پیش گزارش تولید شکل موج سینوسی توسط DAC

نام: آناهیتا نام خانوادگی: اسدی شماره دانشجویی: 9724453

نام مدرس: سرکار خانم فرشته کلانتری روز و ساعت کلاس: یکشنبه ها، ساعت 13:30 الی 15:30

پاسخ مرحله 1

لازم است محاسبه کنیم برای تولید هر فرکانس دلخواه، باید چه تعداد کلاک (Number of Clocks) بگذرد. اگر تعداد نمونه های برداشته شده برابر N باشد، خواهیم داشت:

$$NC = \frac{f_{fpga\ clk}}{f_{sampling} \times N}$$

حالت 1: فرکانس 500Hz:

$$NC = \frac{50MHz}{500 \times 64} = 1562.5 \cong 1563$$

حالت 2: فرکانس 250Hz:

$$NC = \frac{50MHz}{250 \times 64} = 3125$$

حالت 3 و 4: فرکانس 125Hz:

$$NC = \frac{50MHz}{250 \times 64} = 6250$$

.1

اضافه کردن کتابخانه های موردنیاز؛ کتابخانه اول برای گیت های منطقی (در این آزمایش not کردن یک سیگنال)، کتابخانه دوم برای عملیات ریاضی (مانند جمع کردن i و count با 1) و کتابخانه سوم برای اعداد غیرعلامت دار (برای مثال 1000 را 8 (و نه -7) درنظر میگیرد) است.

```
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
```

تعریف پورت های ورودی و خروجی

```
entity dac is port(  clk: in std_logic;
                     din: in std_logic_vector(1 downto 0);
                     dout: out std_logic_vector(7 downto 0));
end dac;
```

تعریف تایپ حافظه ی 64 بایتی و سپس قراردادن 64 سمپل سینوسی در آن

```
architecture behavioral of dac is
   type memory  is array (0 to 63) of integer range 0 to 255;
   signal sin: memory := ( 128,  140,  152,  164,  176,  187,  198,  208,
                           217,  226,  233,  240,  245,  249,  252,  254,
                           255,  254,  252,  249,  245,  240,  233,  226,
                           217,  208,  198,  187,  176,  164,  152,  140,
                           128,  116,  104,  92,   80,   69,   58,   48,
                           39,   30,   23,   16,   11,   7,    4,    2,
                           1,    2,    4,    7,    11,   16,   23,   30,
                           39,   48,   58,   69,   80,   92,   104,  116);
begin
   process(clk)
      variable NC: integer := 0;
      variable din_old: std_logic_vector(1 downto 0) := "00";
      variable count: integer := 0;
      variable i: integer := 0;
   begin
      case din is when "00"   => NC := 1563;
                  when "01"   => NC := 3125;
                  when "10"   => NC := 6250;
                  when others => NC := 6250;
      end case;
         if (clk'event and clk = '1') then
            if (din_old /= din) then
               i := 0;
               count := 0;
               din_old := din;
            end if;
            count := count + 1;
            if (count = NC) then
               i := i + 1;
                  if (i = 63) then
                     i := 0;
                  end if;
               count := 0;
            end if;
         end if;
         case din is when "11"   => dout <= conv_std_logic_vector((sin(i))/2,8);
                     when others => dout <= conv_std_logic_vector((sin(i)),8);
         end case;
   end process;
end behavioral;
```

تعریف NC مطابق فرمول نوشته شده، و تحت عنوان variable، چون قرار بود تنها در پروسه از آن استفاده شود.

تعریف حافظه ای برای din که به محض تغییر din، خروجی تغییر کند

تعریف count برای تقسیم فرکانس، و تعریف i برای نشان دادن شماره آرایه از sin (که تایپ آن memory است)

تعریف حالات مختلف تقسیم فرکانسی، متناسب با ورودی din

با هر لبه بالا روند کلاک، چک میشود آیا به حد تقسیم فرکانسی مدنظر رسیده ایم یا خیر. اگر رسیده بودیم، آرایه بعدی در خروجی قرار داده میشود. همچنین وقتی تمام 64 خانه sin خوانده شدند، این کار از اول تکرار میشود.

در 3 حالت اول، سیگنال سینوسی با دامنه sin و درحالت چهارم دامنه نصف میشود. همچنین چون dout از جنس std_logic_vector و خانه های sin از جنس integer هستند، برای اینکه در هم ریخته شوند، باید تبدیل شوند.

.2

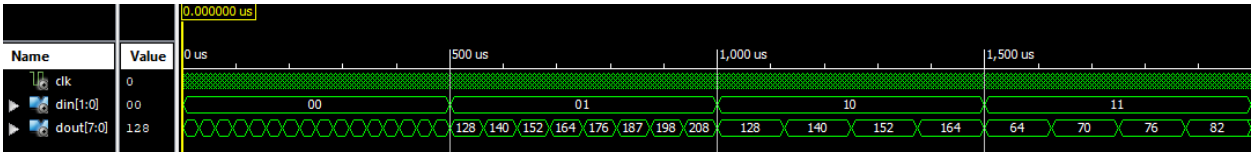
افزایش تعداد سمپل ها، دقت را بالاتر میبرد و خروجی، پیوسته تر بنظر می آید؛ ولی تعداد خانه های حافظه از 64 بیشتر میشود بنابراین حجم بیشتری اشغال خواهد شد.

