SR, D, JK ve T Flip-Flop'lar (FF), temel bellek elemanlarıdır ve dijital devrelerde bir bitlik bilgi depolama işlevi görürler. Bu elemanlar, saat (clock) sinyali ile tetiklenir ve sayıcılar (counter), kaydediciler (register) gibi daha karmaşık devrelerin temel yapı taşlarını oluşturur. Dijital devrelerin yapı taşı olarak Flip-Flop'lar, FPGA içindeki daha büyük devre yapılarının inşasında kritik bir rol oynar.

Ancak, bu yapıları her projede manuel olarak tasarlamamız gerekmez. Vivado gibi sentez araçları, yazdığımız koda göre ihtiyaç duyulan Flip-Flop yapılarını otomatik olarak oluşturur ve FPGA içinde kullanılacak toplam FF sayısını raporlar. Flip-Flop kullanımı, tasarımın yapısına göre farklılık gösterebilir; örneğin, aynı işlevi gerçekleştiren iki farklı kodun if veya when blokları arasındaki farklar bile gereken FF sayısını değiştirebilir.

Karmaşık projelerde kaynak yönetimi önem kazandığından, daha fazla iş yapabilmesi için FPGA işlemcisinin verimli kullanılması hedeflenir ve gereksiz Flip-Flop kullanımı azaltılır. Bu yüzden tasarıma başlarken kaynakları en verimli şekilde kullanacak stratejiler belirlemek gereklidir. Büyük projelerde bu tür optimizasyonlar güç tüketimini ve kaynak kullanımını doğrudan etkilerken, basit projelerde bu kaygı daha azdır; çünkü çoğu FPGA, basit uygulamaların ihtiyaç duyduğu FF miktarını fazlasıyla karşılayabilir.

Bununla birlikte, farklı flip-flop türleri belirli işlevlerde ön plana çıkar: D Flip-Flop'lar veri saklamada yaygınken, JK ve T Flip-Flop'lar sayıcı devrelerde sıklıkla tercih edilir. Bu farklılıklar, tasarımın performansını ve işlevselliğini doğrudan etkileyen önemli detaylardır. Vivado gibi sentez araçları, gereksiz flip-flopları optimize ederek kaynak verimliliğini artırır, bu da karmaşık devrelerde yerden ve güçten tasarruf sağlar.

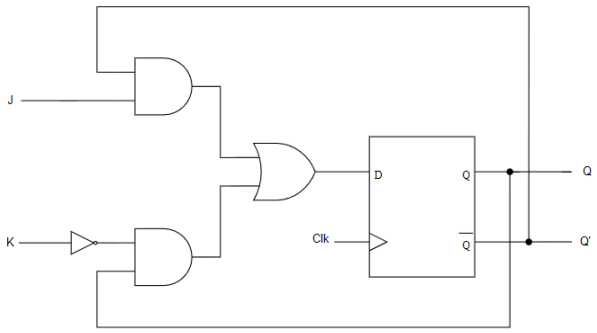
Burada sizlere JK FF leri anlatacağım ve bununla ilgili basit bir VHDL kodunu da yazacağım.

JK Flip-Flop

JK Flip-Flop, SR ve D Flip-Flop türlerinin özelliklerini birleştiren bir yapıdır ve sıralı devrelerin temel taşlarından biridir. Flip-Flop'lar, bellek elemanları olarak bir bitlik veri saklar ve sayıcılar veya kaydediciler gibi daha büyük devrelerin inşasında kullanılır. JK Flip-Flop'lar, veri saklama işlevi gören temel devre elemanları arasında yer alır. Özellikle, JK Flip-Flop'un avantajı, SR Flip-Flop'taki belirsiz (yasaklı) durumu ortadan kaldırması ve hem sıfırdan bire hem de birden sıfıra geçişleri sağlamasıdır; bu özellikleri sayesinde, sayıcı ve kaydedici devrelerde tercih edilen bir flip-flop türüdür.

JK Flip-Flop devre gösterimini aşağıda görebilirsiniz.

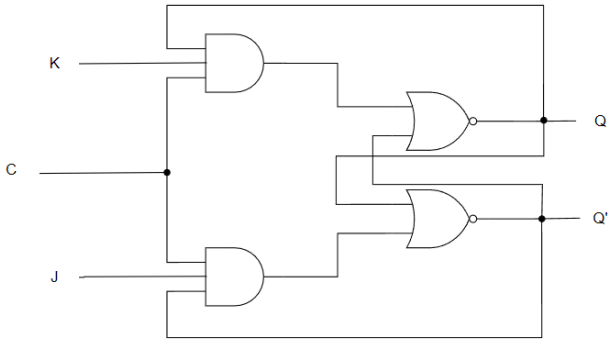
Devre gösterimi



Devrede de göründüğü gibi JK Flip-Flop'u oluşturmak için de devrenin daha derinlerinde yer alan kapılar kullanılmıştır. Burada durumlar oluşturulurken devre gösteriminden faydalanılabilir. JK Flip-Flop'unun doğruluk tablosunu aşağıda görebilirsiniz.

Q ile Q' içeride aslında birbirlerine bağlılardır. Bu devre gösteriminin biraz daha açılmış halini de aşağıda görebilirsiniz.

Detaylı devre



Doğruluk tablosu

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	Q'

Doğruluk tablosunun gösterimi bir üstte bulunan devre ile oluşmaktadır. Devrede bulunan J ve K girişlerine tabloda gözüken değerleri verdiğimiz zaman Q çıkışında neleri göreceğimizi detaylı olarak da anlayabiliriz.

Şimdi de yukarıda edinilen bilgiler doğrultusunda JK Flip-Flop'un VHDL kodunu yazabiliriz.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;


entity jk is
    port(
            J_i     : in std_logic;
            K_i     : in std_logic;
            clk_i   : in std_logic;
            rst_i   : in std_logic;
            Q_o     : out std_logic
        );
    end jk;

architecture Behavioral of jk is
    signal s_Q_int  : std_logic := '0';

begin
    process (clk_i, rst_i)
    begin
        if (rst_i = '1') then
            s_Q_int <= '0';
        elsif rising_edge (clk_i) then
            s_Q_int <= '0' when (J_i = '0' and K_i = '1') else
                       '1' when (J_i = '1' and K_i = '0') else
                      not s_Q_int when (J_i = '1' and K_i = '1') else
                       s_Q_int;
        end if;
    end process;
    Q_o <= s_Q_int;

end Behavioral;
```

Kodu detaylı olarak incelersek;

```vhdl
entity jk is
    port(
            J_i     : in std_logic;
            K_i     : in std_logic;
            clk_i   : in std_logic;
            rst_i   : in std_logic;
            Q_o     : out std_logic
        );
    end jk;
```

Burada bizim JK Flip-Flop için gerekli olan giriş çıkışlarımızı belirledik.

- **J** ve **K** girişleri flip-flop'un durumunu belirlemek için kullanılır.
- **clk** giriş sinyali, flip-flop'un saat sinyali olup tetiklenmeyi sağlar.
- **rst** girişi, flip-flop'un durumunu sıfırlamak için kullanılan asenkron reset sinyalidir.

- **Q** çıkışı ise flip-flop'un durumunu yansıtır.

```
architecture Behavioral of jk is
    signal s_Q_int  : std_logic := '0';
```

Bu bölümde, devrenin **architecture** kısmı başlar. Bu kısımda flip-flop'un iç yapısı tanımlanır yani hangi davranışları sergileyeceği bu kısımda bizlere söylenir:

- **Q_int** sinyali, flip-flop'un iç durumunu temsil eder. Başlangıç değeri 0 olarak atanmıştır.

```
begin
    process (clk_i, rst_i)
```

begin ile mimariyi başlatırız ve sonrasında process bloğu içerisinde hangi sinyallerin değişikliğini inceleyeceğimizi belirtiriz.

- **process(clk, rst)**: Process bloğu, **clk** ve **rst** sinyallerindeki değişiklikleri izler.

