تکلیف عملی سری ۲

درس مدار های منطقی برنامه پذیر

استاد درس: دکتر شریعتمدار

گروه ۴

آنوشا شریعتی ۹۹۲۳۰۴۱

مهشاد اکبری سریزدی ۹۹۲۳۰۹۳

سوال یک : آشنایی با تراشه AD۹۸۳۳

قسمت یک: سوالات کوتاه

الف)

کنترل رجیستر ۱۶ بیتی می باشد که توضیحات مربوط به بیت های آن در سوال دوم آمده است.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	B28	HLB	FSELECT	PSELECT	0	RESET	SLEEP1	SLEEP12	OPBITEN	0	DIV2	0	MODE	0

02704-024

Figure 24. Function of Control Bits

رجیسترهای فرکانس (frequency Registers): دو رجیستر فرکانس ۳۲ بیتی داریم . AD۹۸۳۳ شامل ۲ رجیستر فرکانس است ، FREQ۰ و FREQ۱ که هردو ۲۸ بیتی می باشند و در هرکدام ۲ بیت مختص به کنترل می باشد . (نحوه نوشتن در هرکدام برای ۲۸ بیت به صورت ۱۴بیت داده و ۲ بیت مربوط به تعیین استفاده از کدام رجیستر می باشد و برای ۲۸ بیت نوشتن در آن هارا دو بار انجام میدهد و دفعه اول برای ۱۴ بیت کم ارزش و دفعه دوم برای ۱۴ بیت پر ارزش)

رجیستر های فاز (Phase Registers): AD۹۸۳۳ شامل دو رجیستر فاز ۱۶ بیتی است ، PHASE۰ و PHASE۱ که هردو ۱۲ بیتی می باشند و ۴ بیت اضافه (۱۲و۱۳و۱۴و۱۵) که بیت ۱۴ و ۱۵ به طور پیش فرض ۱ می باشند و بیت ۱۳ مشخص می کند از کدام رجیستر فاز استفاده می کنیم و بیت ۱۲ دونت کر می باشد .

Table 7. Frequency and Phase Registers

Register	Size	Description
FREQ0	28 bits	Frequency Register 0. When the FSELECT bit = 0, this register defines the output frequency as a fraction of the MCLK frequency.
FREQ1	28 bits	Frequency Register 1. When the FSELECT bit = 1, this register defines the output frequency as a fraction of the MCLK frequency.
PHASE0	12 bits	Phase Offset Register 0. When the PSELECT bit = 0, the contents of this register are added to the output of the phase accumulator.
PHASE1	12 bits	Phase Offset Register 1. When the PSELECT bit = 1, the contents of this register are added to the output of the phase accumulator.

ب)

Latency یا همان تاخیر مدت زمانی است که یک درخواست در انتظار رسیدگی است. تاخیر اجتناب ناپذیر است و به نحوه اتصال بین بخش های مختلف بستگی دارد .

اگر یک رجیستر فرکانس یا فاز با یک کلمه جدید لود شود یک مدت زمانی تاخیر ایجاد می شود تا خروجی آنالوگ از حالت قبل تغییر کند . این تاخیر می تواند ۷ یا ۸ دوره MCLK طول بکشد و این بستگی به این دارد که هنگامی که داده در رجیستر مقصد لود می شود با چه فاصله ای از لبه بالارونده MCLK باشد .

د)

از این خازن ها برای حلوگیری از نویز استفاده می شود و مانند فیلتر عمل میکنند . در این جا از دو خازن به صورت موازی استفاده شده است که این دو خازن یک خازن الکترولیتی و دیگری سرامیکی غیر قطبی هستند. خازن الکترولیتی دارای ظرفیت جریان بالایی است به طوری که می تواند پرش جریان بزرگی را در صورت وجود هرگونه پرش در خط منبع تغذیه حمل کند. اما پاسخ فرکانسی این خازن کمتر است که اجازه می دهد پرش تا حدی وجود داشته باشد. خازن سرامیکی در پاسخ فرکانسی خوب است، بنابراین پرش را در خروجی مسدود می کند.

ظرفیت خازن سرامیکی ۰.۱ میکرو فاراد و خازن قطبی ۱۰ میکرو فاراد می باشد .

قسمت دو : ساخت کلاک تراشه

برای ساخت کلاک مورد نیاز برای کار کردن با تراشه در ابتدا مقدار کلاک را در دیتاشیت می یابیم که طبق تصاویر زیر کلاک مستر دارای دوره تناوب ۴۰ نانو ثانیه است یعنی فرکانس ۲۵ مگاهرتز دارد.

Table 2.

