## Sposoby projektowania systemów cyfrowych

**Top-down**

- Idea całości projektu
- Dekompozycja na mniejsze bloki
- Projekt i rafinacja podbloków
- Łączenie bloków w całość

**PRZYKŁAD (sumator kaskadowy)**

- zdefiniowanie operacji sumatora kaskadowego
- określenie projektu za pomocą szeregu pełnych sumatorów
- zdefiniowanie pełnego sumatora za pomocą pół-sumatora
- skonstruowanie pełnego sumatora
- połączenie wszystkich bloków





## Sposoby projektowania systemów cyfrowych

**Down-top (bottom-up)**

- Projektowanie najmniejszych elementów
- Łączenie komponentów w bloki funkcjonalne
- Powtarzanie procesu łączenia

**PRZYKŁAD (sumator kaskadowy)**

- Zdefiniowanie, zaprojektowanie, symulacja oraz synteza pełnego sumatora za pomocą pół-sumatora
- Skonstruowanie pełnego sumatora poprzez połączenie dwóch pół-sumatorów
- Zaprojektowanie sumatora n-bitowego dzięki szeregowemu połączeniu n pełnych sumatorów

## Top-down i Bottom-up

Wybór sposobu projektowania zależy od rozmiaru i złożoności projektu, dostępnych zasobów oraz doświadczenia i preferencji projektanta

Wybór stylu top-down

- duże i złożone projekty z małym wykorzystaniem IP core
- we wczesnej fazie projektowej brak możliwości dekompozycji projektu na mniejsze elementy
- brak informacji dotyczącej komponentów na poziomie strukturalnym





## Kod strukturalny

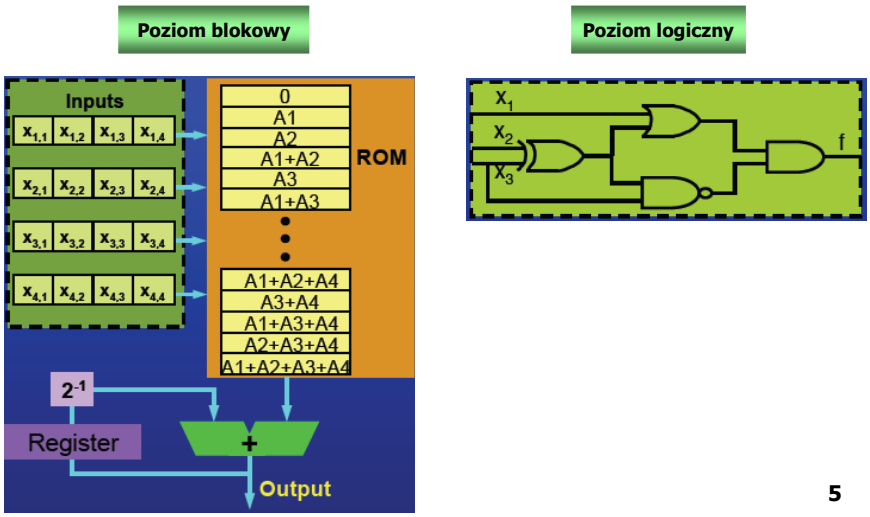
- Możliwość stworzenia hierarchicznego projektu za pomocą zbioru połączonych komponentów
- Komponenty (bloki uprzednio zweryfikowane i zsyntezowane) mogą być umieszczone w bibliotekach
- Uproszczenie procesu rafinacji, uruchamiania i usuwania błędów (mniejszy układ, łatwiejsze uruchamianie)

Tworzenie komponentów:

- Mniejsze komponenty
- Wyrażenia logiczne
- Opis funkcjonalny (behawioralny)

Rodzaje komponentów na różnych poziomach abstrakcji (netlista, RTL)

## Kod strukturalny








## Reprezentacja strukturalna w VHDL

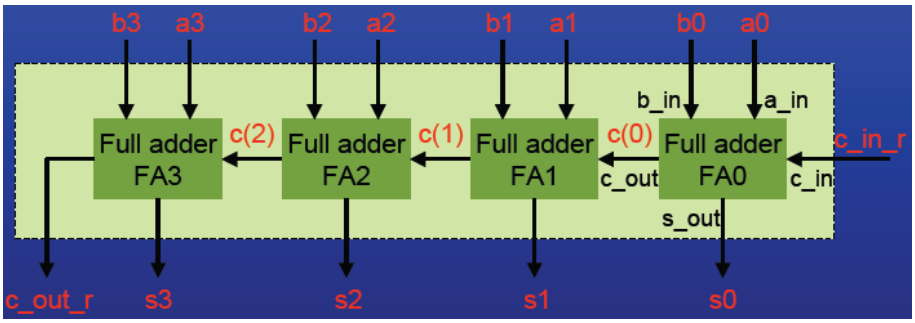
- **Implementacja jednostki projektowej za pomocą specyfikacji połączeń podsystemów**
  - ❖ **Ciało architektury złożone tylko z połączonych podsystemów**

