دانشکده مهندسی برق دانشگاه صنعتی امیرکبیر

درس مدار منطقی

پروژه ضرب کننده

نام : محمد حسین کاشانکی – علی امیری راد

نام استاد: دکتر شریعتمدار مرتضوی

در این پروژه قصد داریم ضرب کننده ای بسازیم که 16 بیت را در 16 بیت ضرب کند و در 32 بیتی خروجی ذخیره کند. ورودی اول 16 بیت کم ارزش a و ورودی دوم 16 بیت کم ارزش b است. مضروب فیه در رجیستر multiplier قرار می گیرد و مضروب در 16 بیت کم ارزش رجیستر multiplicand همچنین یک adder برای جمع product (خروجی رجیستر product) و multiplicand داریم و با استفاده از یک process حساس به لبه بالا رونده کلاک کنترل می کنیم که در هر مرحله add&shift صورت گیرد.

برای این منظور باید یک state machine که شامل دو state : idle , multiply است را پیاده سازی کنیم. در idle state کاری انجام نمی شود و منتظر می مانیم تا start 1 شود. هنگامی که start یک کلاک یک می شود نشان این است که a,b ورودی معتبرند و باید عملیات ضرب آغاز شود؛ بنابراین به state multiply می رویم و رجیستر های multiplier , multiplicand را طبق روش گفته شده پر کنیم. همچنین اگر state = 1 دریافت نشد همچنان در idle باقی می مانیم :

```
23  begin
24      in_product <= multiplicand + product ;
25      y <= product;
26  process (clk)
27  begin
28   if rising_edge(clk) then
29       case state is
30           when idle =>
31                   ready <= '0';
32                   if start = '1' then
33                       multiplier <= b(15 downto 0);
34                       multiplicand <= ("0000000000000000") & a(15 downto 0);
35                       product <= (others =>'0');
36                       state <= multiply;
37                   else
38                       state <= idle;
39                   end if;
40           when multiply =>
41                    if counter <17 then
42                        multiplier <= '0' & multiplier(15 downto 1);
43                        multiplicand <= multiplicand(30 downto 0) & '0';
44                        counter <= counter +1;
45                        if multiplier(0) = '1' then
46                           product <= in_product;
```

در state multiply باید به اندازه 16 کلاک هر بار multiplier به راست و multiplicand به چپ شیفت بخورد (تعداد اجرا با استفاده از یک حلقه for و یک سیگنال counter انجام می شود) و سپس با استفاده از بیت کم ارزش multiplier تعیین شود که آیا حاصل جمع در product ریخته شود یا خیر. حاصل جمع خروجی adder است که با نام in_product مشخص شده است.

در انتها هنگامی که 16 کلاک تمام شد باید یک کلاک ready = 1 داشته باشیم و به state idle برویم و منتظر دو عدد و سیگنال start بعدی باشیم.

```
40          when multiply =>
41                  if counter <17 then
42                      multiplier <= '0' & multiplier(15 downto 1);
43                      multiplicand <= multiplicand(30 downto 0) & '0';
44                      counter <= counter +1;
45                      if multiplier(0) = '1' then
46                         product <= in_product;
47                      end if;
48                  elsif counter = 17 then
49                     ready <= '1';
50                     counter <= 1;
51                     state <= idle;
52                  end if;
53      end case;
54   end if;
55  end process;
56  end Behavioral;
```

برای نوشتن تست بنچ 3 بار به طور متوالی ضرب را انجام می دهیم : فرآیند یک تست به این صورت است :

ابتدا a,b را تعیین می کنیم و یک کلاک بعد start را به مدت یک کلاک 1 می کنیم. سپس به اندازه کافی (به طور مثال 20 کلاک) صبر می کنیم تا خروجی معتبر شود و سیگنال ready یک شود.

```
38      a <= x"00001234";
39      b <= x"00005678";
40          clk <= '1';
41          wait for clkperiod/2;
42          clk <= '0';
43          wait for clkperiod/2;
44      start <= '1';
45          clk <= '1';
46          wait for clkperiod/2;
47          clk <= '0';
48          wait for clkperiod/2;
49      start <= '0';
50      for i in 1 to 20 loop
51          clk <= '1';
52          wait for clkperiod/2;
53          clk <= '0';
54          wait for clkperiod/2;
55       end loop;
```

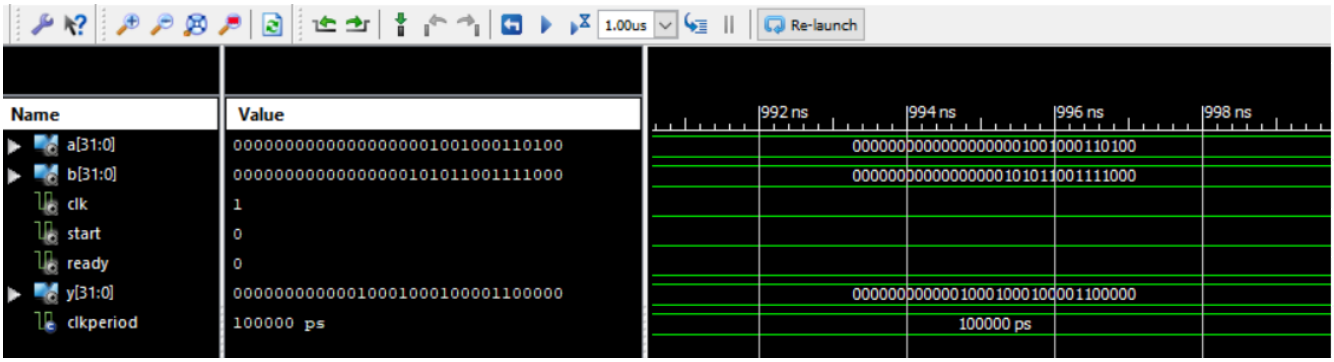
سپس می توان تست دوم و سوم را اعمال کرد.