Parameter	Limit at T_{MIN} to T_{MAX}	Unit	Description
t_1	40	ns min	MCLK period
t_2	16	ns min	MCLK high duration
t_3	16	ns min	MCLK low duration
t_4	25	ns min	SCLK period
t_5	10	ns min	SCLK high duration
t_6	10	ns min	SCLK low duration

CONTROL REGISTER

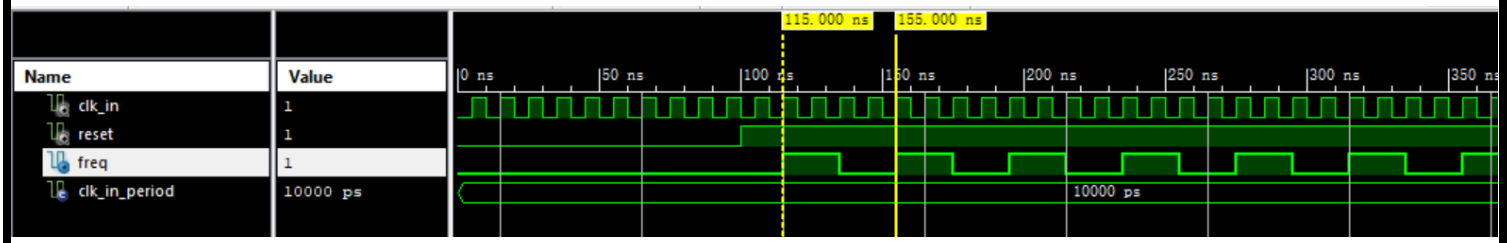
The AD9833 contains a 16-bit control register that allows the user to configure the operation of the AD9833. All control bits other than the mode bit are sampled on the internal falling edge of MCLK.

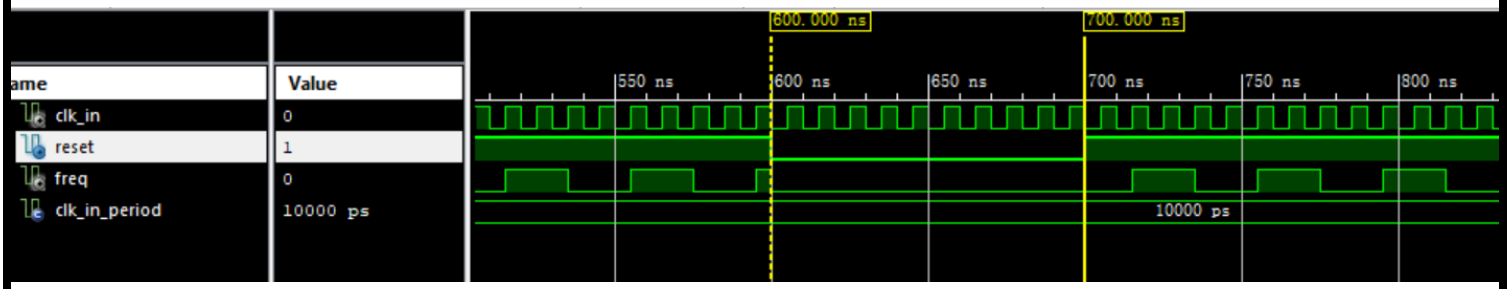
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity main1 is
5      Port ( clk_in : in  STD_LOGIC;
6             reset  : in std_logic;
7             freq   : out std_logic
8             );
9  end main1;
10
11 architecture Behavioral of main1 is
12
13 signal clk_temp : std_logic := '0';
14 signal counter : integer := 0;
15
16 begin
17   process(clk_in, reset)
18   begin
19     if reset = '0' then --active low reset
20
21       counter <= 0;
22
23       clk_temp <= '0';
24
25     elsif rising_edge(clk_in) then
26       -- 100MHz clk (40 nano second)  --> 100 000 000 / 25 000 000 hz
27       if counter =  2-1 then
28         counter <= 0;
29         clk_temp <= not clk_temp;
30       else
31         counter <= counter + 1;
32       end if;
33
34     end if;
35   end process;
36
37   freq <= clk_temp;
38
39 end Behavioral;
```

سپس طبق روش توضیح داده شده در تکلیف سری قبل به صورت بالا کدی برای تولید کلاک با فرکانس ۲۵ مگاهرتز از روی کلاک ورودی ۱۰۰ مگاهرتزی میسازیم. همچنین در نظر میگیریم که این ماژول دارای پایه ریست به صورت active low است یعنی اگر مقدار ریست صفر شود خروجی صفر میشود. با نوشتن تست بنچ زیر کد را در حالات مختلف و هنگام فعال بودن ریست تست کردیم.

