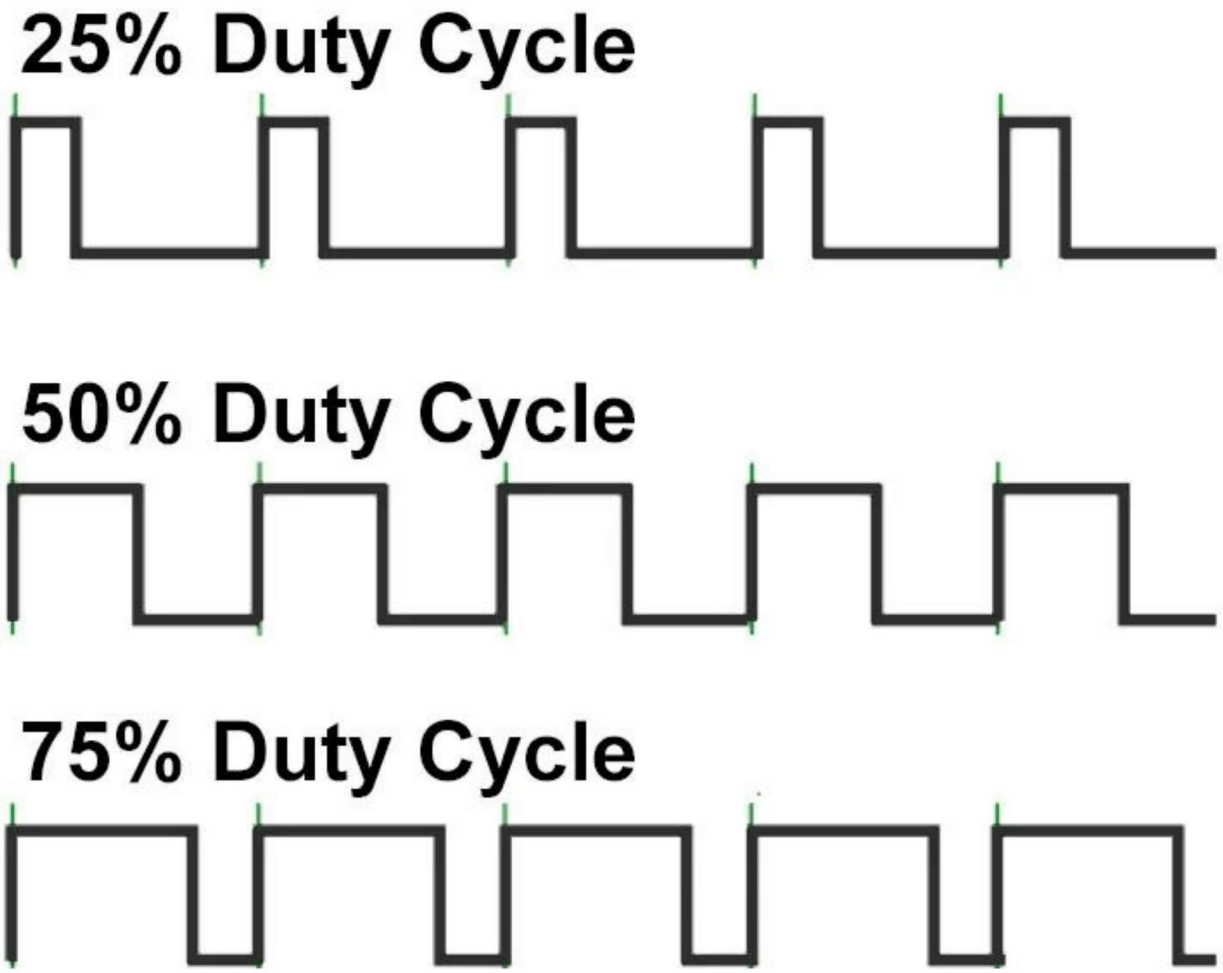
PWM is also pulse width debugging technology, is one of the important technologies in the field of electronics (widely used in switching power supply, motor speed control, LED lamp brightness control and other fields), similar to DSP, microcomputer, STM32 and other microprocessors are with hardware PWM, this article will introduce how to use VERILOG language to write a PWM, and through the way of breathing lamp for the effect demonstration.

