INSTITUTE OF ENGINEERING AND TECHNOLOGY DAVV INDORE

NAME- SHIVAM KUMAR SURARIYA

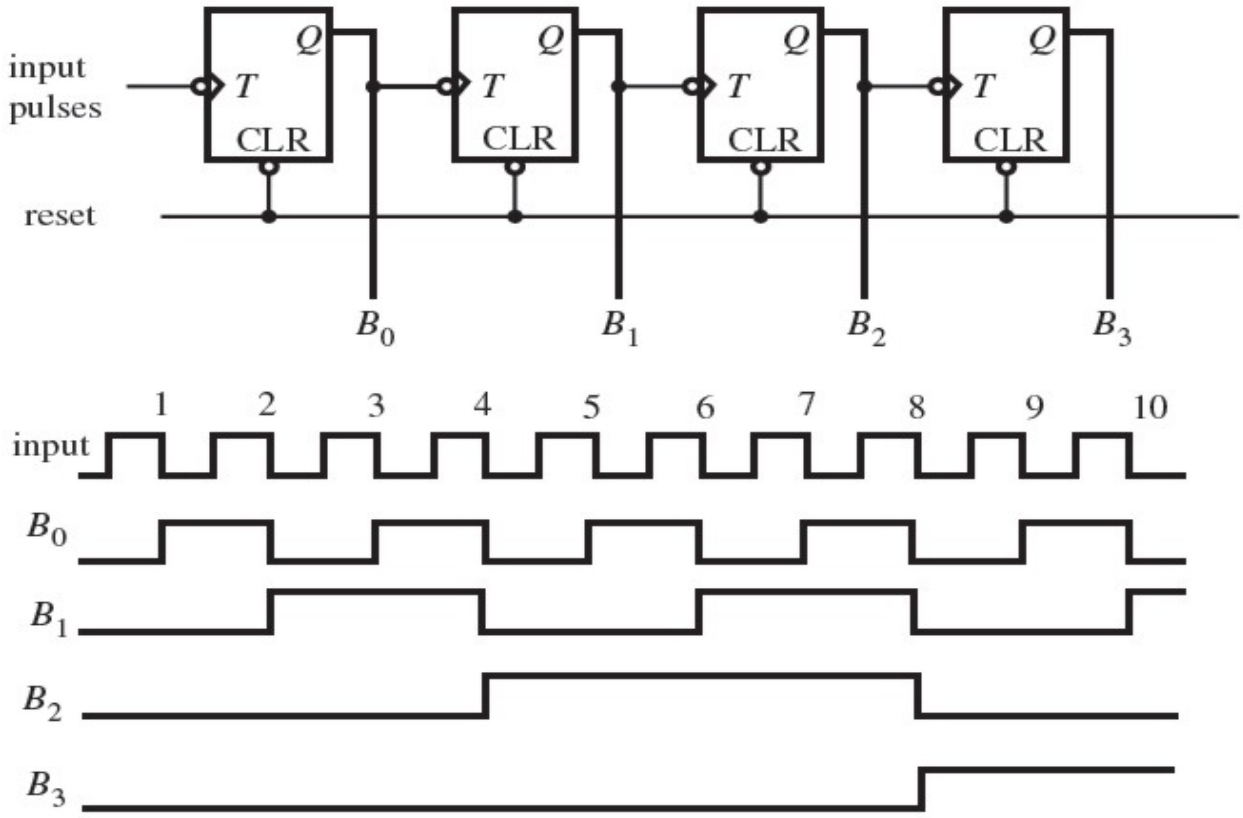
SUBMITTED TO –MR. VAIBHAV NEMA SIR

ELETRONIC AND INSTRUMENTATION ENG.

SUBJECT- CIRCUIT DESIGN USING VHDL

ASYNCRONOUS 4 BIT COUNTER

Asynchronous 4-bit UP counter



. It is capable of counting numbers from 0 to 15. The clock inputs of all flip flops are cascaded and the D input (DATA input) of each flip flop is connected to a state output of the flip flop.

