# آزمایشگاه معماری کامپیوتر

# جلسه سوم

- نحوه پیاده‌سازی testbench
- طراحی DFFبا ریست (Reset) ناهمگام (Asynchronous) در منطق منفی (Active Low)
- طراحی TFFبا ریست (Reset) ناهمگام (Asynchronous).
- طراحی Ripple Counter، چهار بیتی با استفاده از TFF.
- طراحی توالی‌یابی (Sequence Detector) که رشته "۱۱۰۱" را تشخیص دهد. (پیاده‌سازی به استفاده از ماشین حالت Mealy و Moore)
- طراحی مدار توالی‌یابی که رخداد هر یک از رشته‌های "۰۱۱۰" و "۰۱۰۱" را تشخیص دهد.

# ساختار کلی Testbench در VHDL

۰۱ ساختار کلی TestBench در VHDL

۰۲ استفاده از ساختارهای زمانی

۰۳ مثال testbench

# ساختار کلی Testbench در VHDL

- Testbench از کدهای VHDL غیرقابل سنتز تشکیل می‌شود که برای مدار طراحی شده، ورودی تولید می‌کند و با پایایش خروجی نشان می‌دهد که مدار طراحی شده به درستی کار می‌کند.
- دیاگرام زیر معماری یک testbench ساده را نشان می‌دهد.

# ساختار کلی Testbench در VHDL

```
Entity TB_NAME is –empty entity
End TB_NAME
```

```
Architecture TB of TB_NAME is
COMPONENT UUT
PORT
{
                .....
};
End COMPONENT;
Signal .....
Begin
U1 : UUT
PORT MAP (.....);
```

```
STIMULI:
Process
begin
.....
wait;
end process;
end architecture;
```

# نحوه پیاده‌سازی Testbench

- بلوک محرک ورودی‌های طراحی FPGA را ایجاد می‌کند و یک بلوک جداگانه خروجی‌ها را بررسی می‌کند. محرک و بررسی کننده خروجی برای طرح‌های بزرگتر در فایل‌های جداگانه قرار خواهد گرفت. اما برای طرح‌های کوچکتر می‌توان بلوک‌های مختلف را در یک فایل قرار داد.

- Instant کردن واحد تحت آزمایش
  - این عمل را می‌توان مانند instant کردن یک مدار کوچک در یک ساختار بزرگتر، به دو شکل استفاده از Component یا entity instantiation مستقیم انجام داد.

# نحوه پیاده‌سازی Testbench

Unique name

```
component and_gate is
        port (
                a           : in std_logic
                b           : in std_logic
                and_out     : out std_logic
                );
end component  and_gate;
```

```
and_gate_instance: component  and_gate
        portmap (
            a         => signal_a,
            b         => signal_b,
            and_out   => signal_and_out
                    );
```

# نحوه پیاده‌سازی Testbench

```
and_gate entity is
        port (
                a,b         : in std_logic
                and_out     : out std_logic
                );
end and_gate;

Architecture rtl of and_gate is
Begin
 and_out <= a and b;
End and_gate;
```

```
and_gate_instance: entity  work.and_gate(rtl)
        portmap (
            a         => signal_a,
            b         => signal_b,
            and_out   => signal_and_out
                    );
```

# استفاده از ساختارهای زمانی

- تفاوت میان کد توصیف سخت‌افزار و تست
- دو ساختاری که در VHDL برای مصرف کردن زمان استفاده می‌شود:
  1. VHDL after statement
  2. Wait statement
- سطوح مختلف/پیچیدگی Testbench
  ۱. تولید Stimuli ورودی ← تصدیق دستی خروجی‌ها
  ۲. تولید Stimuli ورودی ← تصدیق اتوماتیک خروجی‌ها
  ۳. استفاده از کامپوننت‌های تصدیق

# استفاده از ساختارهای زمانی

## نوع‌های زمانی VHDL

IEEE 1076-2008 VHDL Standard

```
time_ex <= 100 fs; -- 100 femroseconds
time_ex <= 1.1 ns; -- 1100 femroseconds
time_ex <= 1.1 sec; -- 1100 miliseconds
```

| واحد | مقدار |
|---|---|
| fs | |
| ps | 1000 fs |
| ns | 1000 ps |
| us | 1000 ns |
| ms | 1000 us |
| sec | 1000 ms |
| min | 60 sec |
| hr | 60 min |

# VHDL after statment

## کلمه کلیدی After

```
<signal> <= <initial_value>, <end_value> after <time>;
reset <= ‘1’, ‘0’ after 1 us;
clock <= not clock after 10 ns;
```

# VHDL wait statment

- از wait به منظور تعلیق اجرای کد، درون بلوک پروسس استفاده می‌شود.
- این ساختار فقط برای پروسس‌هایی که لیست حساسیت ندارند قابل استفاده است.
- این ساختار دارای سه گونه متفاوت است:
  ۱. استفاده از wait برای توقف اجرا، برای یک دوره زمانی مشخص شده.
  ۲. استفاده از wait برای توقف اجرا، تا زمانی که مقدار بولین یک شرط منطقی تعیین شده true شود.
  ۳. استفاده از wait برای توقف اجرا، تا زمانی که یک سیگنال وضعیت (state) را تغییر دهد.