This article is demonstrated on vivado2018.3, please research for other versions

(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

Simple principle analysis of PWM to adjust LED brightness:

Let's first look at the standard waveform of PWM, as shown in the figure below, where the period T and frequency F of the upper, middle and lower three waveforms are the same, the time when the high level of the first waveform accounts for 25% of the entire cycle of T, the second waveform accounts for 50%, and the third occupies 75%.



BECAUSE THE PERIOD T AND FREQUENCY F OF THE THREE WAVEFORMS ARE THE SAME, FOR EXAMPLE, THE WAVEFORMS OF THE THREE PWM ARE 1KHZ, THEN IN 1 SECOND, ALL THREE PWM WAVEFORMS WILL HAVE 1000 SQUARE WAVES WITH A PERIOD T. If you take these 3 waveforms to light up the 3 LED lights at this time, the 3 LEDs will flash 1000 times (just because of the visual residual factors of the human eye, so you can't feel this flash), but because the 3 have different duty cycles (high level accounts for different percentages of the entire cycle), the top 25% duty cycle waveform has the least time to light up per unit time, and the 75% duty cycle waveform has the longest time to brighten. The visual result is that the LED light lit by the third waveform is the brightest, and the LED light corresponding to the first waveform is the darkest, which is the simple principle of PWM dimming. Similarly, with this method, scenarios such as motors can also be analyzed.

Also when the duty cycle is 0%, the LED light is completely off, when the duty cycle is 100%, the LED light is completely lit, with the help of this principle, we only need to gradually increase the duty cycle from 0% to 100%, and then gradually reduce from 100% to 0% continuous change, you can achieve the effect of breathing lamps.

Code writing

First of all, we need to design a periodic counter to count the waveform period, we define the frequency of the duty cycle to be 1KHZ, then a cycle of the waveform is 1ms, under the 50MHz clock, 1ms is equivalent to oscillating 50000,1100001101010000 times (corresponding to binary 16, that is, 16 bits), so here define a <>-bit register for counting

```
reg [15:0] period_cnt;
```

The code for the count is as follows

```
 reg  [15:0]  time_count;

//从0-50000的计数器，即1ms计数器
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)begin
        time_count<=16'd0;
    end
    else if(time_count == 16'd50_000)
        time_count <= 16'd0;
    else
        time_count <= time_count + 1'b1;
end
```

where if(!rst_n)begin time_count <=16'd0; End This sentence is responsible for reset, and the external rst_n signal is connected to the button

Just a counter can not achieve the output function of the waveform, next we simply write a program to output 50% square wave, just one sentence

wire led=(time_count <16’d25_000)?1:0;

This sentence is equivalent to time_count counting from 0 to 50_000, when less than 25000, output 1, when greater than 25000 output 0, and because the whole cycle is 50000,

and 25000 is exactly half, that produces half of the high level, half of the low level, that is, the standard 50% square wave.

Our goal is to design breathing lights, just turn the 25000 into a register from small to large and from large to small pwm_perid and that's it.

The code is as follows

```
reg  [15:0]  pwm_perid=16'd0;
reg  mode=0;//mode为0则自增，mode为1为自减
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)begin
        pwm_perid<=16'd0;
    end
    else if(time_count== 16'd50_000)begin
        if(mode==1'b0)begin
            if(pwm_perid<16'd50_000) pwm_perid<=pwm_perid+50;
            else mode<=1'b1;
         end
        else if(mode==1'b1)begin
            if(pwm_perid>16'd0) pwm_perid<=pwm_perid-50;
            else mode<=1'b0;
         end
    end
end
```

The change period of the addition and subtraction program is 1ms, that is, 1 changes in 1000 second, here is lazy, directly use the above time_count counter to make the change cycle here with if(time_count== 16'd50_000), of course, you can also write a new one yourself, all the same

There is a reg mode in the program, this is used to record the working mode, mode is 0 pwm_perid autoincrement, mode is 1pwm_perid decrement

The condition for mode switching is that the duty cycle is maximized, or the duty cycle is reduced to 0%

The code is simple, directly compile and synthesize and download the FPGA observations

The additional constraint file is as follows

```
set_property IOSTANDARD LVCMOS33 [get_ports led]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
set_property PACKAGE_PIN M19 [get_ports clk]
set_property PACKAGE_PIN P20 [get_ports led]
set_property PACKAGE_PIN K21 [get_ports rst_n]
```

After downloading, you can see that the LED1 is constantly dimming and the variable is dimmed

The complete project is as follows:

12_PL_PWM_TEST_XC7Z020 **Download**

SMART ZYNQ SP & SL