

مثال: اگر یک feedback بدون تاخیر داشته باشیم و مدل را از طریق روش δ -delay آنالیز کنیم، در آن صورت نمودار ورودی و خروجی آتی:
 نکته: در واقعیت gate بدون تاخیر نداریم.

مثال:

این کلمه کلیدی باید باشد.

ENTITY test IS END test;

ARCHITECTURE test01 OF test IS

SIGNAL x,y,z: bit;

BEGIN

$x \leq '0', '1' \text{ AFTER } 10 \text{ ns}, '0' \text{ AFTER } 20 \text{ ns}, '1' \text{ AFTER } 25 \text{ ns}, '0' \text{ AFTER } 27 \text{ ns};$

$y \leq \text{NOT } x \text{ AFTER } 3 \text{ ns};$ → در این دستور مکانیزم تاخیر از نوع inertial است

$z \leq \text{NOT } y \text{ AFTER } 3 \text{ ns};$ مقدار کوچکی از reject می‌گذاشت اما مقدار مساوی

END test01; باید بیش از ۲ns را reject نمی‌کند

مراحل انجام simulate با modelsim:

1) File → Change Directory... (تعیین محل انجام کار، workspace)

2) File → New → Library در ادامه درسی می‌بینیم

Create work library (نام آن work باشد تا به صورت اتوماتیک use شود.)

3) File → New → Source → VHDL

4) Compile Source code

5) From Library window, find work library and corresponding source file

6) Right click → Simulate without optimization (حتماً باید مسازی غیرفعال باشد.)

7) Choose signal you want

8) Add → To Wave → Selected Signals

Add → To List → Selected Signals (در صورتی که نخواهیم δ -delay اعمال کنیم)

.... (دستورات)

9) Simulate → END simulation

نمونه‌ی دستورات مربوط به simulation:

- run 100ns (به مدت زمان 100ns، simulate را انجام می‌دهد.)
- restart (برای بازگرداندن زمان simulation به زمان صفر استفاده می‌شود. دستورات run، مقدار زمان شبیه‌سازی را جلوی پرند)

ساختار زبان VHDL

ساختار یک ENTITY در زبان VHDL

ENTITY name IS

PORT (port_name : IN/OUT/BUFFER/INOUT;
other ports...);

BEGIN

END name;

انواع port ها:

- پورت IN: به این Port نمی‌توان مقاری را assign کرد اما می‌توان مقدار آن را خواند.
 - پورت OUT: به این Port تنها می‌توان مقاری را assign کرد اما مقدار آن را نمی‌توان خواند.
 - پورت INOUT: مقدار این Port را هم می‌توان خواند و هم می‌توان نوشت.
- هنگامی که می‌خواهیم روی آن مقدار بنویسیم، باید از تیرین High Impedance باشد.
- کار با این نوع، معمولاً سخت است.
- پورت BUFFER: مانند پورت OUT است اما می‌توانیم مقدار آن را بخوانیم.

نکته: هر architecture پس از کامپایل شدن، در درون library قرار می‌گیرد و از آن به بعد در بقیه‌ی جاهای می‌توانیم از آن استفاده کنیم. هر لایبری می‌تواند شامل یک سری package نیز باشد که در آن‌ها تعاریف، توابع و... تعریف خواهند شد.

ادامی درس:

تفاوت type های bit و std_logic:

- نوع بیت تنها دارای مقادیر معزونی ('0' و '1') است.

- نوع std_logic دارای 9 مقدار است: std_logic ('0', '1', 'X', 'Z', 'W', 'L', 'H', 'U', '-')

نوع std_logic دارای مقادیر زیر است:

- Uninitialized ('U'): سیگنال تعریف شده است اما به آن مقدار assign نشده است.

- Corrupt ('X'): هر زمان دو مقدار به آن assign شده است.

- High Impedance ('Z'): به این معناست که سیگنال مقاری نگرفته است اما وضعیت مدار به گونه ای شده است که هیچ driver

این سیگنال را تأمین نمی کند.

'1' — Z (driver of z is '1')

'1' X_Z (z has no driver)

- Weak Unknown ('W') (سیگنال ضعیف)

- Weak Low ('L') (سیگنال معزونی ضعیف)

- Weak High ('H') (سیگنال '1' ضعیف)

این سیگنال امروزه دیگر کاربرد زیادی نداشته و استفاده نمی شوند.

- don't care ('-'): استفاده از این مقدار برای بهینه سازی مدار مفید است. (کارنومپو...)

کتابخانه های استاندارد موجود در VHDL:

library std

- package standard: bit, bit_vector, integer, natural, positive, boolean, boolean_vector

integer_vector, character, real, ...

- package textio: line, text

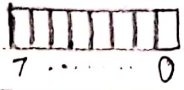
...