# VHDL wait statment

نوع ۱ :

wait for <time>  EX  wait for 1ms;

نوع ۲ :

wait until <condition> for <time>  EX  wait until (sig_a = ‘1’ and sig_b = ‘1’ ) for 1 us;

نوع ۳ :

wait on <signal_name>  EX  wait on sig_a, sig_b;

# مثال testbench

۱. ساخت یک entity و architecture

۲. Instant کردن واحد تحت آزمایش

۳. تولید clock و reset

۴. نوشتن محرک

# مثال testbench

مثال

```
Entity exp_tb is
End entity exp_tb;

Architecture test of exp_tb is
        signal clock :std_logic :='0';
        signal reset  :std_logic :='1';
        signal and_in : std_logic_vector(1downto0) := (others => '0');
        alias in_a is and_in(0);
        signal in_b     :std_logic;
        signal out_q    :std_logic;
begin
```

# مثال testbench

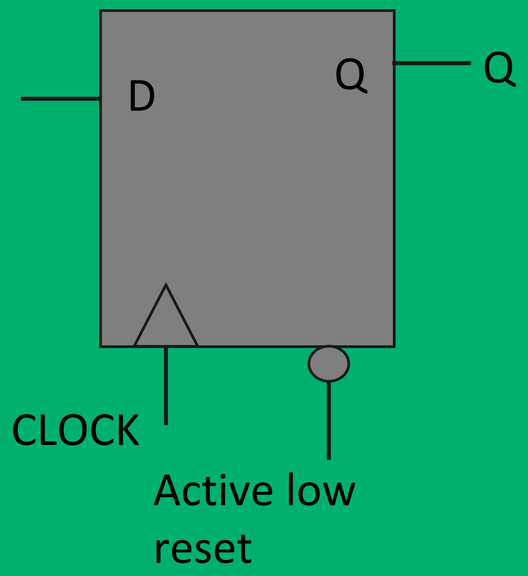
```
--reset and clock
Clock <= not clock after 1ns;
Reset <= ‘1’, ‘0’ after 5 ns;
--instantiate the unit under test (UUT)
UUT: entity work.example_design(rtl)
Port map(
            a=> in_a,
            b=> in_b;
            q=> out_q;
-- generate the test stimulus
Process begin
--wait for the rest to be released before
Wait until ( reset = ‘0’);
--generate each of in turn, waiting 2 clock periods between each iteration ti allow for propagation
time
```

# مثال testbench

```
and_in <= “00”;
wait for 2ns;
and_in <= “01”;
wait for 2ns;
and_in <= “10”;
wait for 2ns;
and_in <= “11”;
--testing complete
Wait;
End process stimulus;
End architecture exp_tb;
```

