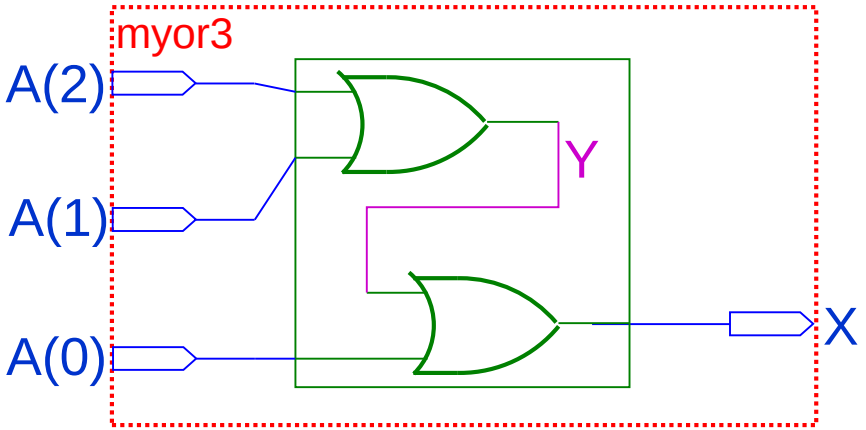
（例）各種宣言：signal宣言

エンティティ名.vhd

```
entity エンティティ名 is
      ジェネリック宣言の記述
      入出力ポートの記述
end エンティティ名;

architecture RTL of エンティティ名 is
各種宣言の記述
begin
      回路動作の記述
end RTL;
```

myor3.vhd



入出力ポートの記述

```
port(A : in std_logic_vector(2 downto 0);  X : out std_logic);
```

各種宣言の記述

```
signal Y : std_logic;
```

信号Yは内部信号
アーキテクチャ部で
のみ用いる信号

回路機能の記述

```
Y <= A(2) or A(1);
X <= Y or A(0);
```

内部信号を用いない記述例

各種宣言の記述

回路機能の記述

```
X <= A(2) or A(1) or A(0);
```

signal宣言文

●signal宣言文：内部信号（アーキテクチャ部でのみ用いる信号）の宣言（定義）

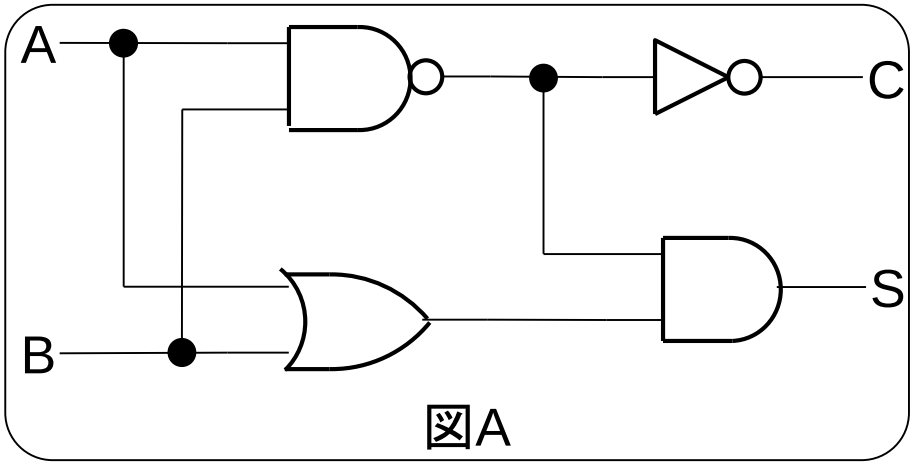
書式 `signal 信号名,信号名,・・・ : データ型 := 初期値(定数);`