```
30
31    -- Clock period definitions
32    constant clk_in_period : time := 10 ns;
33
34  BEGIN
35
36    -- Instantiate the Unit Under Test (UUT)
37    uut: mainl PORT MAP (
38           clk_in => clk_in,
39           reset => reset,
40           freq => freq
41         );
42
43    -- Clock process definitions
44    clk_in_process :process
45    begin
46       clk_in <= '0';
47       wait for clk_in_period/2;
48       clk_in <= '1';
49       wait for clk_in_period/2;
50    end process;
```

```
53    -- Stimulus process
54    stim_proc: process
55    begin
56
57       reset <= '0';
58       wait for 100 ns ; -- hold reset state for 100 ns
59       reset <='1';
60       wait for 500 ns ;
61       reset <= '0';
62       wait for 100 ns ; -- hold reset state for 100 ns
63       reset <='1';
64
65       wait;
66    end process;
67
68  END;
```





سوال دو: مقداردهی رجیسترها

قسمت یک: تعیین مقدار رجیستر ها

برای تعیین مقدار رجیستر ها به دیتاشیت قطعه مراجعه میکنیم تا کارکرد هر رجیستر را متوجه شویم.

کنترل رجیستر : این رجیستر ۱۶ بیتی است و با تغییر دادن بیت های آن کاربر میتواند نحوه عملکرد قطعه را تعیین کند.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	B28	HLB	FSELECT	PSELECT	0	RESET	SLEEP1	SLEEP12	OPBITEN	0	DIV2	0	MODE	0

Figure 24. Function of Control Bits

بیت D۱۵ و D۱٤ همیشه مقدار ۰ دارند و نشان میدهند که مقادیر کنترل رجیستر دارد عوض میشود.

بیتD۱۳ اگر یک باشد اجازه میدهد که یک کلمه کامل درون رجیستر فرکانس در دو مرحله لود شود. ولی اگر بیتD۱۳ صفر باشد به این معناست که رجیستر فرکانس به عنوان دو رجیستر ۱۴ بیتی عمل میکند یکی شامل ۱۴ بیت پر ارزش و یکی شامل ۱۴ بیت کم ارزش.

بیت D۱۲ به همراه بیت D۱۳ کاربرد دارد و فقط وقتی بیت D۱۳ صفر باشد کاربرد دارد و وقتیD۱۳ یک است نادیده گرفته میشود. اگر بیت D۱۲ یک باشد دیتا در ۱۴ بیت پر ارزش رجیستر فرکانس نوشته میشود و اگر صفر باشد در ۱۴ بیت کم ارزش.

بیت D۱۱ نشان میدهد که کدام یک از رجیسترهای FREQ۱ یا FREQ۰ در جمع کننده فاز استفاده میشوند.

بیت D۱۰ نشان میدهد که کدام از رجیسترهای PHASE۱ یا PHASE۰ به خروجی جمع کننده فاز اضافه میشوند.

بیت D۹ همیشه صفر است.

بیت D۸ اگر یک باشد رجیستر های داخلی ریست شده و صفر میشوند. این بیت اگر صفر باشد ریست غیر فعال است.

اگر بیت D۷ یک شود کلاک داخلی ۲۵ هرتزی غیر فعال است و خروجی DAC مقدار خود را حفظ میکند .

اگر بیت D٦ یک باشد DAC غیر فعال میشود و در غیر این صورت فعال است.

اگر بیت D۵ یک باشد خروجی DAC روی پایه VOUT حاضر نیست و اگر صفر باشد DAC به VOUT متصل است.

بیت D٤ همیشه صفر است.

اگر بیتD۳ یک باشد بیت با ارزش DAC به پین VOUT متصل میشود و اگر این بیت صفر باشد نصف آن به پین VOUT میرود.

بیت D۲ همیشه صفر است.

وقتی بیت D۱ صفر باشد در خروجی سیگنال سینوسی ظاهر میشود و اگر یک باشد سیگنال مثلثی.

بیت D۰ همیشه صفر است.

رجیستر فرکانس: این ماژول دارای دو رجیستر فرکانسfreq۱ و freq۰ ۲۸ بیتی است. اگر بیتD۱۱ کنترل رجیستر صفر باشد FREQ۰ نشان دهنده فرکانس خروجی است و اگر بیت D۱۱ کنترل رجیستر یک باشد FREQ۱ نشان دهنده خروجی است.

رجیستر فاز: این ماژول دارای دو رجیستر فاز PHASE۱ و PHASE۰ ۱۲ بیتی است. اگر بیتD۱۰ کنترل رجیستر صفر باشد PHASE۰ به خروجی جمع کننده فاز اضافه میشود و اگر بیت D۱۰ کنترل رجیستر یک باشد PHASE۱ به خروجی جمع کننده فاز اضافه میشود.

تعیین مقدار کنترل رجیستر:

طبق جدول زیر اگر بخواهیم سیگنال خروجی مثلثی باشد در کنترل رجیستر بیت D۵ را صفر و بیت D۱ را یک و بیت D۳ را دونت کر قرار میدهیم که در اینجا ۱ فرض کردیم. همچنین برای ایجاد موج مثلثDAC باید فعال باشد پس D٦ صفر است.

Table 15. Outputs from the VOUT Pin

OPBITEN Bit	Mode Bit	DIV2 Bit	VOUT Pin
0	0	X[1]	Sinusoid
0	1	X[1]	Triangle
1	0	0	DAC data MSB/2
1	0	1	DAC data MSB
1	1	X[1]	Reserved

[1] X = don't care.