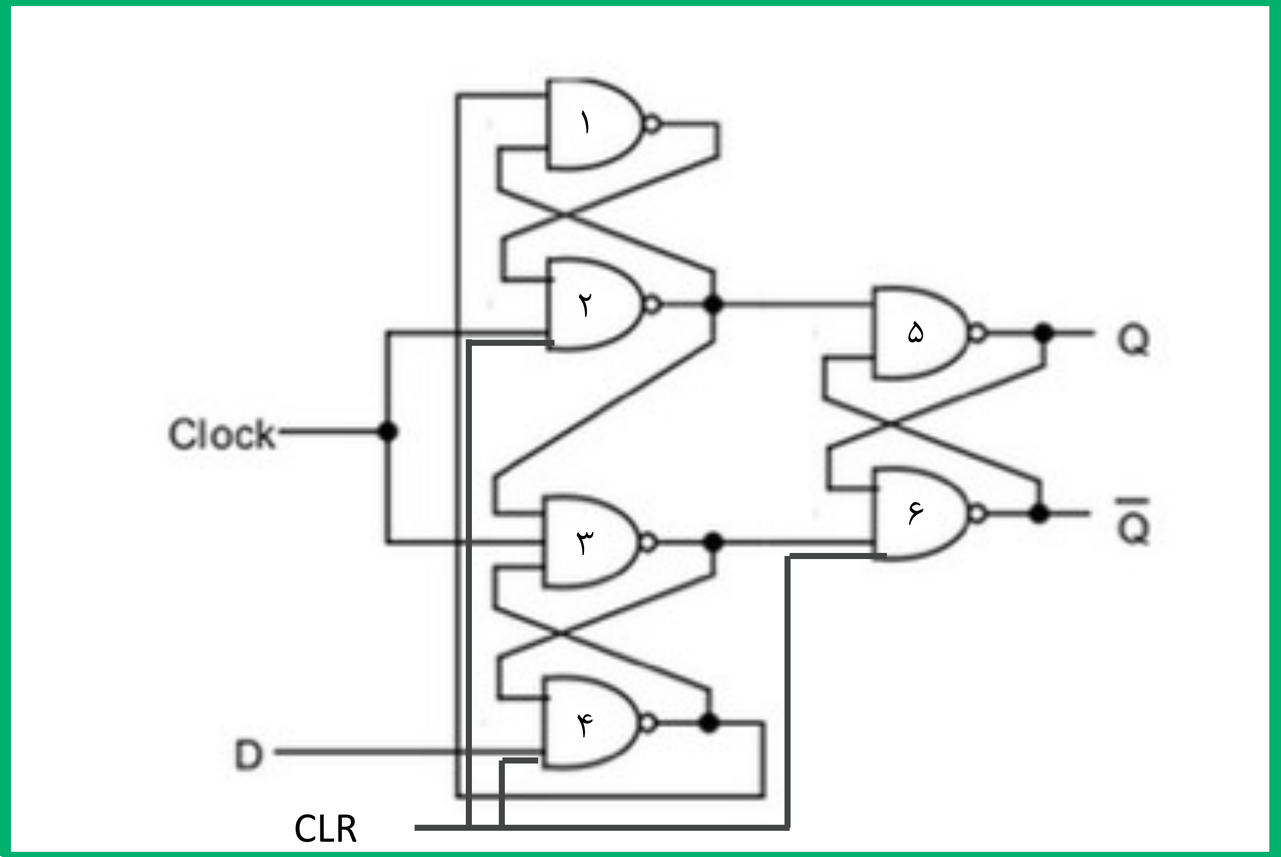
# طراحی DFF

## یک DFF



# طراحی DFF

## پیاده‌سازی شماتیک سطح گیت



# طراحی DFF

## مثال

```
Library IEEE;
USE IEEE.Std_logic_1164.all;
entity RisingEdge_DFlipFlop_AsyncResetLow is
                port( Q : out std_logic;
                        Clk :in std_logic;
                        async_reset: in std_logic;
                        D :in std_logic );
end RisingEdge_DFlipFlop_AsyncResetLow;
```

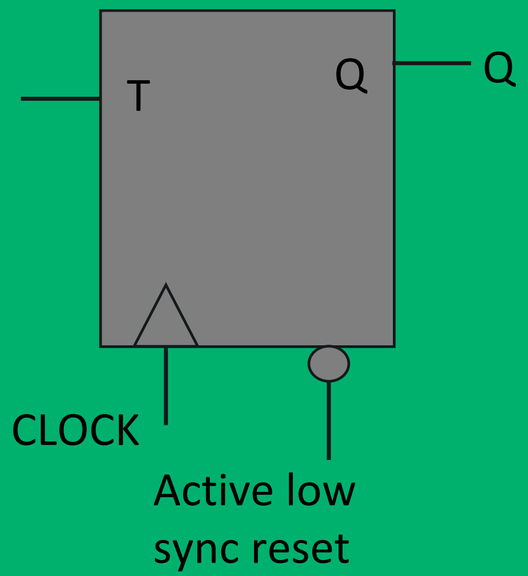
# طراحی DFF

## مثال

```
architecture Behavioral of RisingEdge_DFlipFlop_AsyncResetLow is
Begin
        process(Clk,async_reset)
            begin
              if(async_reset='0') then Q <= '0';
              elsif(rising_edge(Clk)) then Q <= D;
              end if;
        end process;
end Behavioral;
```

# طراحی TFF

## یک TFF



# طراحی TFF

## TFF

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

entity t_trigger is
    port (T,Reset,CLK,CLK_enable: in std_logic;
                Q: out std_logic);
end t_trigger;
architecture beh_t_trigger of t_trigger is

begin
     process (Reset,CLK)
     variable  temp: std_logic;
```

# طراحی TFF

TFF

```
begin
        if (rising_edge(CLK)) then     --sometimes you need to include a package for rising_edge, y
can use CLK'EVENT AND CLK = '1' instead
            if Reset='1' then
                temp := '0';
            elsif CLK_enable ='1' then
                temp := T xor temp;
            end if;
        end if;
    Q <= temp;
    end process;
end beh_t_trigger;
```

# طراحی Ripple Counter

| q3 | q2 | q1 | q0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |