Типы данных языка VHDL

Типы данных языка VHDL определяются:

- набором значений, которые могут принимать объекты данного типа;
- набором операций, которые можно выполнять над объектами данного типа.

Операции на объектами заданного типа подразделяются на:

- набор явно определенных подпрограмм, параметры или возвращаемый результат которых являются объектами данного типа;
- набор базовых и предопределенных операций над объектами данного типа.

Классы типов данных языка VHDL

Типы данных языка VHDL подразделяются на следующие классы:

- скалярные типы;
- агрегатные типы данных;
- указательные типы данных;
- файловые типы данных;
- защищенные типы данных.

Оператор объявления типа данных

```
type имя_типа is определение_типа ;
```

Определение типа может быть :

- определением скалярного типа;
- определением агрегатного типа;
- определением указательного типа;
- определением файлового типа;
- определением защищенного типа.

Пример определения типа :

```
type BIT is ('0','1') ;
```

Скалярные типы данных

Скалярные типы данных подразделяются на :

- численные;
- перечислимые;
- физические.

В пакете **STANDARD** предопределены следующие скалярные типы данных :

- численные : **`integer`**, **`real`** ;
- перечислимые : **`character`**, **`bit`**, **`boolean`**, **`severity_level`**;
- физические : **`time`**.

Скалярные типы данных (численные типы данных)

Синтаксис определения численных типов данных

```
type имя_численного_типа is диапазон ;

range ограничение to|downto ограничение
```

Пример определения численных типов в пакете **STANDARD** :

```
type INTEGER is range -2147483647 to 2147483647;

type REAL is range -1.7014111e+308 to 1.7014111e+308;
```

Пример определения других численных типов :

```
type unsigned_short is range 0 to 255;

type data is range 0 to 15;
```

Предопределенные операции для численных типов данных

Для численных типов данных **integer** и **real** предопределены следующие операции:

- операции сравнения : **= , /= , < , <= , > , >=** ;

  тип возвращаемого результата для операций сравнения – **boolean**.

- арифметические операции : **+ , – , abs , * , / , ****** ;

- для типа integer дополнительно определены операции **mod , rem** ;

  тип возвращаемого результата для арифметических операций – **integer** или **real**.

Скалярные типы данных (перечислимые типы данных)

Синтаксис определения перечислимых типов данных

```
type имя_перечислимого_типа is (значение1[,значение2]);
```

```
идентификатор | символьная_константа
```

Пример определения перечислимых типов в пакете **STANDARD** :

```
type BOOLEAN is (TRUE,FALSE);

type BIT  is ('0','1');

type SEVERITY_LEVEL is (NOTE,WARNING,ERROR,FAILURE);
```

Предопределенные операции для численных типов данных

Для перечислимых типов данных **boolean** и **bit** предопределены следующие операции:

- операции сравнения : = , /= , < , <= , > , >= ;

  тип возвращаемого результата для операций сравнения – **boolean**.

- логические операции : **and, or, nand, nor, xor, xnor, not** ;

  тип возвращаемого результата для логических операций – **boolean** или **real**.

Операции сравнения также определены и для перечислимых типов **character** и **severity_level**

Скалярные типы данных (физические типы данных)

Синтаксис определения физических типов данных :

```
type имя_физического_типа is диапазон;
   units
      базовая_единица_измерения ;
      [производная_единица_измерения ;]
   end units [имя_физического_типа];
```

Скалярные типы данных (физические типы данных)

Пример определения физического типа в пакете STANDARD :

```
type TIME is range -2147483647 to 2147483647
   units
      fs;                          -- femtosecond
      ps    = 1000 fs;             -- picosecond
      ns    = 1000 ps;             -- nanosecond
      us    = 1000 ns;             -- microsecond
      ms    = 1000 us;             -- millisecond
      sec   = 1000 ms;             -- second
      min   =   60 sec;            -- minute
      hr    =   60 min;            -- hour
   end units;
```

Предопределенные операции для физических типов данных

Для физического типа данных **time** предопределены следующие операции:

- операции сравнения : **= , /= , < , <= , > , >=** ;

  тип возвращаемого результата для операций сравнения – **boolean**.

- арифметические операции : **+ , – , abs , * ,** / ;

  тип возвращаемого результата для арифметических операций – **time**.

- операция получения текущего времени моделирования **now**.

Агрегатные типы данных

Агрегатные типы данных подразделяются на массивы и записи.

Массивы – это набор данных одного типа, объединенных общим именем и различаемых по порядковым номерам (индексам). Массивы могут быть ограниченными и неограниченными.

Запись – это структура данных, каждая информационная единица которой, называемая полем записи, имеет индивидуальное имя и может быть индивидуального типа.

Агрегатные типы данных (массивы)

Синтаксис определения массивов :

```
type имя_массива is array (диапазон1[,диапазон2]);
```

Диапазон может быть ограниченным :

```
[имя_типа range]
   ограничение1 to|downto ограничение2
```

или неограниченным :

```
имя_типа range<>
```

Агрегатные типы данных (примеры объявления массивов)

Пример объявления агрегатного типа для описания памяти из 256 целочисленных элементов :

```
type ram is array (255 downto 0) of integer;
```

Предопределенные в пакете **STANDARD** неограниченные агрегатные типы данных:

```
type STRING is array(POSITIVE range<>)of CHARACTER;

type BIT_VECTOR is array (NATURAL range<>) of BIT;
```

Агрегатные типы данных (примеры обращения к массивам)

При обращении к элементам массива индексы помещаются в скобках за именем массива :

```
ram (x) <= '0';
```

Тип индексного выражения должен соответствовать типу индекса, объявленного при декларации массива

При обращении к элементам многомерного массива индексные выражения записываются через запятые в порядке, определенном в декларации типа

```
ram (x,y) <= '0';
```

Агрегатные типы данных (примеры обращения к массивам)

Возможно обращение сразу к нескольким элементам массива с использованием оператора диапазона :

```
ram (10 downto 8) <= "010";
```

Для одномерных массивов определена групповая операция конкатенации (объединения) – &.

```
signal a,b   : bit_vector(1 downto 0);
signal c     : bit_vector(3 downto 0);
a <= "00";
b <= "11";
c <= a & b;
```

Здесь сигнал C примет значение "0011".

Предопределенные операции для агрегатных типов данных

Для агрегатных типов данных **string** и **bit_vector** предопределены следующие операции:

- операции сравнения : **= , /= , < , <= , > , >=** ;

  тип возвращаемого результата для операций сравнения – **boolean**.

- операция конкатенации : **&** ;

  тип возвращаемого результата для операции конкатенации – **string** или **bit_vector**.

Для типа данных **bit_vector** дополнительно определены :

- логические операции : **and, or, nand, nor, xor, xnor, not** ;
- операция сдвига : **sll, srl, sla, sra, rol, ror**.

  Тип возвращаемого результат : **bit_vector**.

Агрегатные типы данных (записи)

Синтаксис определения записей :

```
type имя_записи is
   record
      список_элементов : тип;
      [список_элементов : тип;]
   end record;
```

Агрегатные типы данных (записи)

Пример объявления типа запись :

```
type date is
   record
      day     : integer range 1 to 31;
      month   : integer range 1 to 12;
      year    : integer range 0 to 4000;
   end record;
```

Файловые типы данных

Файловые типы данных используются для представления файлов на локальной системе. Значения объектов файлового типа – это последовательность данных, содержащихся в определенном файле локальной системы.

Синтаксис определения файлового типа данных :

```
type имя_файлового_типа is file of имя_типа;
```

Имя типа задает подтип значений, содержащихся в файле. В качестве базового типа могут использоваться скалярные типы, записи и ограниченные массивы.

