Latch’ler(mandallar) dijital devrelerde kullanılan bir bellek elemanlarıdır. Adını yaptığı işe benzerliğinden ötürü almıştır. Mandal, yani tutucu demektir. Genellikle, bir sinyalin belirli bir anda sahip olduğu değeri hatırlamak için kullanılır. Latch'ler, özellikle geçici veriyi saklamak amacıyla kullanılır ve flip-flop'lara göre daha basit yapılara sahiptir. Latchler ile flip-floplar arasındaki fark saat (**clk**) sinyaline bağlı çalışmamalarıdır. Saat sinyalinden bağımsız olarak veriyi tutarlar.

Özetle:

- Latch, giriş sinyali aktifken mevcut veri değerini saklar.
- Giriş sinyali değiştiğinde, latch yeni sinyal değerini alır ve saklar.
- Saat sinyaline bağlı olmadığı için sürekli olarak girişe duyarlıdır ve bu yüzden level-triggered olarak tanımlanır (örneğin, giriş yüksek seviyedeyken saklar, düşük seviyeye geçtiğinde tutmayı bırakır).

Latch'ler, flip-flop'lara kıyasla daha az kararlıdır ve zamanlamaya bağlı hatalara daha yatkındır. Genellikle flip-flop'lardan önceki geçici depolama işlemlerinde kullanılır.

Yine flip-flop’lar gibi farklı türleri bulunmaktadır. En yaygın olarak bilinenleri SR Latch ve D Latch olanlarıdır. Kısaca bahsetmek gerekirse;

SR (Set-Reset) Latch: En temel latch türüdür. "Set" ve "Reset" girişleri vardır. Set girişine 1 verildiğinde latch çıkışı 1 olur, Reset girişine 1 verildiğinde ise çıkış 0 olur. Ancak, her iki giriş aynı anda 1 olduğunda çelişkili bir durum oluşur, bu yüzden bu giriş kombinasyonundan kaçınılır.

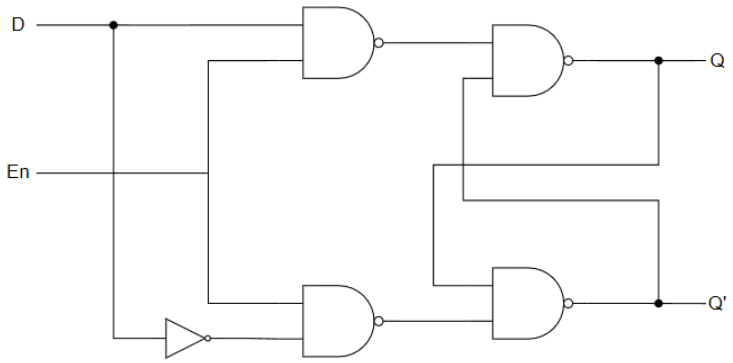
D (Data) Latch: Tek girişe sahiptir ve bu girişe verilen veriyi saklar. Girişin aktif olduğu süre boyunca latch, girişteki değeri çıkışa yansıtır ve saklar. Bu latch, SR latch'in sorunlarını gidermek için tasarlanmıştır ve flip-flop'ların temelini oluşturur.

Burada sizlere D Latch ile alakalı daha detaylı bilgi verip bununla ilgili bir VHDL kodu yazacağım.

D LATCH

D latch, dijital devrelerde kullanılan önemli bir bellek elemanıdır. "D" harfi "Data"yı temsil eder ve bu latch'in temel amacı, girişteki bir veri bitini geçici olarak saklamaktır. D latch'in çalışma prensibi basittir, tek bir girişi vardır. Çıkışta Q ve Q' (ters Q) bulunur. Q, verinin kendisini, Q' ise verinin tersini temsil eder. Aşağıda giriş çıkışları devre üzerinden görebilirsiniz.

Devre Gösterimi



Çalışma prensibi şu şekildedir:

D latch, giriş sinyaline bağlı olarak level-triggered çalışır, yani Enable (E) girişinin durumuna bağlı olarak veriyi alır veya saklar:

Enable (**E**) = 1 (aktif): D latch, **D** girişindeki veriyi doğrudan **Q** çıkışına yansıtır. Yani D'deki değişimler Enable sinyali aktifken anında **Q** çıkışında görülür.

Enable (**E**) = 0 (pasif): D latch, **D** girişindeki veriyi yok sayar ve o anda **Q** çıkışında olan değeri saklar.

Bu sayede D latch, Enable sinyali aktif olduğu sürece girişteki veriyi güncelleyebilir, pasif olduğunda ise çıkışını koruyarak bellek işlevi görür.

Çalışma prensibinin tablo halini de aşağıda görebilirsiniz.

Doğruluk Tablosu

E	D	Q	Tanım
0	X(0,1)	Q	Değişiklik yok
1	0	0	Q = 0; reset durumu
1	1	1	Q = 1; set durumu

Kullanılan bu D Latch'in avantajı olarak da belirsiz durumları önleyeceğini söyleyebiliriz. SR Latch'teki çelişkili durumu ortadan kaldırır. Böylece daha kararlı bir çalışma sağlar. Ayrıca tek bir girişi olduğu için de tek bir veri girişine ihtiyaç duyar. Bu da dijital devrelerde kullanımı basitleştirir.

Şimdi bu D mandalının basitçe VHDL kodunu ve açıklamalarını aşağıda bulabilirsiniz.

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


entity dlatch is
    Port ( D_i      : in STD_LOGIC;
           E_i      : in STD_LOGIC;
           clk_i    : in STD_LOGIC;
           rst_i    : in STD_LOGIC;
           Q_o      : out STD_LOGIC
           );
end dlatch;

architecture Behavioral of dlatch is

begin
    process (clk_i, D_i, E_i)
        variable DE_i   : std_logic_vector (1 downto 0);
        variable latch  : std_logic:='0';
        variable cikis  : std_logic;

        begin
            DE_i:= D_i & E_i;

                if (rst_i = '1') then
                    Q_o <= '0';
                elsif rising_edge (clk_i) then
                    case DE_i is
                       when "00" => cikis := latch;
                       when "01" => cikis := latch;
                       when "10" => cikis := '0';
                       when "11" => cikis := '1';
                       when others => null;
                    end case;
                end if;
        Q_o <= cikis;
    end process;
end Behavioral;
```

Kodu detaylı olarak incelersek;

```vhdl
entity dlatch is
    Port ( D_i      : in STD_LOGIC;
           E_i      : in STD_LOGIC;
           clk_i    : in STD_LOGIC;
           rst_i    : in STD_LOGIC;
           Q_o      : out STD_LOGIC
           );
end dlatch;
```

dlatch isimli bu entity, dış dünyaya açılan giriş ve çıkışları tanımlar. Giriş ve çıkış sinyalleri şunlardır:

D_i: Veri girişi sinyali. Bu, latch'in saklayacağı ya da Q_o çıkışına göndereceği veriyi sağlar.

E_i: Enable ya da etkinleştirme sinyali. E sinyali '1' olduğunda, latch D girişini saklayabilir veya D değerine göre çıkışı değiştirebilir.

clk_i: Saat sinyali. Latch çıkışı bu saat sinyaliyle tetiklenir; rising_edge (pozitif kenar) olduğunda çıkış güncellenir.

rst_i: Reset sinyali. Reset aktifken, devre çıkışı (Q_o) sıfırlanır.

Q_o: Çıkış. Latch'in tutmakta olduğu değeri dış dünyaya iletir.

```
architecture Behavioral of dlatch is

begin
    process (clk_i, D_i, E_i)
        variable DE_i   : std_logic_vector (1 downto 0);
        variable latch  : std_logic:='0';
        variable cikis  : std_logic;
```

Mimari tanımında, process bloğu **clk_i**, **D_i** ve **E_i** sinyallerini kullanır. Bu işlem bloğunda üç değişken tanımlanmıştır:

DE_i: **D_i** ve **E_i** sinyallerini birleştirerek 2 bitlik bir vektör oluşturur.

latch: Latch'in önceki durumunu saklayan değişken, başlangıç değeri '0' olarak tanımlanmıştır.

cikis: Q_o çıkışına atanacak geçici değer.

Bu değişkenlerle, devrenin rst_i ve DE_i durumlarına göre çıkışı nasıl etkileyeceği belirlenir.

```
begin
     DE_i:= D_i & E_i;
```

Bu satır, **D_i** ve **E_i** sinyallerini birleştirerek **DE_i** adlı 2 bitlik bir vektör elde eder.

