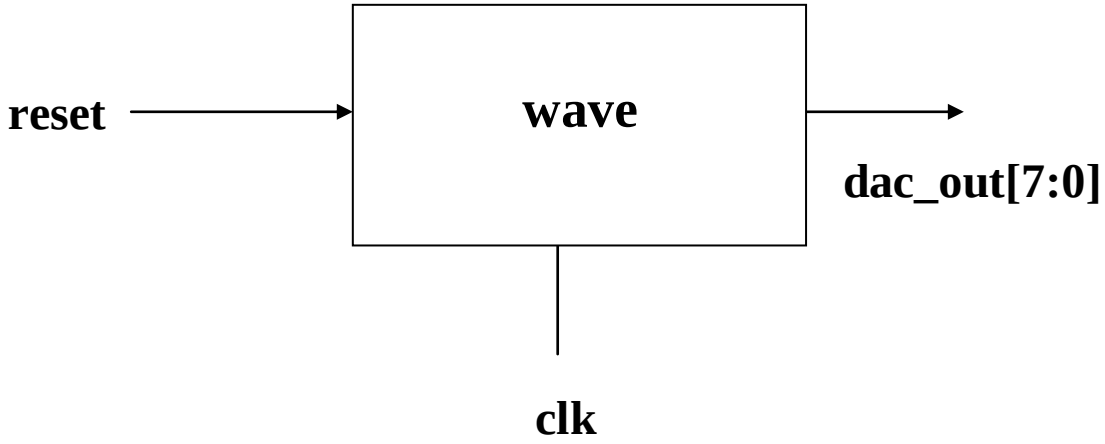
HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC change the frequency and amplitude.

External View



Theory:

DAC0808 8-Bit D/A Converter

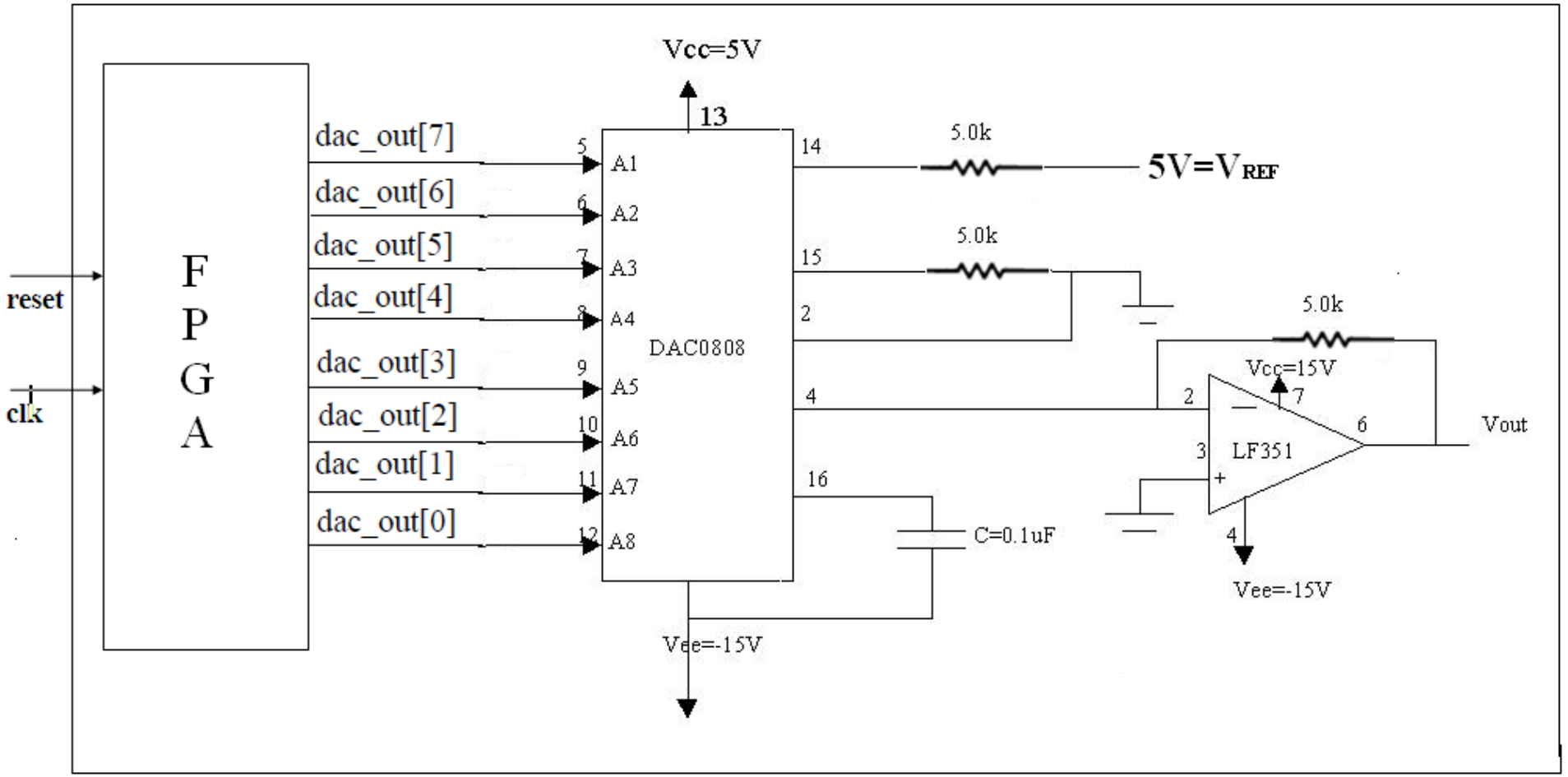
General Description

The DAC0808 is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with ±5V supplies. No reference current (I_{REF}) trimming is required for most applications since the full scale output current is typically ±1 LSB of 255 $I_{REF}/256$. Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8-bit zero accuracy for $I_{REF} \geq 2$ mA. The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