طبق خواسته سوال میخواهیم از ۱ frequency register و ۰ phase register استفاده شود پس در کنترل رجیستر بیت D۱۰ صفر و بیت D۱۱ یک است.

همچنین اگر بخواهیم در ابتدا تراشه ریست شود بیت D۸ کنترل رجیستر باید ۱ شود.

با توجه به این که دیتایی که در رجیستر فرکانس ریخته میشود بیشتر از ۱۴ بیت است پس باید از کل حافظه رجیستر فرکانس استفاده شود پس بیتD۱۳ کنترل رجیستر ۱ است تا دیتا با دو عملیات لود شدن در رجیستر ۲۸ بیتی نوشته شود. همچنین بیتD۱۲ کنترل رجیستر دونت کر میشود.

طبق توضیحات و فرضیات بالا دیتا رجیستر به صورت زیر باید پر شود. بیت های زرد رنگ مقدار ثابت ۰ را داشتند. بیت های سبز رنگ را با توجه به خواسته های مسئله مشخص کردیم و بیت های قرمز رنگ دونت کر بودند که فرضی در نظر گرفتیم.

Control Register = ۰۰۱۰ ۱۰۰۱ ۰۰۰۰ ۱۰۱۰

تعیین مقدار رجیستر فرکانس:

برای تعیین کردن مقدار رجیستر فرکانس از فرمول زیر استفاده میکنیم با فرض این که فرکانس ورودی ۲۵ مگاهرتز و فرکانس سیگنال خروجی ۹ کیلو هرتز است.

$$FreqReg = \frac{f_{OUT} \times 2^{28}}{f_{MCLK}}$$

با انجام محاسبات عدد دسیمال۹۶۶۳۶.۷۶۴۱۶ به دست می آید. با روند کردن عدد به سم بالا و تغییر مبنا عدد هگزD۷۹۷ ۰x۱ به دست می آید.

Frequency register = ۰۰۰۱ ۰۱۱۱ ۱۰۰۱ ۰۱۱۱ ۱۱۰۱

این دیتا به صورت دو کلمه ۱۶ بیتی به شکل زیر به رجیستر فرکانس منتقل میشود.

بیت ۱۴ و۱۵ در سمت چپ باید به صورت ۱۰ ثابت باشد زیرا نشان میدهد دیتا در رجیستر FREQ۱ نوشته میشود. اول ۱۴ بیت کم ارزش به صورت زیر ارسال میشود.

۱۰۱۱ ۱۰۰۱ ۰۱۱۱ ۱۱۰۱

سپس ۱۴ بیت پر ارزش ارسال میشود.

۱۰۰۰ ۰۰۰۰ ۰۰۰۰ ۰۱۰۱

تعیین مقدار رجیستر فاز:

چون در صورت سوال اختلاف فاز مشخص نشده است ما این مقدار را ۰ در نظر میگیریم.
طبق جدول زیر عدد ۰ باید به صورت زیر به رجیستر فاز داده شود.

phase register = ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

۱۱۰۰ ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

Table 12. Phase Register Bits

D15	D14	D13	D12	D11	D0
1	1	0	X	MSB 12 PHASE0 bits	LSB
1	1	1	X	MSB 12 PHASE1 bits	LSB

قسمت دو : ارسال اطلاعات به تراشه با spi

برای انجام این قسمت ابتدا برنامه نوشته شده در قسمت قبل را کامپوننت کردیم تا کلاک مستر مورد نیاز را بدست بیاوریم سپس با تعریف یک پراسس حساس به لبه پایین رونده کلاک استیت ماشین خواسته شده برای ارتباط spi را به صورت switch case پیاده کردیم. در ابتدا برنامه وارد استیت ریست میشود و در این استیت رجیسترها مقداردهی شده و یکی یکی وارد استیت های بعدی میشود که به ترتیب رجیستر کنترل و رجیستر فرکانس و رجیستر فاز روی باس قرار گیرد. بین استیت های کنترل و فرکانس و فاز رجیستر میانی در نظر گرفته شده که fsync را یک میکند و یک کلاک وقفه ایجاد میکند تا تراشه متوجه اتمام فرستاده شدن ۱۶ بیت بشود سپس رجیستر بعدی را دریافت کند. در واقع تبادل اطلاعات وقتی انجام میشود که کلاک مستر لبه پایین رونده میزند و مقدارfsync صفر است پس کلاک اسلیو هم طبق این رابطه تعریف میشود.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use ieee.numeric_std.all;
5
6  entity master is
7      Port ( m_clk : out  STD_LOGIC;
8             fsync : out  STD_LOGIC;
9             clk_input : in STD_LOGIC;
10            rst : in STD_LOGIC;
11            s_clk : out STD_LOGIC;
12            sdata : out STD_LOGIC);
13 end master;
14
15 architecture Behavioral of master is
16
17 signal control_register : STD_LOGIC_VECTOR (16 downto 0);
18 signal phase_register : STD_LOGIC_VECTOR (16 downto 0);
19 signal counter : integer := 0;
20 signal freq_counter : integer := 0;
21 signal freq_lsb , freq_msb : STD_LOGIC_VECTOR (16 downto 0);
22 signal m_clk_in : std_logic ;
23 signal fsync_in : std_logic := '0';
24
25 component clk is
26     Port ( clk_in : in  STD_LOGIC;
27            reset  : in std_logic;
28            freq   : out std_logic
29            );
30 end component;
```