That means the flip flops will toggle at each active edge or positive edge of the clock signal. The clock input is connected to first flip flop. The other flip flops in counter

receive the clock signal input from Q' output of previous flip flop. The output of the first flip flop will change, when the positive edge on clock signal occurs.

In the asynchronous 4- bit up counter, the flip flops are connected in toggle mode, so when the when the clock input is connected to first flip flop FF0, then its output after one clock pulse will become 20.

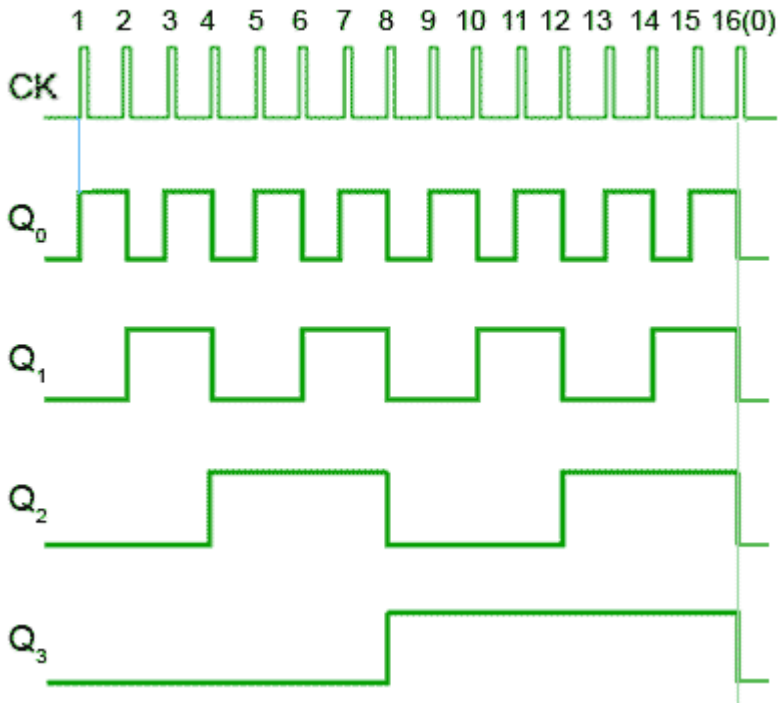
The rising edge of the Q output of each flip flop triggers the clock input of its next flip flop. It triggers the next clock frequency to half of its applied input. The Q outputs of every individual flip flop (Q0, Q1, Q2, Q3) represents the count of the 4 bit UP counter such as 20 (1) to 23 (8).

Working of asynchronous up counter is explained below,

Let us assume that the 4 Q outputs of the flip flops are initially 0000. When the rising edge of the clock pulse is applied to the FF0, then the output Q0 will change to logic 1 and the next clock pulse will change the Q0 output to logic 0. This means the output state of the clock pulse toggles (changes from 0 to1) for one cycle.

As the Q' of FF0 is connected to the clock input of FF1, then the clock input of second flip flop will become 1. This makes the output of FF1 to be high (i.e. Q1 = 1), which indicates the value 20. In this way the next clock pulse will make the Q0 to become high again.

So now both Q0 and Q1 are high, this results in making the 4 bit output 11002. Now if we apply the fourth clock pulse, it will make the Q0 and Q1 to low state and toggles the FF2. So the output Q2 will become 0010¬2. As this circuit is 4 bit up counter, the output is sequence of binary values from 0, 1, 2, 3 … .15 i.e. 00002 to 11112 (0 to 1510).



Timing diagram of Asynchronous counter

UP Counting

If the UP input and down inputs are 1 and 0 respectively, then the NAND gates between first flip flop to third flip flop will pass the non inverted output of FF 0 to the clock input of FF 1. Similarly, Q output of FF 1 will pass to the clock input of FF 2. Thus the UP /down counter performs up counting.