```
if (rst_i = '1') then
                    Q_o <= '0';
                elsif rising_edge (clk_i) then
                    case DE_i is
                        when "00" => cikis := latch;
                        when "01" => cikis := latch;
                        when "10" => cikis := '0';
                        when "11" => cikis := '1';
                        when others => null;
                    end case;
                end if;
        Q_o <= cikis;
```

rst_i (reset) sinyali '1' olduğunda, **Q_o** çıkışına sıfır atanır. Bu, devrenin başlangıç durumuna geri döndüğünü gösterir.

Reset aktif olmadığında ve **clk_i** sinyali pozitif kenara geldiğinde, **DE_i** durumuna göre cikis değişkeni belirlenir:

- "00": **D_i** ve **E_i** ikisi de '0' olduğunda, çıkış latch'te tutulan önceki değeri (eski değeri) korur.
- "01": **D_i** '0', **E_i** '1' olduğunda da yine çıkış eski değeri korur.
- "10": **D_i** '1', **E_i** '0' olduğunda çıkış sıfırlanır.
- "11": **D_i** ve **E_i** ikisi de '1' olduğunda çıkış bir yapılır.

Bu, **DE_i** durumlarına göre çıkışın nasıl etkileneceğini belirleyen temel çalışma prensibini ortaya koyar.

En son satırda, **cikis** değişkeninin değeri **Q_o** çıkışına atanır. Bu, saat sinyalinin pozitif kenarında güncellenmiş **cikis** değerini **Q_o** çıkışına aktarır.

Bir sonraki sayfada ise yazdığımız kodun testini gerçekleştirmek üzere Testbenc kodunu inceleyelim.

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


entity tb_dlatch is
end tb_dlatch;


architecture Behavioral of tb_dlatch is

component dlatch is
     Port (
           D_i       : in STD_LOGIC;
           E_i       : in STD_LOGIC;
           clk_i     : in STD_LOGIC;
           rst_i     : in STD_LOGIC;
           Q_o       : out STD_LOGIC
           );
end component;

signal SD_i          : std_logic := '0' ;
signal SE_i          : std_logic := '0' ;
signal Sclk_i        : std_logic := '0' ;
signal Srst_i        : std_logic := '0' ;
signal SQ_o          : std_logic := '0' ;

constant clk_period :time := 10ns;

begin

uut: dlatch port map (
                        D_i     => SD_i,
                        E_i     => SE_i,
                        clk_i   => Sclk_i,
                        rst_i   => Srst_i,
                        Q_o     => SQ_o
                    );


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

ED_in:  process
            begin
                Srst_i <= '1';
                wait for clk_period*2;
                Srst_i <= '0';
                wait for clk_period*2;

                SE_i <= '0';
                SD_i <= '0';
                wait for clk_period*2;
                SE_i <= '0';
                SD_i <= '1';
                wait for clk_period*2;
                SE_i <= '1';
                SD_i <= '0';
                wait for clk_period*2;
                SE_i <= '1';
                SD_i <= '1';
                wait for clk_period*2;

        end process ED_in;
end Behavioral;
```

Detaylı incelemesini aşağıda yapalım:

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


entity tb_dlatch is
end tb_dlatch;
```

Bu kısımda, tb_dlatch adlı testbench entity tanımlanır. Testbenchlerde genellikle giriş veya çıkış portları olmaz, çünkü testbench doğrudan sinyal ve işlemler tanımlar ve modülün çalışmasını test etmek için kendi içerisinde sinyalleri yönlendirir.

```
architecture Behavioral of tb_dlatch is

component dlatch is
     Port (
           D_i       : in STD_LOGIC;
           E_i       : in STD_LOGIC;
           clk_i     : in STD_LOGIC;
           rst_i     : in STD_LOGIC;
           Q_o       : out STD_LOGIC
           );
end component;
```

Bu kısımda, test edilecek olan dlatch bileşeni tanımlanır. **dlatch** bileşeninin giriş ve çıkış portları, daha önceki dlatch kodunda tanımlananlarla aynıdır. Aslında ara bağlantıları burada yaparız:

- **D_i:** Latch'e girilecek veri.
- **E_i:** Latch'in etkinleştirme sinyali.
- **clk_i:** Saat sinyali.
- **rst_i:** Reset sinyali.
- **Q_o:** Latch'in çıkışı.