Пример определения файлового типа

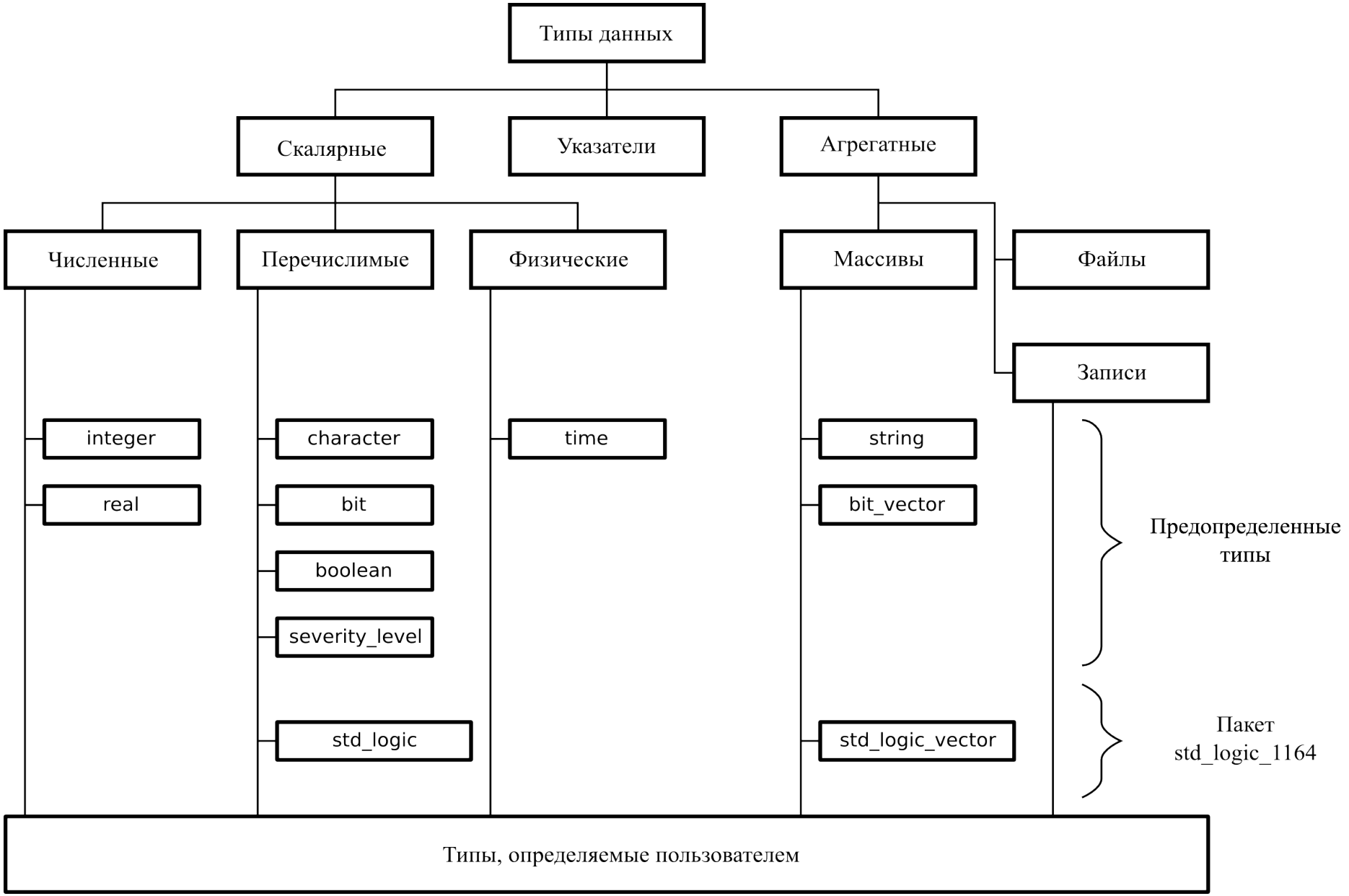
```
type string_file is file of string;
```

Операции с файловыми типами данных

Для файловых типов данных язык VHDL неявно определяет следующие операции:

- **`FileOpen`** – открытие файла ;
- **`FileClose`** – закрытие файла ;
- **`ReadFile`** – чтение из файла ;
- **`WriteFile`** – запись в файл ;
- **`EndFile`** – возвращает **TRUE** при невозможности дальнейшего чтения файла .

Типы данных языка VHDL



Подтипы языка VHDL

Подтип – специфическое понятие языка VHDL. Подтип определяет новый тип на основе уже существующего (базового) типа с определенными ограничениями. Ограничения задают подмножество значений базового типа.

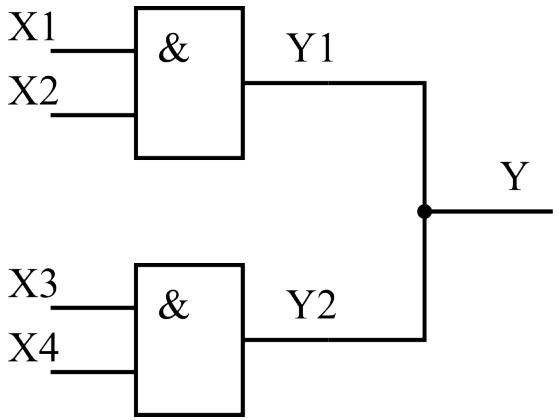
Ограничения подтипа позволяют выявить ошибки на этапе моделирования. Объекты разных подтипов, у которых один базовый тип, могут участвовать в вычислениях без конфликтов типов.