Features

- Relative accuracy: ±0.19% error maximum
- Full scale current match: ±1 LSB typ
- Fast settling time: 150 ns typ
- Noninverting digital inputs are TTL and CMOS compatible
- High speed multiplying input slew rate: 8 mA/µs
- Power supply voltage range: ±4.5V to ±18V
- Low power consumption: 33 mW @ ±5V

Interfacing Diagram



$$V_O = 5V\left(\frac{A1}{2}+\frac{A2}{4}+\ldots\frac{A8}{256}\right)$$

Triangular Wave

VHDL File:

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

entity triangular_wave is
Port ( clk : in std_logic;
       reset : in std_logic;
       dac_out: out std_logic_vector(7 downto 0));
end triangular_wave ;
architecture triangular_wave of triangular_wave is
signal counter : std_logic_vector(8 downto 0);
signal clk_div : std_logic_vector(25 downto 0);
begin
process(clk)
begin
if rising_edge(clk) then
clk_div <= clk_div + '1' ;
end if;
end process;
process(clk_div (3))
begin
if reset='1' then
counter <= "111111110";
elsif rising_edge(clk_div (3)) then
counter <= counter + 1 ;
if counter(8)='0' then
dac_out <=counter(7 downto 0);
else
dac_out <=not(counter(7 downto 0));
end if;
end if;
end process;
end triangular_wave;
```

XDC File:

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {reset}]

        set_property IOSTANDARD LVCMOS33 [get_ports {reset}]

##Pmod Header JC
##Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {dac_out[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[0]}]
##Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports {dac_out[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[1]}]
##Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {dac_out[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[2]}]
##Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports {dac_out[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[3]}]
##Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports {dac_out[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[4]}]
##Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports {dac_out[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[5]}]
##Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports {dac_out[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[6]}]
##Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports {dac_out[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dac_out[7]}]
```

Square Wave VHDL Code

```
entity square_wave is
Port ( clk : in std_logic;
       reset : in std_logic;
       dac_out : out std_logic_vector(7 downto 0));
end square_wave;

architecture square_wave of square_wave is
signal clk_div : std_logic_vector(3 downto 0);
signal counter : std_logic_vector(7 downto 0);
signal en :std_logic;
begin

process(clk)
begin
if rising_edge(clk) then
clk_div <= clk_div + '1' ;
end if;
end process;
process(clk_div(3))
begin
if reset='1' then
counter <= "00000000";
elsif rising_edge(clk_div(3)) then
  if counter<255 and en='0' then
  counter <= counter + 1 ;
  en<='0';
  dac_out <="00000000";
  elsif counter=0 then
  en<='0';
  else
  en<='1';
counter <= counter-1;
dac_out <="11111111";
 end if;
end if;
end process;
end square_wave;
```

Sawtooth Wave

VHDL Code:

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

entity swatooth_wave is
Port ( clk : in std_logic;
        reset : in std_logic;
       dac_out : out std_logic_vector(7 downto 0));
end swatooth_wave;

architecture swatooth_wave of swatooth_wave is
signal clk_div : std_logic_vector(3 downto 0);
signal counter : std_logic_vector(7 downto 0);
signal en :std_logic;
begin

process(clk)
begin
if rising_edge(clk) then
clk_div <= clk_div + '1' ;
end if;
end process;

process(clk_div (3))
begin
   if reset='1' then
   counter <= "00000000";
  elsif rising_edge(clk_div (3)) then
  counter <= counter + 1 ;
  end if;
end process;

dac_out <=counter;
end swatooth_wave;
```

Sine Wave

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

Entity sinewave is
port ( clk : in std_logic;
        reset : in std_logic;
        dac_out : out std_logic_vector(7 downto 0));
End sinewave;

architecture sinewave of sinewave is
Signal clk_div:std_logic_vector(7 downto 0);
Signal I :integer range 0 to 179;
Type sine is array (0 to 179) of integer range 0 to 255;
Constant  value:
sine:=(128,132,136,141,154,150,154,158,163,167,171,175,180,184,188,19
2,195,199,203,206,210,213,216,220,223,226,228,231,234,236,238,241,243
,244,246,247,248,249,250,251,252,253,254,255,255,255,255,255,254,254,
253,252,251,249,246,244,243,241,238,236,234,231,228,226,223,220,216,2
13,210,206,203,199,195,192,188,184,180,175,171,167,163,158,154,150,14
5,141,136,132,128,123,119,114,110,105,101,97,92,88,84,80,75,71,67,64,6
0,56,52,49,45,42,39,35,32,29,27,24,21,19,17,14,12,11,9,7,6,4,3,2,1,1,0,0,0,
0,0,0,0,0,1,1,2,3,4,6,7,9,11,12,14,17,19,21,24,27,29,32,35,39,42,45,49,52,5
6,60,64,67,71,75,80,84,88,92,97,101,105,110,114,119,123,128);
```

```
begin

Process(clk,reset)
begin
      if(reset='1') then
      clk_div <=(others=>'0');
      elsif(clk'event and clk='1') then
      clk_div <= clk_div +1;
      end if;
end process;

process(clk_div (3))
begin

if(clk_div (3)'event and clk_div (3)='1')then
   dac_out<=conv_std_logic_vector(value(i),8);
   I<=I+1;
   if(i=179) then
   i<=0;
  end if;
end if;
end process;

end sinewave;
```

Angle vs. Voltage Magnitude for Sine wave			
Angle Ø (Degrees)	Sin Ø	V_{out} (Voltage magnitude) 5V+(5V* Sin Ø)	Values sent to DAC (decimal) (Voltage mag. * 25.6)
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128