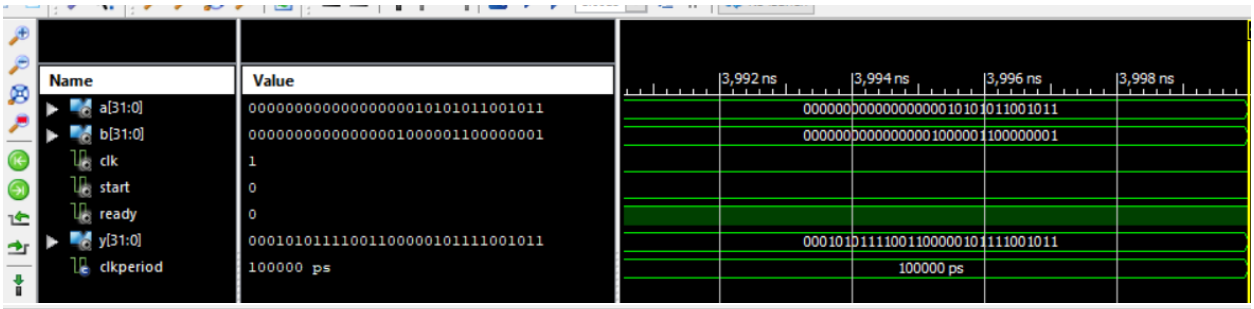
در شبیه سازی نهایی مشاهده می شود که مطابق انتظار خروجی و ready آماده می شوند : (اعداد مبنای 16)

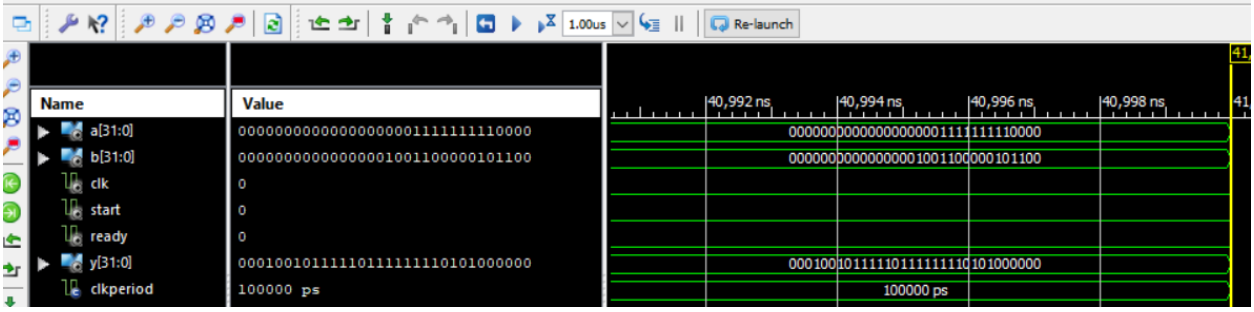
1234 * 5678 = 6260060

8301 * 2acb = 15e60bcb

982c * 1ff0 = 12fbfd40



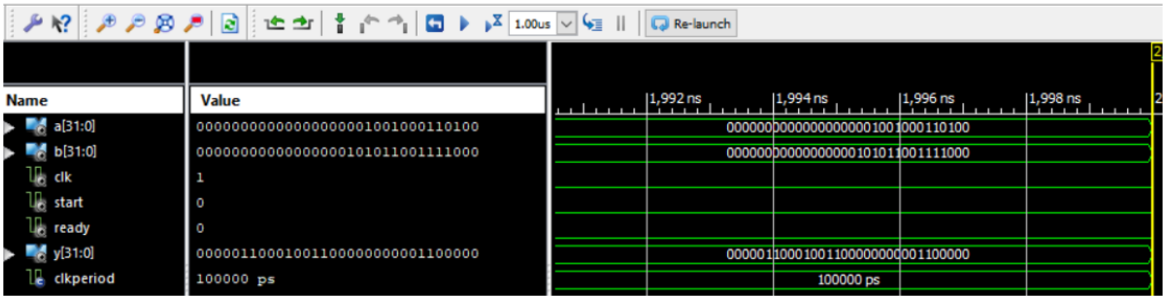




00001234 00005678

1001000110100 101011001111000 11000100110000000000001100000

4660 22136



1111111110000 1001100000101100 10010111110111111110101000000

8176 38956