DOWN Counting

If the DOWN input and up inputs are 1 and 0 respectively, then the NAND gates between first flip flop to third flip flop will pass the inverted output of FF 0 to the clock input of FF 1. Similarly, Q output of FF 1 will pass to the clock input of FF 2. Thus the UP /down counter performs down counting.

The up/ down counter is slower than up counter or a down counter, because the addition propagation delay will added to the NAND gate network

Advantages

- Asynchronous counters can be easily designed by T flip flop or D flip flop.
- These are also called as Ripple counters, and are used in low speed circuits.
- They are used as Divide by- n counters, which divide the input by n, where n is an integer.
- Asynchronous counters are also used as Truncated counters. These can be used to design any mod number counters, i.e. even Mod (ex: mod 4) or odd Mod (ex: mod3).

Disadvantages

- Sometimes extra flip flop may be required for “Re synchronization”.
- To count the sequence of truncated counters (mod is not equal to 2n), we need additional feedback logic.
- While counting large number of bits, the propagation delay of asynchronous counters is very large.
- For high clock frequencies, counting errors may occur, due to propagation delay.

Applications of Asynchronous Counters

- Asynchronous counters are used as frequency dividers, as divide by N counters.
- These are used for low power applications and low noise emission.
- These are used in designing asynchronous decade counter.
- Also used in Ring counter and Johnson counter.
- Asynchronous counters are used in Mod N ripple counters. EX: Mod 3, Mod 4, Mod 8, Mod 14, Mod 10 etc.

VHDL CODE

```
library ieee;
use ieee.std_logic_1164.all;

entity ctr3 is
  port (
    input:    in std_logic_vector (2 downto 0);
    load:     in std_logic;
    reset:    in std_logic;
    enable:   in std_logic;
    clk:      in std_logic;
    q:        out std_logic_vector (2 downto 0)
  );
```

```
end entity;

architecture behave of ctr3 is

  signal cntin,cntd,cntq: std_logic_vector
(2 downto 0);

  begin

cntinc:      -- 3 bit incrementer
  process(cntq)
    begin
  cntin(0) <= not cntq(0);
  cntin(1) <= cntq(1) xor cntq(0);
  cntin(2) <= cntq(2) xor (cntq(0) and
cntq(1));
 end process;

cntdin:
  cntd <= (others =>'0') when reset = '1' else
       cntq            when reset = '0' and
enable = '0' else
       input           when reset = '0' and
```

```
    enable = '1' and
                        load = '1' else
           cntin              when reset = '0' and
    enable = '1' and
                        load = '0' else
           (others => 'X');


    cntreg:
      process
        begin
        wait until clk'event and clk = '1';
        cntq <= cntd;
      end process;


    cntout:
      q <= cntq;


end architecture;
```

TEST BENCH

```
library ieee;
use ieee.std_logic_1164.all;

entity ctr3_tb is
end entity;

architecture foo of ctr3_tb is
   signal input:          std_logic_vector (2 downto 0);
   signal load:     std_logic;
   signal reset:     std_logic;
   signal enable:    std_logic;
```

```
   signal clk:      std_logic := '0';
   signal q:         std_logic_vector (2 downto
0);
begin
CLOCK:
   process
   begin
      wait for 5 ns;
      clk <= not clk;
      if NOW > 100 ns then
         wait;
      end if;
   end process;
DUT:
   entity work.ctr3
      port map (
         input  => input,
         load   => load,
         reset  => reset,
         enable => enable,
         clk    => clk,
```

```
        q     => q
    );
STIMULUS:
  process
  begin
    wait for 6 ns;
    input <= "001";
    load <= '0';
    reset <= '1';
    enable <= '0';
    wait for 10 ns;
    reset <= '0';
    wait for 10 ns;
    enable <= '1';
    wait for 30 ns;
    load <= '1';
    wait for 20 ns;
    load <= '0';
    enable <= '0';
    wait;
```

```
    end process;
end architecture;
```

WAVE FORM