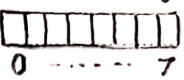
library ieee

- package std_logic_1164: std_logic, std_ulogic, std_logic_vector

...

استاندارد مورد استفاده
ieee

SIGNAL a, b : bit_vector (7 DOWNTO 0) →  (little-endian) مثال :

SIGNAL z : bit_vector (0 TO 7) →  (big-endian)

Usage: z(index)

نکته: در حالتی که پردازنده ما به صورت little endian باشد و به صورت byte addressable است یعنی به هر بایت به صورت little endian ذخیره می شود. بابت ترتیب byte ها به این گونه خواهد بود little endian رعایت شود.

نکته: integer یک عدد ۳۲ بیتی علامت دار است.

integer: -2147483687 ~ +2147483687 بازه ی آن در VHDL به صورت مقدارین انتخاب شده است.

INTEGER'LOW INTEGER'HIGH

$-(2^{31}-1) \sim +(2^{31}-1)$

natural: 0 ~ INTEGER'HIGH

نکته:

positive: 1 ~ INTEGER'HIGH

نکته: type های زیر دارای بازه های زیر هستند:

BYTE: 7 DOWNTO 0

نکته: نمونه ی عدد صحیح ۲۴ بیتی در زبان ++C، int64 است.

HALFWORD: 15 DOWNTO 0

WORD: 31 DOWNTO 0

DOUBLE WORD: 63 DOWNTO 0

نکته: یک signal، می توان از نوع boolean تعریف کرد که در آن صورت مقدار TRUE و FALSE می گیرد.

ENTITY mux4to1 IS

مقدار بیش فرض آن برابر با FALSE است.

PORT(a, b, c, d : IN bit;

s : IN bit_vector(1 DOWNTO 0);

z : OUT bit);

مثال یک multiplexer:

END mux4to1;

ARCHITECTURE test OF mux4to1 IS BEGIN

نکته: مشابه زبان ++C، یک بیت در بین ۱ قرار

z <= a WHEN s = "00" ELSE

می گیرد و چند بیت در بین " " قرار می گیرد.

b WHEN s = "01" ELSE

c WHEN s = "10" ELSE

d;

END test;

PAPCO

برای استفاده از package یک لایبری از دستورات و پروتکل‌های فایل استفاده می‌شود:

```
LIBRARY library;
```

```
USE library.package.ALL;
```

تمام موارد موجود در standard package (جزر std) و لایبری work به صورت پیش فرض تعریف شده و بدون استفاده از تعریف با لا قابل استفاده هستند.

مثال یک جمع کننده:

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.ALL;
```

```
ENTITY adder4bit IS
```

```
    PORT (a, b : IN std_logic_vector(31 DOWNTO 0);
```

```
          ci : IN std_logic;
```

```
          sum : OUT std_logic_vector(31 DOWNTO 0);
```

```
          co : OUT std_logic);
```

```
END adder4bit;
```

```
ARCHITECTURE test OF adder4bit IS
```

```
    SIGNAL c : std_logic_vector(32 DOWNTO 0); → سیگنال‌های داخلی یک entity در این قسمت تعریف می‌شود.
```

```
BEGIN
```

```
    c(0) <= ci;
```

```
    L1: FOR i IN 0 TO 31 GENERATE
```

```
        sum(i) <= a(i) XOR b(i) XOR c(i)
```

```
        c(i+1) <= (a(i) AND b(i)) OR (a(i) AND c(i)) OR (b(i) AND c(i));
```

```
    END GENERATE L1;
```

```
    co <= c(32);
```

```
END test;
```

دستورات مورد نیاز برای simulation :

توسط دستور force می توان به یک سیگنال، یک مقداردهی اولیه بکنیم مثلاً:

force signal_name "value" 0ns, "value2" 30ns, ...

نکته: ماد و type به نام های std-logic و std-ulogic داریم که کاملاً مشابه هم بوده و مقادیر یکسانی به آن ها می توان assign نمود.

تفاوت این دو در این است که std-ulogic دارای resolution نیست و به همین resolve نمی شود.

← unresolved

اگر در یک زمان از تحلیل مدار، دو مقدار به یک signal نسبت داده شود، توسط جدول resolution که یک جدول دوبیتی است مشخص می شود که مقدارهای سیگنال چیست.

این جدول در قسمت تعریف یک type است و ما می توانیم بر اساس نیاز خود (مثال wired-and یا wired-or) آن را تغییر دهیم.

std-ulogic دارای این جدول نیست و به همین دلیل resolve نمی شود. به همین دلیل اگر بیش از یک مقدار به آن assign شود، در هنگام کامپایل، خطا گرفته می شود.

مزیت های std-logic نسبت به bit :

- دارای ۹ مقدار است که دقت بیش تری به ما می دهد.

- قابلیت resolve شدن دارد.

- مقدار زیادی کتابخانه commercial برای آن وجود دارد.