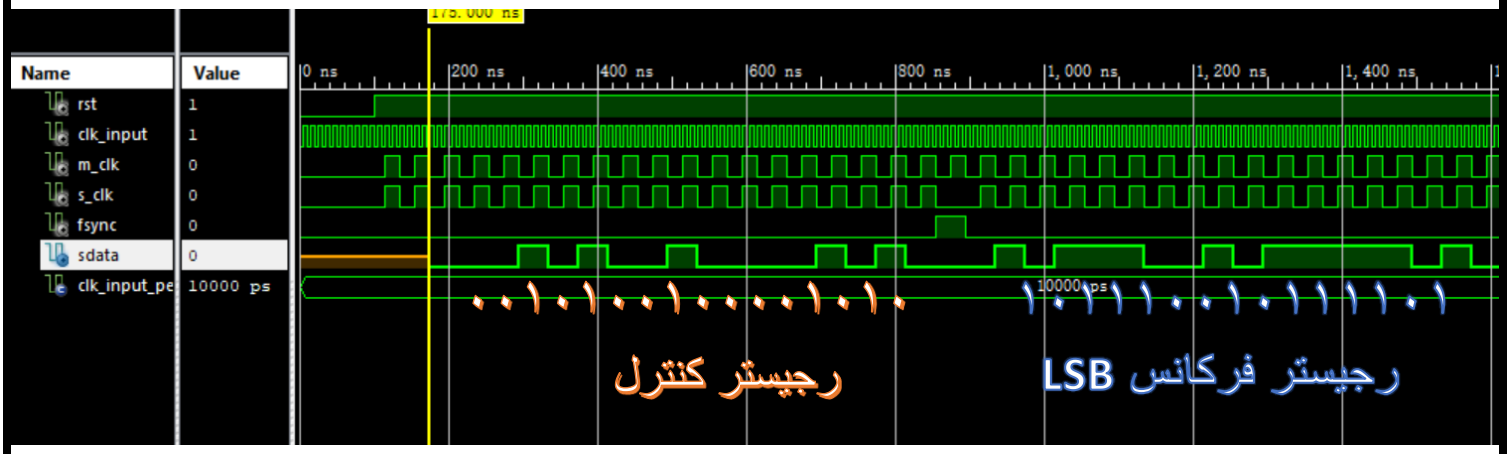
```
32
33  type t_state is (reset_state, control_state , middle1_state ,middle2_state , freq_state , phase_state );
34     signal state : t_state;
35
36  begin
37
38  j : CLK port map (
39                       clk_in=>clk_input,
40                       reset=>rst,
41                       freq=> m_clk_in
42                       );
43
44
45  process(m_clk_in , fsync_in)
46     begin
47     if (falling_edge (m_clk_in) )then
48        state <= reset_state ;
49        case state is
50
51              when reset_state =>    --reset state sets all the registers to its current value
52
53                 control_register <= "00010100100001010";
54                 freq_lsb <= "01011100101111101";   --lsb
55                 freq_msb <= "01000000000000101"; --msb
56                 phase_register <= "01100000000000000";
57                 fsync_in <= '0';
58                 state <= control_state;
59
60              when control_state =>
61
62                 fsync_in <= '0';
63                 for i in 0 to 15 loop
64                    control_register(i+1) <= control_register(i);
65                 end loop ;
66                 sdata <= control_register(16);
67
68                 if (counter <16) then
69                    counter <= counter + 1;
70                    state <= control_state;
71                 else
72                    counter <= 0;
73                    state<= middle1_state;
74                 end if;
75
76              when middle1_state =>
77
78                 fsync_in <= '1';
79                 state <= freq_state;
80
81              when freq_state =>
83                 fsync_in <= '0';
84                 if ( freq_counter = 0 )then
85                    for i in 0 to 15 loop
86                       freq_lsb(i+1) <= freq_lsb(i);
87                    end loop ;
88                    sdata <= freq_lsb(16);
89                    if (counter <16) then
90                       counter <= counter+1;
91                       state <= freq_state;
92                    else
93                       counter <= 0;
94                       freq_counter <= 1 ;
95                       state <= freq_state;
96                    end if;
97                 else
98                    for i in 0 to 15 loop
99                       freq_msb(i+1) <= freq_msb(i);
100                   end loop ;
101                   sdata <= freq_msb(16);
102                   if (counter <16) then
103                      counter <= counter+1;
104                      state <= freq_state;
105                   else
106                      counter <= 0;
107                      freq_counter <= 0 ;
108                      state<= middle2_state;
109                   end if;
110                end if;
111
```

