CONTADORES

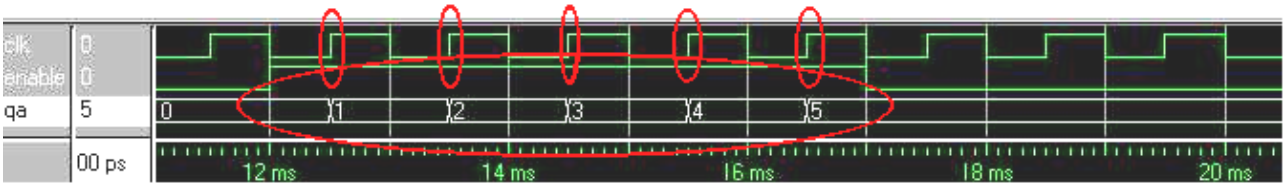
```
entity conta1 is
port (clk,enable:in bit;
     qa: out integer range 0 to 15);
end conta1;

architecture algoritmo of conta1 is
begin
     process (clk)
     variable cnt :  integer range  0 to 15;
     begin
          if (clk='1') then
               if enable = '1' then
                    cnt:=cnt+1;
               end if;
          end if;
          qa<=cnt;
      end process;
end algoritmo;

--para la simulación la señal de reloj es periódica y tiene
una frecuencia de 1 Khz. La Señal enable se habilita durante
5 ms a partir del tiempo 12 ms
```

PARA ESTO CREAMOS EL SIGUIENTE ARCHIVO DE SIMULACIÓN

```
ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

     COMPONENT conta1
     PORT(
          clk : IN bit;
          enable : IN bit;
          qa : OUT integer range 0 to 15);
     END COMPONENT;

     SIGNAL clk :  bit;
     SIGNAL enable :  bit;
     SIGNAL qa :  integer range 0 to 15;

BEGIN

     uut: conta1 PORT MAP(
          clk => clk,
          enable => enable,
```

```
            qa => qa
      );

   tb : PROCESS
   BEGIN
     clk<= (not clk) after 0.5 ms;
   wait for 0.5 ms;
     END PROCESS;
 tba : PROCESS
   BEGIN
     enable<='0';
     wait for 12 ms;
     enable<='1';
     wait for 5 ms;
     enable<='0';
     wait; -- will wait forever
   END PROCESS;
END;
```

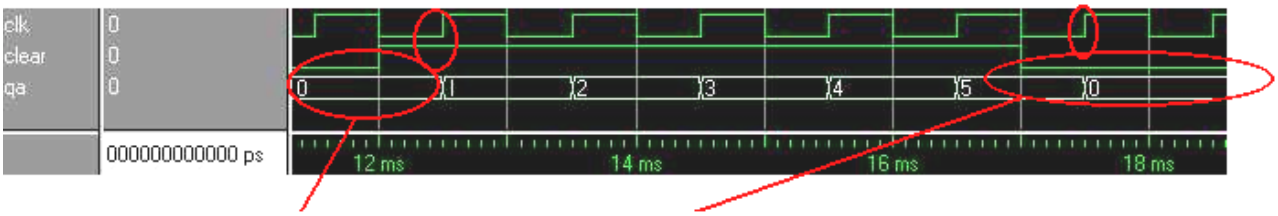
Y obtenemos el siguiente gráfico



aquí vemos que la salida se incrementa cuando esta activo ='1' el valor de enable, y además se sincroniza con cada flanco de subida del reloj, por eso solo contará hasta el número 5, y como ya no esta activo el enable, dejará de contar, quedando el último valor en la salida Q

```
entity conta2 is
port(clk,clear:in bit;
          qa: out integer range 0 to 15);
end conta2;

architecture algoritmo of conta2 is
begin
     process (clk)
     variable cnt :  integer range  0 to 15;
     begin
          if (clk='1') then
               if clear = '0' then
                    cnt:=0;
               else cnt:=cnt+1;
               end if;
          end if;
```

```
          qa<=cnt;
      end process;
end algoritmo;
--IMPLEMENTAMOS EL SIGUIENTE ARCHIVO DE SIMULACIÓN
ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

     COMPONENT conta2
     PORT(
          clk : IN bit;
          clear : IN bit;
          qa : OUT integer range 0 to 15);
     END COMPONENT;

     SIGNAL clk :  bit;
     SIGNAL clear :  bit;
     SIGNAL qa :  integer range 0 to 15;

BEGIN

     uut: conta2 PORT MAP(
          clk => clk,
          clear => clear,
          qa => qa
     );


-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
     clk<= (not clk) after 0.5 ms;
   wait for 0.5 ms;
     END PROCESS;
 tba : PROCESS
   BEGIN
     clear<='0';
     wait for 12 ms;
     clear<='1';
     wait for 5 ms;
     clear<='0';
     wait; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```



la salida qa será cero, siempre y cuando la señal de clear sea igual a 0, además si el reloj está en flanco de subida, recién ahí la salida cambiará de estado.

Las diferencias entre ambos contadores son, que uno usa la señal enable para decidir cuando va a contar y cuando no, en cambio la señal clear, sirve que para con un nivel bajo de está, la salida sea cero, (siempre sincronizada con el reloj) y mientras no lo sea, seguirá su cuenta.

6.- Simular el siguiente circuito

```
ENTITY pipo IS
PORT( d   :IN BIT_VECTOR (7 DOWNTO 0);
          clk:IN BIT;
          q  :OUT BIT_VECTOR (7 DOWNTO 0));
END pipo;
ARCHITECTURE ALGORITMO OF PIPO IS
BEGIN
     PROCESS
          BEGIN
               WAIT UNTIL clk='1';
               q<=d;
     END PROCESS;
END ALGORITMO;
--NOTA : DEBEN CARGARSE COMO MINIMO 10 DATOS

Para ello creamos el archivo simulador

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

     COMPONENT pipo
     PORT(
          d : IN bit_vector(7 downto 0);
          clk : IN bit;
          q : OUT bit_vector(7 downto 0)
          );
     END COMPONENT;

     SIGNAL d :  bit_vector(7 downto 0);
     SIGNAL clk :  bit;
```

```
     SIGNAL q :  bit_vector(7 downto 0);

BEGIN

     uut: pipo PORT MAP(
          d => d,
          clk => clk,
          q => q
     );


-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
     clk<= (not clk ) after 10 ns;
      wait for 10 ns;
   END PROCESS;
   tb1 : PROCESS
   BEGIN
     d<= "00001001"; wait for 10 ns;
     d<= "00001111"; wait for 10 ns;
     d<= "10001001"; wait for 10 ns;
     d<= "11111001"; wait for 10 ns;
     d<= "11111111"; wait for 10 ns;
     d<= "00111001"; wait for 20 ns;
     d<= "11110000"; wait for 20 ns;
     d<= "10001111"; wait for 20 ns;
     d<= "10000001"; wait for 20 ns;
     d<= "00000000"; wait for 20 ns;
      wait; -- will wait forever
   END PROCESS;


-- *** End Test Bench - User Defined Section ***

END;
```

Y revisamos la gráfica

Aquí vemos que el último dato que coincide con el flanco de subida del reloj es el que realmente pasa, no importando si el dato cambié antes, durante o despues, o en el intervalo entre flancos del reloj, siempre pasará el que detecte el FLANCO DE SUBIDA DEL RELOJ