Синтаксис:

```
subtype имя_подтипа is [имя_функции_разрешения]
   имя_базового_типа [ограничение];
```

Функция разрешения подтипа

Функция разрешения для объектов заданного типа вызывается при возникновении конфликтов, когда несколько источников формируют один и тот же сигнал.




```
entity logic is
    port (
        X1,X2,X3,X4     : in std_logic;
        Y               : out std_logic
    );
end logic;

architecture logic of logic is
begin

    Y <= x1 and x2;     -- Y1
    Y <= x3 and x4;     -- Y2

end logic;
```

Функция разрешения подтипа

Для типа **std_logic** функция разрешения использует следующую таблицу :

U	X	0	1	Z	W	L	H	–	
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	U
'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	X
'U'	'X'	'0'	'X'	'0'	'0'	'0'	'0'	'X'	0
'U'	'X'	'X'	'1'	'1'	'1'	'1'	'1'	'X'	1
'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'X'	Z
'U'	'X'	'0'	'1'	'W'	'W'	'W'	'W'	'X'	W
'U'	'X'	'0'	'1'	'L'	'W'	'L'	'W'	'X'	L
'U'	'X'	'0'	'1'	'H'	'W'	'W'	'H'	'X'	H
'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	–

Типы данных пакета STD_LOGIC_1164

В этом пакете определены следующие типы :

- **std_ulogic** ;
- **std_ulogic_vector.**

Данные типы определены следующим образом :

```
type std_ulogic is ('U','X','0','1','Z','W','L','H');

type std_ulogic_vector is

      array (NATURAL range <>) of std_ulogic;
```

Кроме этого определены подтипы с заданной функцией разрешения (других отличий нет) :

- **std_logic** ;
- **std_logic_vector.**

Типы данных пакета STD_LOGIC_1164

Значения типов **`std_ulogic`** и **`std_logic`** :

`'U'` – неустановленное значение ;

`'X'` – неизвестное значение ;

`'0'` – логический ноль ;

`'1'` – логическая единица ;

`'Z'` – состояние высоко сопротивления ;

`'W'` – слабое неизвестное значение

`'L'` – слабый логический ноль ;

`'H'` – слабая логическая единица .

Объекты данных языка VHDL

Любой проект на языке VHDL является описанием преобразования различных наборов данных в дискретных системах. При этом объекты языка VHDL представляют эти данные в проекте и подразделяются на следующие четыре категории :

- **`constant`** – константа ;
- **`signal`** – сигнал ;
- **`variable`** – переменная ;
- **`file`** – файл .

Объекты данных языка VHDL (константа)

Константа определяет данные, не изменяемые в процессе функционирования устройства.

Синтаксис :

```
constant имя_константы : тип_константы [:= значение] ;
```

Поле **"значение"** задает значение константы. В случае отсутствия этого поля такое определение константы называется *отложенным*. Отложенные определения констант используются в декларативной части пакета. При этом в теле пакета должно присутствовать полное определение константы.

Пример определения константы :

```
constant pi : real := 3.141592 ;
```

Объекты данных языка VHDL (сигнал)

Сигнал определяет объект выполняющий функцию передачи информации между модулями проекта (блоками, процессами, проектными модулями и т.д.) или представляющий входные и выходные данные проекта. Сигналу присваиваются свойства изменения во времени.

Синтаксис :

```
signal имя_сигнала : тип_сигнала [:= значение] ;
```

Поле **"значение"** определяет значение сигнала «по-умолчанию».

Пример определения константы :

```
signal clk : std_logic := '0' ;
```

Объекты данных языка VHDL (переменная)

Переменная – это вспомогательная информационная единица, используемая для описания внутренних операций в программных блоках.

Синтаксис :

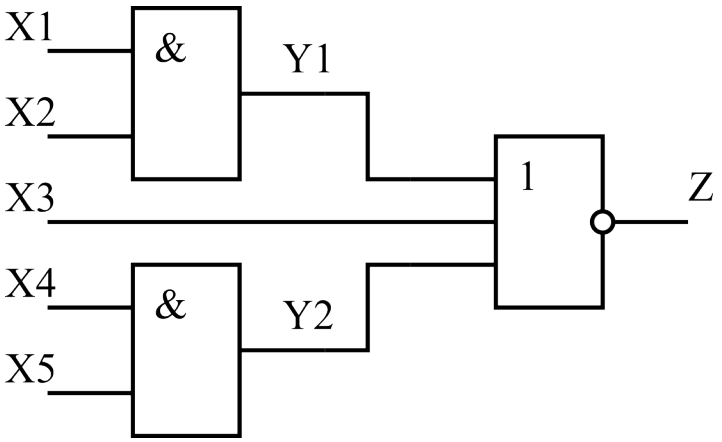
```
variable имя_переменной : тип_переменной [:= значение] ;
```

Поле "значение" определяет начальное значение переменной.