```
    begin
        if (rst_i = '1') then
            s_Q_int <= '0';
        elsif rising_edge (clk_i) then
            s_Q_int <= '0' when (J_i = '0' and K_i = '1') else
                       '1' when (J_i = '1' and K_i = '0') else
                       not s_Q_int when (J_i = '1' and K_i = '1') else
                       s_Q_int;
        end if;
    end process;
    Q_o <= s_Q_int;

end Behavioral;
```

Tekrardan **begin** komutu ile bu sefer hangi durumlarda hangi davranışları sergilemesi gerektiği yazılır.

- İlk kontrol olarak rst = '1' koşulu sorgulanır. Eğer rst sinyali aktif (yani 1) durumdaysa, **asenkron reset** olarak çalışır ve Q_int sıfırlanır. Bu durumda çıkış Q da sıfırlanır.

rst sinyali aktif değilse, clk sinyali yükselen kenarda tetiklendiğinde (rising_edge) J ve K giriş değerlerine göre **Q_int** güncellenir:

- **J = 0 ve K = 1** ise, **Reset Durumu**: Q_int sıfırlanır.
- **J = 1 ve K = 0** ise, **Set Durumu**: Q_int bir olur.
- **J = 1 ve K = 1** ise, **Toggle Durumu**: Q_int değeri tersine çevrilir.
- **J = 0 ve K = 0** ise, **Durum Korunur**: Q_int aynı kalır.
- Son olarak, **Q** çıkışı **Q_int** sinyaline atanır, yani **Q_int**'in değeri **Q** çıkışına aktarılır.

Şimdi bu yazılan kodun doğruluğunu Testbench ile kontrol edelim.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity tb_jk is
end tb_jk;

architecture Behavioral of tb_jk is

component jk is
    port(
            J_i     : in std_logic;
            K_i     : in std_logic;
            clk_i   : in std_logic;
            rst_i   : in std_logic;
            Q_o     : out std_logic
        );
end component;

signal  SJ   : std_logic := '0';
signal  SK   : std_logic := '0';
signal  Sclk : std_logic := '0';
signal  Srst : std_logic := '0';
signal  SQ_o : std_logic;

constant clk_period :time := 10ns;

begin

uut: jk port map (
                    J_i     => SJ,
                    K_i     => SK,
                    clk_i   => Sclk,
                    rst_i   => Srst,
                    Q_o     => SQ_o
                );


clk_i_process : process
                    begin
                        Sclk <= '0';
                        wait for clk_period/2;
                        Sclk <= '1';
                        wait for clk_period/2;
                end process;
--Test
    process
        begin
            Srst <= '1';
            wait for clk_period;
            Srst <= '0';
            wait for 2*clk_period;

            SJ <= '0';
            SK <= '0';
            wait for 2*clk_period;

            SJ <= '0';
            SK <= '1';
            wait for 2*clk_period;

            SJ <= '1';
            SK <= '0';
            wait for 2*clk_period;

            SJ <= '1';
            SK <= '1';
            wait for 2*clk_period;

    end process;

end Behavioral;
```

```
entity tb_jk is
end tb_jk;

architecture Behavioral of tb_jk is
```

Bukısım, testbench'in yapısını tanımlar. tb_jk adı verilen bu testbench'in bir entity tanımı vardır, ancak giriş veya çıkış portu bulunmaz, çünkü bir testbench yalnızca diğer modüllerin test edilmesi için kullanılır. architecture **Behavioral** bölümü ise bu testbench'in davranışını açıklar.

```
component jk is
    port(
            J_i     : in std_logic;
            K_i     : in std_logic;
            clk_i   : in std_logic;
            rst_i   : in std_logic;
            Q_o     : out std_logic
        );
end component;
```

Bu bölüm, **jk** adlı flip-flop bileşenini (component) tanımlar. Bu bileşen, test edilen JK flip-flop'un ara bağlantılarının tanımlanmasını sağlar. Yani bizim oluşturduğumuz jk adlı bloğu testbench bloğumuz üst modül olacak şekilde buraya bağlarız. Bileşen, aşağıdaki portlardan oluşur:

- **J_i**: J giriş sinyali.
- **K_i**: K giriş sinyali.
- **clk_i**: Saat sinyali.
- **rst_i**: Asenkron reset sinyali.
- **Q_o**: Flip-flop'un çıkış sinyali.

```
signal  SJ   : std_logic := '0';
signal  SK   : std_logic := '0';
signal  Sclk : std_logic := '0';
signal  Srst : std_logic := '0';
signal  SQ_o : std_logic;
```

Burada, bileşendeki portlara karşılık gelen sinyaller tanımlanır:

- **SJ**, **SK**: J ve K giriş sinyalleri.
- **Sclk**: Saat sinyali.
- **Srst**: Reset sinyali.
- **SQ_o**: Çıkış sinyal

Buradaki sinyaller yine yukarıda belirttiğimiz gibi üst modül ile alt modülü birbirine bağlamak için kullanılır. Üst modülün girişleri ve çıkışlarını burada oluşturduğumuz testbench modülüne bağlamak için ara kablolar ile bağlıyor olarak düşünebilirsiniz. Bu ara kablolar da bizim sinyallerimizdir.

```
constant clk_period :time := 10ns;
```

clk_period adında bir sabit tanımlanır. Bu sabit, saat sinyalinin periyodunu belirler ve burada 10ns olarak ayarlanmıştır.

```
uut: jk port map (
                J_i     => SJ,
                K_i     => SK,
                clk_i   => Sclk,
                rst_i   => Srst,
                Q_o     => SQ_o
            );
```

Bu kısım, jk bileşeninin uut (unit under test) adıyla haritalanmasını sağlar. Bu haritalama ile, jk bileşeninin portları testbench'te tanımlanan sinyallere bağlanır.

```
clk_i_process : process
                begin
                    Sclk <= '0';
                    wait for clk_period/2;
                    Sclk <= '1';
                    wait for clk_period/2;
                end process;
```

Bu process bloğu, saat sinyalini **(Sclk)** oluşturur. Saat sinyali, **clk_period/2** süresince '0' ve ardından **clk_period/2** süresince '1' olarak atanarak bir kare dalga sinyali üretir. Bu döngü, test süresince devam eder.

Sonrasında gelenler ise bizim oluşturduğumuz test senaryolarıdır. Burada girişlerimize gelecek değerlerin senaryoları tek tek yazılır ve verilen JK Flip-Flop'un durumlara göre Q çıkışındaki sonuçlar tablodaki gibi doğru bir şekilde elde edilecek mi diye kontrolü yapılır.

```
--Test
    process
        begin
            Srst <= '1';
            wait for clk_period;
            Srst <= '0';
            wait for 2*clk_period;

            SJ <= '0';
            SK <= '0';
            wait for 2*clk_period;

            SJ <= '0';
            SK <= '1';
            wait for 2*clk_period;

            SJ <= '1';
            SK <= '0';
            wait for 2*clk_period;

            SJ <= '1';
            SK <= '1';
            wait for 2*clk_period;

    end process;

end Behavioral;
```

Bu process bloğu, flip-flop'un çeşitli durumlarını test etmek için belirli sinyalleri sırayla uygulayarak test senaryolarını simüle eder:

1. **Reset Durumu**: Srst sinyali '1' yapılarak JK flip-flop resetlenir, ardından Srst sinyali '0' yapılır.
2. **No Change (Durum Koruma) Durumu**: J=0 ve K=0 atanarak flip-flop'un çıkışının değişmeden kalması beklenir.
3. **Reset (Sıfırlama) Durumu**: J=0 ve K=1 atanarak çıkışın sıfırlanması beklenir.
4. **Set (1 Yapma) Durumu**: J=1 ve K=0 atanarak çıkışın 1 yapılması beklenir.
5. **Toggle (Durum Değiştirme) Durumu**: J=1 ve K=1 atanarak çıkışın mevcut değerinin tersine çevrilmesi beklenir.

Her adımda, **2*clk_period** kadar beklenerek saat sinyali ile işlemin doğru şekilde tetiklenmesi sağlanır.

Sonuç olarak aşağıdaki grafiğe bakarak verdiğimiz senaryonun doğruluğunu gözlemleme şansına sahip oluruz. Bu grafikteki sinyaller ile birlikte neyin ne zaman ne durumda olması gerektiğini gözlemleyebiliriz.