VHDL cz.1

Rafał Walkowiak
IIn PP
12.11.2019

Struktura projektu

- Definiowanie jednostek projektowych na wielu poziomach.
- Określenie jednostki projektowej najwyższego poziomu – aktualnie realizowanej – testowanej.
- Pozostałe jednostki projektowe możliwe do wykorzystania jako komponenty jednostki najwyższego poziomu.
- Wykorzystanie jednostek projektowych z dołączonych bibliotek jako komponentów innych jednostek projektowych.

Jednostka projektowa – desing entity

Moduł deklaracji jednostki – wej-wy, parametry

```
-- deklaracja standardowej
    biblioteki i pakietu
library ieee;
-- import elementów pakietu biblioteki
use ieee.std_logic_1164.all;
entity sumator is
port(
a:          in std_logic;
b:          in std_logic;
cin:        in std_logic;
s:          out std_logic;
cout:       out std_logic);
-- sygnały
end sumator ;
```

Moduł opisu działania jednostki

```
architecture pierwsza of sumator is
signal s1:std_logic; --deklaracja sygnału

begin
-- ZBIÓR WSPÓŁBIERZNIE
-- REALIZOWANYCH PROCESÓW
-  KAŻDA LINIA ODDZIELNY PROCES
s1<= (a and not b) or (b and not a);
s <= (cin and not s1) or (not cin and s1);
cout<= (a and b) or (s1 and cin);
end pierwsza;
```

Opis działania jednostki – architecture

architecture nazwa_arch of nazwa_jednostki is

Definicje i deklaracje:
(typów, podtypów, stałych, sygnałów, komponentów, konfiguracji)

begin

- instrukcje przypisania wartości do sygnałów
- procesy
- komponenty

end nazwa_arch;

Obiekty danych

- Obiekty danych służą do przechowywania wartości.
- Wyróżnia się trzy klasy obiektów:
  - sygnał – *signal*
  - zmienne – *variable (możliwe tylko w procesach (obiekty lokalne) lub podprogramach)*
  - stałe – *constant*

Obiekty danych

Sygnał :

- Odpowiednikiem sprzętowym sygnału jest ścieżka w układzie scalonym. Wartości sygnałów są przypisywane zgodnie z obowiązującymi w zakresie regułami – np. z pewnym opóźnieniem.
- deklarowane w interfejsie jednostki projektowej oraz jako sygnały wewnętrzne jednostki;
- do przenoszenia informacji między zakresami kodu typu **proces**
- posiadają typ i wartość, która zmienia się w możliwych do określenia i testowania momentach czasu;

```
signal reset:        std_logic;  --sygnał jednobitowy
signal dane:         std_logic_vector(0 to 7); -- wektor sygnałowy
```

Obiekty danych

Zmienne:

Służą do specyfikacji sposobu przetwarzania informacji lokalnie w zakresie **procesu**. Wartości zmiennych są przypisywane natychmiast. Zmienna nie ma odpowiednika sprzętowego i służy do obliczeń w ramach kodu VHDL.

```
variable zmienna:   bit;
variable temp_var: BIT_VECTOR(0 TO 4) :=”10101”;
```

Typy danych

Dotyczą sygnałów, zmiennych i stałych

- BIT, BIT_VECTOR
- **STD_LOGIC, STD_LOGIC_VECTOR,**
- STD_ULOGIC, STD_ULOGIC_VECTOR,
- INTEGER, BOOLEAN,
- TYP WYLICZENIOWY

Typ STD_LOGIC

- STD_LOGIC, STD_LOGIC_VECTOR –
  - *standardowy typ sygnałów*
  - *posiadający 9 możliwych wartości używanych do symulacji różnych konfiguracji połączeń,*
  - *typ rozstrzygalny (możliwych wiele nośników – przypisań równoległych),*
  - *funkcja rozstrzygająca*
- Zastosowanie wymaga deklaracji:
  - LIBRARY IEEE;
  - use IEEE.Std_Logic_1164.all;
  - Pakiet zawiera rozszerzenia standardowego VHDL:
    - std_logic,
    - funkcja rozstrzygająca,
    - funkcje logiczne dla std_logic, std_logic_vector
    - funkcje dla detekcji zboczy sygałów: rising_edge, falling_edge
    - funkcje konwersji typu (następna strona)

STD_LOGIC - STD_LOGIC_VECTOR

- Wartości

  ‘U’ – wartość niezainicjowana,

  ’0’,’1’, ‘X’, - wartości silne: 0, 1, stan nieznany np. efekt jednoczesnego przypisania wartości 0 i 1

  ‘Z’ – stan wysokiej impedancji

  ‘W’,’L’,’H’ – wartości słabe: stan nieznany, 0, 1 – służą do implementacji poziomów sygnałów generowanych przy użyciu rezystorów podciągających (połączenia typu suma na drucie) – jednoczesne podstawienia do sygnału wartości L i 1 rozstrzygane jako1.