Пример определения переменной :

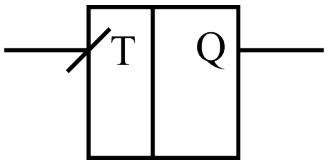
```
variable flip_flop : std_logic := '1' ;
```

Пример использования сигналов в проекте на VHDL




```
entity F is
    port
    (
        X1,X2,X3,X4,X5 : in    std_logic;
        Z              : out   std_logic
    );
end F;

architecture behavioural of F is

    signal Y1 : std_logic;
    signal Y2 : std_logic;

begin

    Y1 <= X1 and X2;
    Y2 <= X4 and X5;
    Z <= not(Y1 or Y2 or X3);

end behavioural;
```

Пример использования переменных в проекте на VHDL




```
entity FF is
    port
    (
        T : in     std_logic;
        Q : out    std_logic
    );
end FF;

architecture behavioural of FF is

begin

    process(T)
        variable q_int : std_logic := '0';
    begin
        if T'event and T = '1' then
            q_int := not q_int;
        end if;
        Q <= q_int;
    end process;

end behavioural;
```

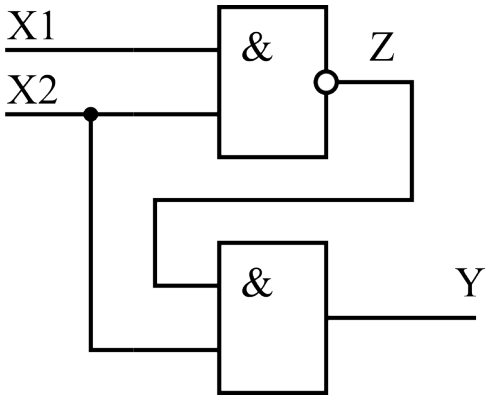
Различия между переменными и сигналами

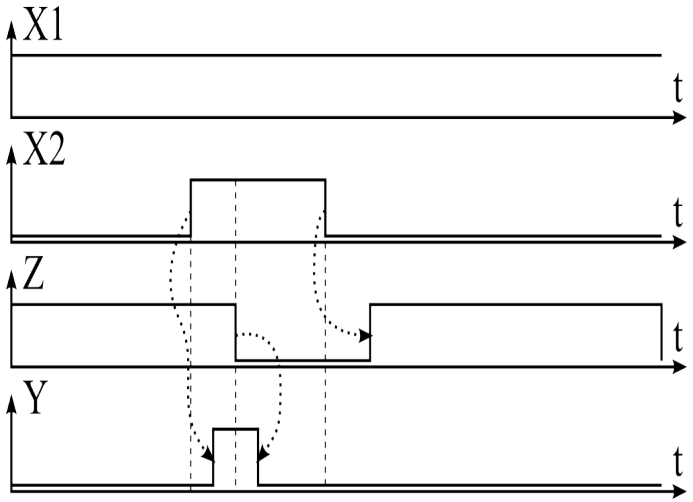
Переменные меняют значения сразу после присвоения, и новое значение непосредственно используются в последующих операторах.

Значение сигнала меняется не сразу после выполнения операции присвоения. Каждому оператору присвоения сопоставляется некий буфер, называемый драйвером. Оператор присваивания передает новое значение драйверу сигнала, и только после того, как завершат выполнение все параллельные операторы, инициированные общим событие, значение драйвера передастся сигналу.

Передача значения может быть задержана еще дольше, если оператор присваивания содержит выражение задержки – **`after`**.

Пример присваивания значения сигналу






```
entity logic is
    port
    (
        X1,X2   : in        std_logic;
        Y       : out       std_logic
    );
end logic;

architecture behavioural of logic is
    signal Z : std_logic;
begin

    process(X1,X2)
    begin
        Z <= not (X1 and X2);
    end process;

    process(X2,Z)
    begin
        Y <= X2 and Z;
    end process;

end behavioural;
```