```
signal SD_i          : std_logic := '0' ;
signal SE_i          : std_logic := '0' ;
signal Sclk_i        : std_logic := '0' ;
signal Srst_i        : std_logic := '0' ;
signal SQ_o          : std_logic := '0' ;

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

Bu kısımda dlatch bileşeninin portlarına bağlı olan sinyaller tanımlanır. Bu sinyaller, test sırasında bileşene uygulanacak olan giriş ve çıkışları temsil eder:

- **SD_i**, **SE_i**, **Sclk_i**, **Srst_i**: **D_i**, **E_i**, **clk_i**, ve **rst_i** portlarına bağlanan sinyallerdir.
- **SQ_o**: **Q_o** çıkışına bağlanır.

clk_period ise saat sinyali için kullanılan bir zaman sabitidir ve 10ns olarak tanımlanmıştır.

```
uut: dlatch port map (
                        D_i     => SD_i,
                        E_i     => SE_i,
                        clk_i   => Sclk_i,
                        rst_i   => Srst_i,
                        Q_o     => SQ_o
                    );
```

Bu satırda, **uut** (unit under test) adı verilen dlatch bileşeni testbench'e dahil edilir ve giriş/çıkışları ilgili sinyallere bağlanır. **uut**, **SD_i**, **SE_i**, **Sclk_i**, **Srst_i** ve **SQ_o** sinyallerine bağlanarak test edilecek olan dlatch bileşeninin tb_dlatch üzerindeki portlara erişmesini sağlar.

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

Bu işlem, saat sinyalini üretir ve **clk_period** süresine göre her yarı periyotta bir sinyalin durumunu değiştirir (0 ve 1). Yani, burada 10ns periyot tanımlandığı için her **5ns**'de bir durum değiştiren bir saat sinyali üretilmiş olur. Bu saat sinyali, dlatch bileşeninin **clk_i** girişini sürmek için kullanılır.

```
ED_in:  process
            begin
                Srst_i <= '1';
                wait for clk_period*2;
                Srst_i <= '0';
                wait for clk_period*2;

                SE_i <= '0';
                SD_i <= '0';
                wait for clk_period*2;
                SE_i <= '0';
                SD_i <= '1';
                wait for clk_period*2;
                SE_i <= '1';
                SD_i <= '0';
                wait for clk_period*2;
                SE_i <= '1';
                SD_i <= '1';
                wait for clk_period*2;

        end process ED_in;
end Behavioral;
```

Bu işlem bloğu, dlatch devresinin giriş sinyallerini belirli bir sırayla değiştirerek testler uygular. Her bir durumda, **clk_period*2** kadar beklenir ve ardından yeni bir giriş kombinasyonu uygulanır:

1. **Başlangıçta reset aktif edilir: Srst_i** '1' yapılır ve iki saat periyodu boyunca beklenir. Bu sırada **Q_o** çıkışı sıfırlanır.

2. **Reset devre dışı bırakılır: Srst_i** '0' yapılır ve yine iki saat periyodu beklenir. Bu adımda, latch devresi artık girişlere göre davranmaya başlar.

3. **İlk test kombinasyonu: SE_i** = 0, **SD_i** = 0. Latch devresi etkin değilken D sinyali sıfırdır.
4. **İkinci test kombinasyonu: SE_i** = 0, **SD_i** = 1. Latch devresi etkin değilken D sinyali bir olur.
5. **Üçüncü test kombinasyonu: SE_i** = 1, **SD_i** = 0. Latch devresi etkinken D sinyali sıfır olur.
6. **Dördüncü test kombinasyonu: SE_i** = 1, **SD_i** = 1. Latch devresi etkinken D sinyali bir olur.

Bu giriş kombinasyonları, latch'in **D** ve **E** girişlerine göre **Q_o** çıkışında doğru davranışı sergileyip sergilemediğini test etmek için kullanılır.

Yazdığımız bu test sonrasında aşağıda gözüken simülasyon ekranımız karşımıza çıkmaktadır. Buradaki ekrandan sinyalleri kontrol ederek doğru yaptığımızı da kontrol edebiliriz.