**Elementy strukturalne ciała architektury**

- **deklaracja sygnałów**
- **deklaracja komponentów**
- **konkretyzacja komponentów**

## Przykład architektury strukturalnej

**Czterobitowy sumator kaskadowy**








## Deklaracja komponentów (component)

- **Deklaracja komponentów – specyfikacja zewnętrznego interfejsu pod kątem stałych (*generic*) oraz portów (*port*)**
  - ❖ **brak wymagania implementacji komponentu, konieczność znajomości jego interfejsu**

**Deklaracja komponentu wykorzystuje nazwę, którą posiada jednostka projektowa definiująca komponent, a także taką samą deklarację portów.**

**Istnienie wielu komponentów o tej samej nazwie -> konieczność specyfikacji komponentu do wykorzystania**

## Przykłady deklaracji komponentu

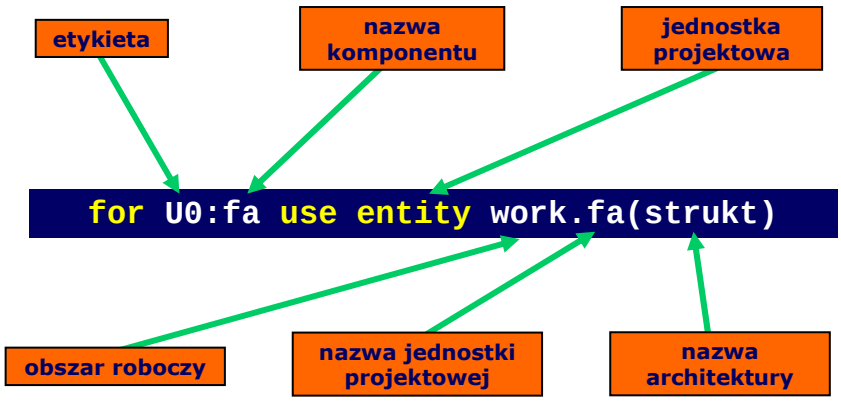
```
architecture struct of ripple4 is
signal c: bit_vector(2 downto 0);

component full_adder
  port(
    a_in: in bit;
    b_in: in bit;
    c_in: in bit;
    s: out bit;
    c_out: out bit
  );
end component;
```

```
architecture struct of reg8 is

component reg_blk
generic(N : integer := 8)
  port(
  clk: in bit;
  d: in bit_vector(0 to N-1);
  q: out bit_vector(0 to N-1)
);
end component;
```

9



## Specyfikacja komponentu




10

## Specyfikacja komponentów i synteza

- Specyfikacja komponentów tylko do celów symulacji
- Narzędzia syntezy ignorują specyfikację komponentów
- Do syntezy projektu używane są najpóźniej skompilowane komponenty

### Konkretyzacja komponentów

- deklaracja komponentu – definicja rodzaju modułu
- konkretyzacja – wykorzystanie modułu w projekcie
- deklaracja komponentu w ciele architektury wraz z deklaracją sygnału (przed *begin*)
- konkretyzacja komponentu w ciele architektury (po *begin*)

Uwaga: komponenty mogą być opisane strukturalnie lub behawioralnie w oddzielnych jednostkach projektowych, traktując je jako oddzielne projekty