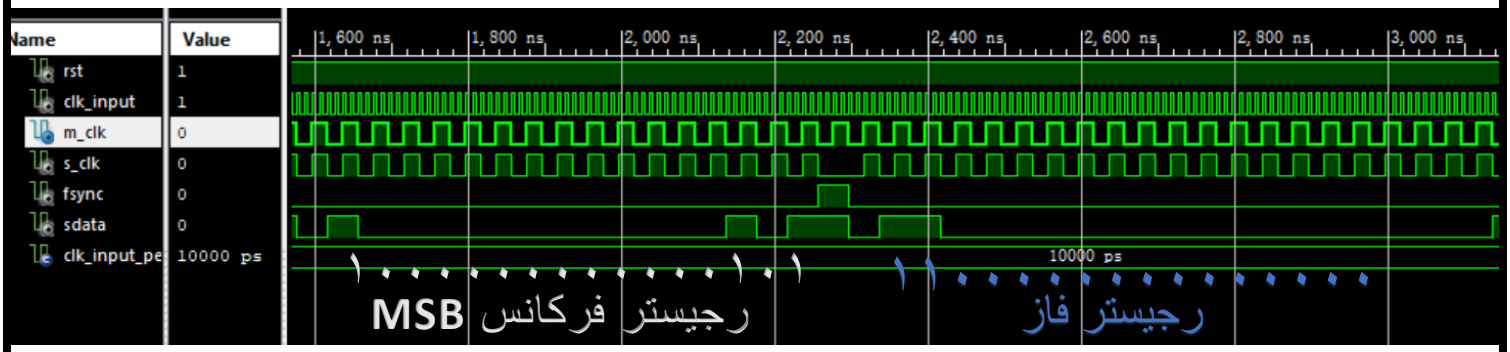
```
            when middle2_state =>

               fsync_in <= '1';
               fsync <= fsync_in  ;
               state <= phase_state;

            when phase_state =>

               fsync_in <= '0';
               for i in 0 to 15 loop
                  phase_register(i+1) <= phase_register(i);
               end loop ;
               sdata <= phase_register(16);
               if (counter <16) then
                  counter <= counter+1;
                  state <= phase_state;
               else
                  counter <= 0;
                  state <= reset_state;
               end if;
         end case;

      end if;
      s_clk <= m_clk_in and (not fsync_in) ;
      fsync <= fsync_in;
      m_clk <= m_clk_in;

   end process;

end Behavioral;
```

برای تست کردن برنامه بالا تست بنچ زیر نوشته شد که با ورودی دادن کلاک با دوره تناوب ۱۰ نانو ثانیه و مقدار ریست که همیشه ۱ است کار میکند.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY main_tbl IS
END main_tbl;

ARCHITECTURE behavior OF main_tbl IS

    -- Component Declaration for the Unit Under Test

    COMPONENT master
    PORT(
         m_clk : OUT  std_logic;
         s_clk : OUT  std_logic;
         fsync : OUT  std_logic;
         rst : IN  std_logic;
         clk_input : IN  std_logic;
         sdata : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal rst : std_logic := '0';
   signal clk_input : std_logic := '0';

   --Outputs
   signal m_clk : std_logic;
   signal fsync : std_logic;
   signal sdata : std_logic;
   signal s_clk : std_logic;