پاسخ مرحله 2

فرکانس کلاک 50MHz است؛ یعنی ورودی clk باید هر 10ns، not شود:

```
clk <= not(clk) after 10 ns;
din <= "00" after 5 ns, "01" after 500 us, "10" after 1 ms, "11" after 1500 us;
```

نتیجه تست بنچ:



در حالت اول، فرکانس سمپلینگ برابر است با:

$$\frac{16}{500\mu s} = 32kHz$$

در حالت دوم:

$$\frac{8}{500\mu s} = 64kHz$$

سوم:

$$\frac{4}{500\mu s} = 128kHz$$

فرکانس سمپلینگ حالت چهارم برابر سوم است، ولی دامنه نصف شده است. در نهایت در خروجی موج سینوسی با فرکانس به ترتیب 500، 250 و 125 هرتز خواهیم داشت.

پاسخ مرحله 3

```
=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================


Synthesizing (advanced) Unit <dac>.
INFO:Xst:3034 - In order to maximize performance and save block RAM resources,
Unit <dac> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 1
 64x8-bit ROM                                          : 1
# Adders/Subtractors                                   : 2
 32-bit adder                                          : 2
# Registers                                            : 66
 Flip-Flops                                            : 66
# Comparators                                          : 3
 2-bit comparator not equal                            : 1
 32-bit comparator equal                               : 2
# Multiplexers                                         : 1
 32-bit 4-to-1 multiplexer                             : 1

=========================================================================
```

از یک ROM (64 بایتی) برای نگهداری 64 سمپل سینوسی 8 بیتی

بجای شمارشگر از ترکیب رجیستر با جمع کننده استفاده شده است. بنابراین از جمع کننده و مقایسه گر 32 بیتی (چون count از 0 تا NC (حداکثر برابر 6250 یعنی 32 بیت) میشمارد و جمع میکند و مقایسه میکند) استفاده شده است. جمع کننده و شمارنده دوم متعلق به i است. تا اینجا 64 رجیستر استفاده شده است.

مقایسه گر دو بیتی برای این است که اگر کاربر din را عوض کرد، dout از نمونه اول سینوسی موج را ایجاد کند (و نه ادامه شمارش قبلی). 2 بیت رجیستر هم اینجا استفاده شده یعنی مجموعا 66 رجیستر.

MUX32x4، برای حالات مختلف NC است. ورودی های 2 بیتی din را میگیرد و خروجی 32 بیتی NC میدهد. چون 4 حالت وجود دارد، 4 خروجی دارد.

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : dac.ngr
Top Level Output File Name         : dac
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : No

Design Statistics
# IOs                              : 11

Cell Usage :
# BELS                             : 320
#      GND                         : 1
#      INV                         : 2
#      LUT2                        : 96
#      LUT2_L                      : 2
#      LUT3                        : 5
#      LUT3_L                      : 1
#      LUT4                        : 56
#      LUT4_D                      : 2
#      LUT4_L                      : 1
#      MUXCY                       : 72
#      MUXF5                       : 11
#      MUXF6                       : 6
#      VCC                         : 1
#      XORCY                       : 64
# FlipFlops/Latches                : 66
#      FDE                         : 34
#      FDR                         : 32
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 10
#      IBUF                        : 2
#      OBUF                        : 8
=========================================================================
```

برای ساخت بیت های 0 و 1 از GND و Vcc استفاده شده است.

از INV داخل کلاک استفاده شده است.

LUT ها برای نگهداری حالات din و NC، و همچنین در ROMها بکار رفته اند. برای انتخاب هر حالت NC از MUX، برای ساخت مقایسه گر و جمع کننده از XOR، و برای مقادیر داخل پروسه از لچ استفاده شده است.

رجیسترها از فلیپ فلاپ تشکیل شده اند پس 66 فلیپ فلاپ نیاز است.

2 بیت ورودی، 1 کلاک و 8 بیت خروجی داریم، بنابراین 1 بافر کلاک، 10 پورت ورودی/خروجی استفاده شده است.

```
Device utilization summary:
---------------------------

Selected Device : 3s400pq208-4

 Number of Slices:                       93  out of   3584     2%
 Number of Slice Flip Flops:             66  out of   7168     0%
 Number of 4 input LUTs:                165  out of   7168     2%
 Number of IOs:                          11
 Number of bonded IOBs:                  11  out of    141     7%
 Number of GCLKs:                         1  out of      8    12%
```

جمع بندی موارد بالا، و محاسبه درصد استفاده از آنها، نسبت به کل منابع موجود در fpga.

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 16.707ns (Maximum Frequency: 59.854MHz)
   Minimum input arrival time before clock: 17.475ns
   Maximum output required time after clock: 13.214ns
   Maximum combinational path delay: 9.573ns
```

توضیحات تایمر، که بیان میکند تایمر fpga بیشتر از حدود 60MHz نمیتواند باشد، و نباید در تست بنچ بعنوان مثال هر 2ns کلاک را عوض کرد چون با واقعیت تفاوت خواهد داشت. همچنین تاخیر رسیدن دیتا در گیت های منطقی دستورات combinational، 9ns است بنابراین اگر زودتر از این زمان، سیگنال ها را عوض کنیم، خروجی نتیجه درستی نخواهد داد. همچنین حدود 13ns طول میکشد تا خروجی، نهایی شود.

پاسخ مرحله 4

کد UCF:

```
net "clk" loc = p80;

net "din[0]" loc = p34;
net "din[1]" loc = p33;

net "dout[0]" loc = p48;
net "dout[1]" loc = p50;
net "dout[2]" loc = p51;
net "dout[3]" loc = p52;
net "dout[4]" loc = p62;
net "dout[5]" loc = p61;
net "dout[6]" loc = p58;
net "dout[7]" loc = p57;
```

clk به کلاک داخلی fpga (50MHz)، din به D0 و D1 از DIP Switch 1، و dout به D0 تا D7 از DACمتصل شده است.