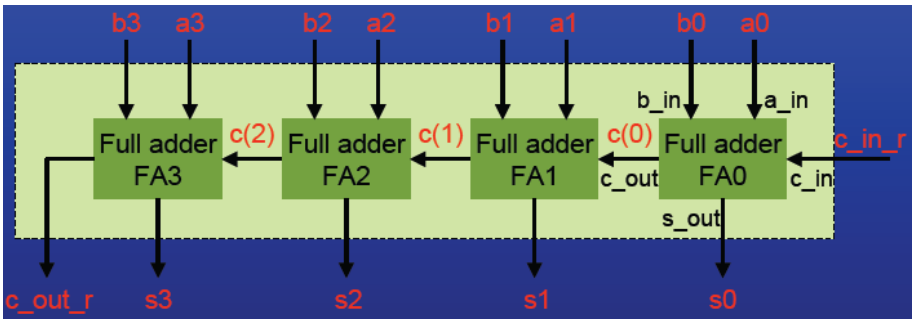
## Konkretyzacja komponentów

```
architecture struct of ripple4 is
--deklaracja komponentu

--specyfikacja komponentu
for U0:fa use entity work.fa(struk)

signal c: bit_vector(2 downto 0);

begin
FA0:fa
  port map(
     a_in => a(0),
     b_in => b(0),
     c_in => c_in_f,
     s_out => s_out_f(0),
     c_out => c(0)
     );
```

```
component fa
        port(
                a_in: in bit;
                b_in: in bit;
                c_in: in bit;
                s: out bit;
                c_out: out bit
        );
end component;
```

## Przykład projektowy

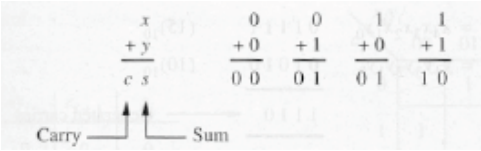
**Czterobitowy sumator złożony z czterech pełnych sumatorów połączonych szeregowo bitem przeniesienia**







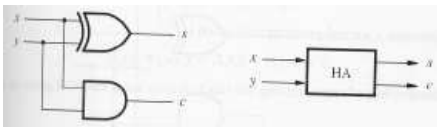

## Konstrukcja pół-sumatora




**4 kombinacje dodawania**

| x | y | Carry c | Sum s |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**tablica prawdy**




**realizacja (struktura)**

## Kod VHDL pół-sumatora

```
library IEEE;
use ieee.std_logic_1164.all;

entity ha is
  port( a, b: in bit;
    sum, c_out: out bit);
end ha;

architecture behav of ha is
begin
  sum <= a xor b;
  c_out <= a and b;
end behav;
```

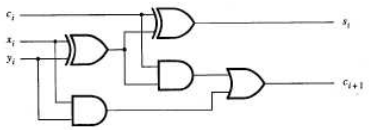




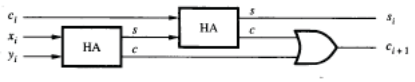
## Konstrukcja pełnego sumatora

| $c_i$ | $x_i$ | $y_i$ | $c_{i+1}$ | $s_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

tablica prawdy



realizacja




konstrukcja z pół-sumatorów

## Kod VHDL pełnego sumatora

```
library IEEE;
use ieee.std_logic_1164.all;

entity fa is
port( x, y,cin: in bit;
s, cout: out bit);
end fa;

architecture struc of fa is

-- deklaracja komponentów
component ha
  port( a, b: in bit;
    sum, c_out: out bit);
end component;

-- specyfikacja komponentów
for U0: ha use entity work.ha(behav);
for U1: ha use entity work.ha(behav);
```





## Kod VHDL pełnego sumatora c.d.

```
-- deklaracja sygnałów
signal s_tmp: bit;
signal c_tmp1, c_tmp2: bit;

begin
-- konkretyzacja komponentów
  U0: ha port map(x, y, s_tmp, c_tmp1);
  U1: ha port map(b => s_tmp,
       a => cin,
       c_out => c_tmp2,
       sum => s);
-- generowanie wyjścia za pomocą przypisania sygnałów
  cout <= c_tmp1 or c_tmp2;
end struc;
```

## Kod VHDL sumatora czterobitowego

```
library IEEE;
use ieee.std_logic_1164.all;

entity adder4 is
   port( x4, y4: in bit_vector(3 downto 0);
      cin4: in bit;
      s4: out bit_vector(3 downto 0);
      cout4: out bit);
end adder4;

architecture struc of adder4 is
-- deklaracja komponentów
component fa
   port(x, y,cin: in bit;
      s, cout: out bit);
end component;
```





## Kod VHDL sumatora czterobitowego c.d.

```
-- specyfikacja komponentów
for U0: fa use entity work.fa(struc);
for U1: fa use entity work.fa(struc);
for U2: fa use entity work.fa(struc);
for U3: fa use entity work.fa(struc);

signal c1, c2, c3: bit;

begin
   U0: fa port map(x4(0), y4(0), cin4, s4(0), c1);
   U1: fa port map(x4(1), y4(1), c1, s4(1), c2);
   U2: fa port map(x4(2), y4(2), c2, s4(2), c3);
   U3: fa port map(x4(3), y4(3), c3, s4(3), cout4);
end struc;
```