  ‘-’ – stan dowolny

  (wielkie litery obowiązkowo)

- STD_LOGIC_VECTOR jest tablicą obiektów STD_LOGIC
- Dla STD_LOGIC_VECTOR operatory arytmetyczne: +,- są dostępne (przeciążenie) po zastosowaniu dodatkowo:

  use IEEE.std_logic_unsigned.all;

Typ *std_ulogic* – charakterystyka jak wyżej, lecz typ nierozstrzygalny

Sygnały

- Sygnały mogą być zdefiniowane w bloku deklaracji jednostki lub w części deklaracyjnej architektury (pomiędzy architecture a begin).
- Przypisanie wartości do sygnału nie jest natychmiastowe lecz jest „harmonogramowane”.
- Z sygnałem skojarzone są: typ i wartość
- Składnia deklaracji:

signal nazwa_sygnału:

typ_sygnału[:=wartość_początkowa];

Przykłady:

```
signal s1, s2 : bit;
signal liczba : integer := 7;
```

Atrybuty sygnałów

Dostarczają dodatkowych informacji o obiektach

obiekt'atrybut[(parametr)];

- Predefiniowane atrybuty:
  - 'left; 'right;'high;'low;
  - Atrybut określa numer lewego, prawego, najmniejszego, największego elementu danego typu, 'left = 'high jeśli definiowano sygnał poprzez downto
  - 'length; 'range; - atrybut określa liczbę bitów, zakres sygnału
  - 'event; 'stable; - atrybut przyjmuje wartość true/false – informacja, że sygnał zmienił lub nie wartość

Atrybuty sygnałów i typów -przykłady

type count is integer range 127 downto 0;

type states is (idle, read, write);

count'left oznacza 127

count'high oznacza 127

states'right oznacza write

states'high oznacza write

If Clock'event and Clock = '1' then Q <= D;

Analogiczne do *'event* i *'stable* informacje dostarczają funkcje określone dla sygnałów - rising_edge(nazwa_sygnalu), falling_edge(nazwa_sygnalu)

Instrukcje

Sekwencyjne

- Działanie układu określone przez kolejność zapisu instrukcji sekwencyjnych.
- Instrukcje sekwencyjne są stosowane w specyfikacji behawioralnej (ang. behavioral description). Przede wszystkim w tzw. procesach

Współbieżne

- Zachowanie układu jest niezależne od kolejności instrukcji przyporządkowania sygnałów.
- Stosowane w specyfikacji typu „przepływ danych” (ang. dataflow description).

Operatory

- Służą do określenia sposobu wyznaczenia nowych wartości.
- Specyfikacja w tabeli poniżej wg malejącego priorytetu
- Konieczna zgodność typu argumentów

Klasa	operator	Typ danych
negacja	NOT	INTEGER, BIT, STD_LOGIC
MNOŻENIA	*,/, mod, rem	INTEGER
ZNAKU	+-	INTEGER
SUMOWANIA	+-	INTEGER, BIT, STD_LOGIC
PRZESUWANIA	*sll, srl, sla, sra, rol, ror* *Przesunięcie wpis 0,* *Przesunięcie wpis wartości krańcowej, przesuniecie cykliczne*	STD_LOGIC,BIT (dla tablic)
RELACYJNE	=, /=, <,<=,>,>=	Argumenty – tego samego typu; wynik – BOOLEAN
LOGICZNE konieczność określenia kolejności realizacji !!!	*and, or, nand, nor, xor, xnor*	BIT, BOOLEAN, STD_LOGIC

Instrukcja przypisania

- Instrukcja przypisania (realizowana sekwencyjnie w procesie, równolegle poza procesem):

  Nazwa_sygnalu **<= wyrazenie;**

- Przykład:

```
architecture strukturalna of error_test is
signal q_reg, q_next : std_logic_vector (7 downto 0);
begin
error <= we(2) and (we(1) or we(0));
 b <= (0=>1,others => '0');  -- przypisuje 0 do wszystkich
   oprócz zerowego bitów wektora sygnału
q_reg <= q_next; -- przypisanie wektorów
end a;
```

Instrukcja przypisania

X <= ‘1’; --skalar

Y <= ”0000” -- wektor 4 bitowy

Wektor1 <= wektor2 & bit ; -- konkatenacja

Z <= a nand c; -- nadanie wartości bez opóźnienia

Z <= a nand c after 10 ns ;

-- nadanie wartości po czasie 10 s opóźnienia (opóźnienie inercyjne)

Do obiektu **sygnal** można przyporządkować wartość obiektu **zmienna** (i zmiennej wartość sygnału również) pod warunkiem przynależności do tego samego typu.

Realizacja instrukcji współbieżnych

- Instrukcja współbieżnego przypisania wartości do sygnału ma ukrytą listę czułości, która zawiera wszystkie **sygnały** znajdujące się po prawej stronie symbolu <=.
- Instrukcja współbieżnego przypisania jest wykonywana zawsze, gdy nastąpi zmiana dowolnego sygnału występującego na liście czułości.
- Dla typów rozstrzygalnych współbieżne przypisania wartości do sygnałów powodują utworzenie kolejnych nośników (ang. driver) dla uaktualnianych sygnałów. W przypadku uaktualniania sygnału przez wiele instrukcji dla utworzonych nośników sygnału stosowana jest funkcja rozstrzygająca określająca stan sygnału. Wykorzystanie w przypadku magistral.
- W przypadku wielokrotnego przypisania w instrukcjach sekwencyjnych stosowany jest jeden nośnik sygnału.

Specyfikacja behawioralna układu

- Opis działania układu poprzez opis sposobu przetwarzania danych za pomocą sekwencji operacji niezbędnych do uzyskania oczekiwanego wyniku przetwarzania.
- **Proces** w VHDL jest sposobem opisu kolejnych kroków działań w przetwarzaniu danych.

Proces (1)

Składnia:

[etykieta:] **process [(lista_czułości)] [is]**

--definicje i deklaracje

begin

--instrukcje_sekwencyjne;

end process [etykieta] ;

- Lista czułości - jest to lista sygnałów, których zmiana powoduje aktywację procesu (ponowne wykonanie wyspecyfikowanych kroków – lista czułości równoważna jest z **wait on** na końcu procesu)
- Przypisanie wartości sygnałów bazuje na bieżącej wartości sygnałów znajdujących się po prawej stronie znaku <=. Modyfikowane sygnały zostaną uaktualnione dopiero po wykonaniu wszystkich instrukcji procesu; uaktualnienie powoduje „zawieszenie" procesu. Sygnały zostają uaktualnione wartością ostatniego przypisania. Zmiany dotyczące obiektów zmiennych (**variable)** są wykonywane natychmiast.

Proces (2)

1. Występuje wewnątrz architektury.
2. Instrukcje wewnętrzne wykonywane są sekwencyjnie
3. Procesy nie mogą być zagnieżdżane.
4. W części deklaracyjnej procesu można definiować: typy, podtypy, stałe, atrybuty i zmienne.
5. W części deklaracyjnej procesu nie można deklarować sygnałów.
6. Instrukcje danego procesu wykonywane w kolejności wystąpienia w kodzie, lecz równolegle z instrukcjami innych procesów.
7. Zgodnie z semantyką języka VHDL sygnał zachowuje wartość jeśli nie następuje do niego przypisanie.
8. Ze względu na pkt 7 warto zatem dla czytelności kodu przypisywać wartości we wszystkich gałęziach konstrukcji **if** lub przypisać im wartość domyślną na początku procesu.