   -- Clock period definitions
   constant clk_input_period : time := 10 ns;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: master PORT MAP (
          m_clk => m_clk,
          fsync => fsync,
          rst => rst,
          clk_input => clk_input,
          s_clk => s_clk,
          sdata => sdata
        );
   -- Clock process definitions
   clk_input_process :process
   begin
      clk_input<= '0';
      wait for clk_input_period/2;
      clk_input <= '1';
      wait for clk_input_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      rst<='1';

      wait;
   end process;
END;
```

با اجرای کد تست بنچ نتایج زیر به دست آمد که طبق خواست مسئله است.





قسمت امتیازی :

در ابتدا با توجه به خواسته های مسئله رجیستر ها را مقدار دهی می کنیم .

تراشه اول : مقدار دهی رجیستر های این تراشه مطابق مقداردهی تراشه بخش ۱-۲ می باشد و همانطور که در بالا توضیح داده شد به صورت زیر مقدار دهی می شود .

کنترل رجیستر : بیت های زرد رنگ مقدار ثابت ۰ را داشتند. بیت های سبز رنگ را با توجه به خواسته های مسئله مشخص کردیم و بیت های قرمز رنگ دونت کر بودند که فرضی در نظر گرفتیم.

Control Register = ۰۰۱۰ ۱۰۰۱ ۰۰۰۰ ۱۰۱۰

رجیستر فرکانس :

Frequency register = ۰۰۰۱ ۰۱۱۱ ۱۰۰۱ ۰۱۱۱ ۱۱۰۱

این دیتا به صورت دو کلمه ۱۶ بیتی به شکل زیر به رجیستر فرکانس منتقل میشود.

بیت ۱۴ و۱۵ در سمت چپ باید به صورت ۱۰ ثابت باشد زیرا نشان میدهد دیتا در رجیستر FREQ۱ نوشته میشود. اول ۱۴ بیت کم ارزش به صورت زیر ارسال میشود.

۱۰۱۱ ۱۰۰۱ ۰۱۱۱ ۱۱۰۱

سپس ۱۴ بیت پر ارزش :

۱۰۰۰ ۰۰۰۰ ۰۰۰۰ ۰۱۰۱

رجیستر فاز:

چون در صورت سوال اختلاف فاز مشخص نشده است ما این مقدار را ۰ در نظر میگیریم.

phase register = ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

۱۱۰۰ ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

تراشه دوم: حال به مقدار دهی تراشه دوم که می خواهیم خروجی سینوسی با فرکانس ۸۰۰KHz داشته باشد می پردازیم .

کنترل رجیستر :

در این جا فرض می کنیم از ۱ frequency register و ۰ phase register استفاده شود پس در کنترل رجیستر بیت D۱۰ صفر و بیت D۱۱ یک است.

فرض می کنیم در ابتدا تراشه ریست شود بیت D۸ کنترل رجیستر را ۱ قرار می دهیم.

با توجه به این که دیتایی که در رجیستر فرکانس ریخته میشود بیشتر از ۱۴ بیت است پس باید از کل حافظه رجیستر فرکانس استفاده شود پس بیتD۱۳ کنترل رجیستر ۱ است تا دیتا با دو عملیات لود شدن در رجیستر ۲۸ بیتی نوشته شود. همچنین بیتD۱۲ کنترل رجیستر دونت کر میشود که در اینجا ۰ فرض می کنیم .

با توجه به این که میخواهیم سیگنال خروجی سینوسی باشد باید بیت D۵ و D۱ مقدار ۰ و بیت D۳ دونت کر می باشد که در اینجا ۱ فرض شده .

پس دیتا رجیستر به صورت زیر باید پر شود. بیت های زرد رنگ مقدار ثابت ۰ را داشتند. بیت های سبز رنگ را با توجه به خواسته های مسئله مشخص کردیم و بیت های قرمز رنگ دونت کر بودند که فرضی در نظر گرفتیم.

Control Register = ۰۰۱۰ ۱۰۰۱ ۰۰۰۰ ۱۰۰۰

رجیستر فرکانس:

برای تعیین کردن مقدار رجیستر فرکانس از فرمول زیر استفاده میکنیم با فرض این که فرکانس ورودی ۲۵ مگاهرتز و فرکانس سیگنال خروجی ۸۰۰کیلوهرتز است.

$$FreqReg = \frac{f_{OUT} \times 2^{28}}{f_{MCLK}}$$

عدد دسیمال۸۵۸۹۹۳۴.۵۹۲ به دست می آید که با تغییر مبنا عدد باینری زیر به دست می آید.

Frequency register =۱۰۰۰۰۰۱۱۰۰۰۱۰۰۱۰۰۱۱۰۱۱۱۱

این دیتا به صورت دو کلمه ۱۶ بیتی به شکل زیر به رجیستر فرکانس منتقل میشود.

بیت ۱۴ و۱۵ در سمت چپ باید به صورت ۱۰ ثابت باشد زیرا نشان میدهد دیتا در رجیستر FREQ۱ نوشته میشود. اول ۱۴ بیت کم ارزش به صورت زیر ارسال میشود.

۱۰۰۱ ۰۰۱۰ ۰۱۱۰ ۱۱۱۱

سپس ۱۴ بیت پر ارزش ارسال میشود.

۱۰۰۰ ۰۰۱۰ ۰۰۰۰ ۱۱۰۰

رجیستر فاز:

چون در صورت سوال اختلاف فاز مشخص نشده است ما این مقدار را ۰ در نظر میگیریم.
طبق جدول زیر عدد ۰ باید به صورت زیر به رجیستر فاز داده شود.

phase register = ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

۱۱۰۰ ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

Table 12. Phase Register Bits

D15	D14	D13	D12	D11	D0
1	1	0	X	MSB 12 PHASE0 bits	LSB
1	1	1	X	MSB 12 PHASE1 bits	LSB

تراشه سوم : حال به مقدار دهی تراشه سوم که می خواهیم خروجی سینوسی با بیشترین فرکانس خروجی را داشته باشد می پردازیم .

کنترل رجیستر :

در این جا همانند تراشه ی قبل مقادیر رجیستر را مشخص می کنیم.

Control Register = ۰۰۱۰ ۱۰۰۱ ۰۰۰۰ ۱۰۰۰

رجیستر فرکانس:

حداکثر فرکانس خروجی این تراشه در حالتی است که تمام ۲۸ بیت ۱ باشد که با توجه فرمول پایین و محاسبات فرکانس خروجی حدودا ۲۵MHz می باشد .

$$FreqReg = \frac{f_{OUT} \times 2^{28}}{f_{MCLK}}$$

Frequency register = ۱۰۱۱ ۱۱۱۱ ۱۱۱۱ ۱۱۱۱ ۱۰۱۱ ۱۱۱۱ ۱۱۱۱ ۱۱۱۱

این دیتا به صورت دو کلمه ۱۶ بیتی به شکل زیر به رجیستر فرکانس منتقل میشود.

اول ۱۴ بیت کم ارزش به صورت زیر ارسال میشود و سپس ۱۴ بیت پر ارزش.

۱۰۱۱ ۱۱۱۱ ۱۱۱۱ ۱۱۱۱

۱۰۱۱ ۱۱۱۱ ۱۱۱۱ ۱۱۱۱

تعیین مقدار رجیستر فاز:

چون در صورت سوال اختلاف فاز مشخص نشده است ما این مقدار را ۰ در نظر میگیریم.
طبق جدول زیر عدد ۰ باید به صورت زیر به رجیستر فاز داده شود.

phase register = ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

۱۱۰۰ ۰۰۰۰ ۰۰۰۰ ۰۰۰۰

Table 12. Phase Register Bits

D15	D14	D13	D12	D11	D0
1	1	0	X	MSB 12 PHASE0 bits	LSB
1	1	1	X	MSB 12 PHASE1 bits	LSB

برای نوشتن برنامه این بخش نیاز به تعریف سه خروجی cs۱ و cs۲ و cs۳ داشتیم که یک شدن هرکدام از آنها به معنی انتخاب ماژول است. همچنین طبق خواسته های مسئله برای هر ماژول مقادیر فرکانس متفاوت است. طبق برنامه زیر در استیت ریست شرطی تعریف شده که چک میکنیم کدام یک از cs۱ یا cs۲ یا cs۳ یک است که رجیستر های مربوط به آن ماژول روی باس spi قرار گیرد. بقیه قسمت های کد تقریبا شبیه قسمت قبل است زیرا تغییری در نحوه پر شدن رجیستر ها صورت نمیگیرد .به همین علت از قرار دادن قسمت های تکراری در گزارش کار صرف نظر کردیم.

```
54      case state is
55
56            when reset_state =>    --reset state sets all the registers to its current value
57               if (cs1_s='1') then              --9khz triangle
58                  control_register <= "0001010010000010";
59                  freq_lsb <= "0101110010111101";   --lsb
60                  freq_msb <= "0100000000000101"; --msb
61                  phase_register <= "0110000000000000";
62                  fsync_in <= '0';
63                  cs1_s<='0';
64                  cs2_s<='1';
65                  state <= control_state;
66               elsif (cs2_s='1') then           --800khz sinusoid
67                  control_register <= "0001010010000010";
68                  freq_lsb <= "0100100110110111";   --lsb
69                  freq_msb <= "0100000110000100"; --msb
70                  phase_register <= "0110000000000000";
71                  fsync_in <= '0';
72                  cs2_s<='0';
73                  cs3_s<='1';
74                  state <= control_state;
75               elsif (cs3_s='1') then              --max frequency = 24mhz
76                  control_register <= "0001010010000010";
77                  freq_lsb <= "0111111111111111";   --lsb
78                  freq_msb <= "0111111111111111"; --msb
79                  phase_register <= "0110000000000000";
80                  fsync_in <= '0';
81                  cs3_s<='0';
82                  cs1_s<='1';
83                  state <= control_state;
84               end if;
```

همان گونه که مشاهده میشود با یک بودن cs۱ رجیستر های مربوط به ماژول یک روی باس قرار میگیرند. و به همین ترتیب رجیسترهای مربوط به مازول های دیگر روی باس قرار میگیرد.