Podstawienie sygnałów

współbieżne

```
A,B,C std_logic;
begin
A<=B;
A<=C;
end
```

B	C	A
0	Z	0
1	Z	1
0/1	1/0	NIEDOZWOLONE
Z	0	0
Z	1	1

szeregowe

```
A,B,C std_logic;
process (clk)
Begin
if clk'event and clk ='1' then
    A<=B;
    A<=C;
    D<=A
end
```

A^+	B	D^+	D^{++}
C	NIEISTOTNE	A	C
X^+ TO WARTOŚĆ X PO 1 ZBOCZU NAR. CLK			
X^{++} TO WARTOŚĆ X PO 2 ZBOCZU NAR. CLK			

Konstrukcje behawioralne i strukturalne - porównanie

Behawioralna (w procesie)

```
if a='0' then if b='0' then y2<='1';
                       else y1<='0';
                       end if;
              else if b='1' then y2<='0';
                                 else
                       y1<='1';
                       end if;
end if;
```

-- przy braku spełnienia warunków wartość sygnału pozostaje bez zmian

strukturalna

```
y2<=(not a and not b)  or
     (y2 and (not a or not b)) ;
y1<=(a and not b) or
     (y1 and (a or not b));
```

a	b	y1	y2
0	0	y1	1
0	1	0	y2
1	1	y1	0
1	0	1	y2

Syganał a zmienna w procesie

```
signal A integer;
variable B integer;
process (A);
A<= A+1;    -- wyrażenie niepoprawne
            -- operacja  „powtarzana „
            -- nieskończenie wiele razy
B := B+1;   -- wyrażenie poprawne
end process;
```

Proces przerzutnika D

```
-- Przerzutnik synchroniczny z
-- zerowaniem synchronicznym
-- kompletny, uruchomiony projekt
library IEEE;
use ieee.std_logic_1164.all;
entity D is       port     (
          D       : in  std_logic;
          CLK     : in  std_logic;
          CLR:     in  std_logic;
          pre     : in  std_logic;
          Q       : out std_logic;
          NQ      : out std_logic );
end d;
```

```
architecture first of D is
begin
Synch_D: process (CLK)
constant low:std_logic:='0';
constant high:std_logic:='1';

begin
    if rising_edge(CLK) then
          if (CLR='0') then
                Q<=low; NQ<= high;
          else if ((CLR='1') and (PRE='0'))
                then
                NQ<=low; Q<= high;
          else
                Q<=D; NQ<= not D;
                end if;
          end  if;
    end if;
end process Synch_D;
end first;
```

Komponent

- Jednostka projektowa raz zdefiniowana i przeznaczona do wielokrotnego wykorzystania w projektach.
- Komponenty można tworzyć w ramach projektu w pliku głównym, plikach dołączonych do projektu lub w bibliotece projektu.
- W celu korzystania z komponentów konieczne są:
  - Definicja komponentu lub wskazanie biblioteki- źródła
  - Deklaracja komponentu w bloku deklaracji architektury jednostki projektowej wykorzystującej komponent (zawiera porty - parametry formalne)
  - Konkretyzacja – specyfikacja (w bloku architektury jednostki) wykorzystania komponentu (zawiera przyporządkowanie do parametrów formalnych parametrów aktualnych związanych z wykorzystaniem komponentu)
- Trzy rodzaje specyfikacji użycia (konkretyzacji) układu: bezpośrednia, powiązania portów pozycją w specyfikacji, powiązania portów nazwami portów.
- Projekty zawierające komponenty do wykorzystania w innych projektach mogą stanowić pakiety przechowywane w bibliotekach.

Przykład wykorzystania komponentu - konkretyzacja

```
architecture strukturalna of uklad1 is
-- deklaracja komponentu w architekturze jego wykorzystania
component mux3_5to1
port( adres,d1,d2,d3,d4,d5,d6,d7,d8: in std_logic_vector(2 downto 0);
wy:out std_logic_vector(2 downto 0)); --  parametry formalne
end component;
-- pozostałe deklaracje
begin
-- użycie jednostki – konkretyzacja komponentu w innym układzie
mult1:mux3_5to1 port map (adr,we0, we1,we2,we3,we4,we5,we6,we7,wy1);
-- są to parametry aktualne związane z użyciem komponentu
-- powiązanie sygnałów do portów poprzez pozycję w deklaracji komponentu
mux2:mux3_5to1 port map (wy =>wy1, adres =>adr, d1=>we0, d2=> we=>,…)
-- powiązanie sygnałów do portów poprzez odwołanie się do nazw portów
-- format: port_komponentu => sygnał_układu
--pozostała część definicji architektury układ1
end strukturalna;
```