「:」以下が同一である複数の信号名は、
コンマで区切り並べて記述できる。

必要ならば、
初期値を設定することができる。

■必須課題■

■プロジェクト「HA1bit」
図AをVHDLを用いて記述し、設計製作フローを進めよ。



図A

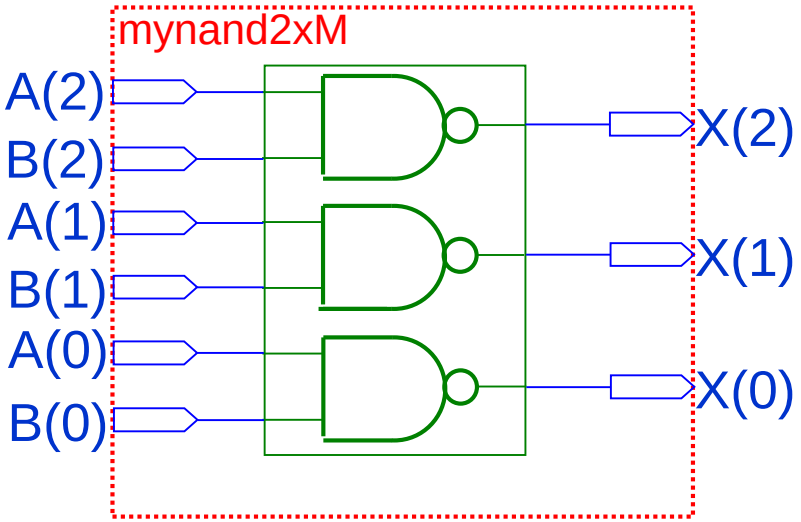
（例）各種宣言：generic宣言

エンティティ名.vhd

```
entity エンティティ名 is
    ジェネリック宣言の記述
    入出力ポートの記述
end エンティティ名;

architecture RTL of エンティティ名 is
各種宣言の記述
begin
    回路機能の記述
end RTL;
```

mynand2xM.vhd



ジェネリック宣言の記述

```
generic(M:integer:=3);
```

入出力ポートの記述

```
port(
  A : in std_logic_vector(M-1 downto 0);
  B : in std_logic_vector(M-1 downto 0);
  X : out std_logic_vector(M-1 downto 0)
);
```

回路機能の記述

```
X <= A nand B;
```

generic宣言文

●generic宣言文：デザイン中のパラメータの共通化宣言

書式 `generic(宣言1; ･･･ 宣言i; ･･･ 宣言N);` ➡ 宣言は、同デザインファイル内で有効

`変数名:データ型:=定数;`

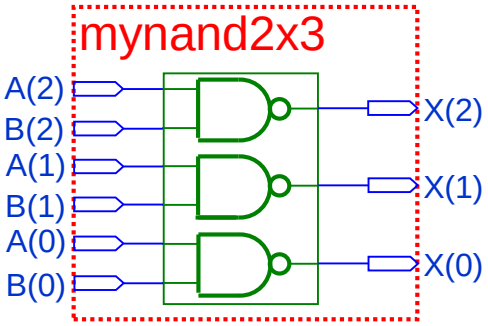
- ➢先頭文字が半角英文字の半角英数文字列。
- ➢大小文字を区別しない。
- ➢VHDL予約語は使えない。

●補足のデータ型

integer
【整数型】
ただし、型のサイズは
論理合成ツールに依存。

●integer型の定数表現
修飾文字なし。10進数値をそのまま記述する。

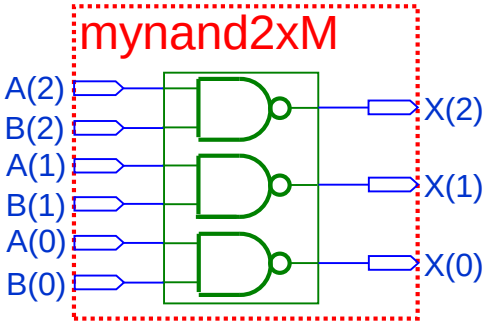
【注】
デザインにおけるgeneric宣言文では、
基本的にinteger型のみで足りる。




```
port(
  A : in std_logic_vector(2 downto 0);
  B : in std_logic_vector(2 downto 0);
  X : out std_logic_vector(2 downto 0)
);
```

抽象化

✓変更が容易になる
✓再利用性が高まる




```
generic(M:integer:=3);
port(
  A : in std_logic_vector(M-1 downto 0);
  B : in std_logic_vector(M-1 downto 0);
  X : out std_logic_vector(M-1 downto 0)
);
```

【注】 を2として
をMとして
も意味は同じだが、
Mがビット幅を
表わさなくなる。

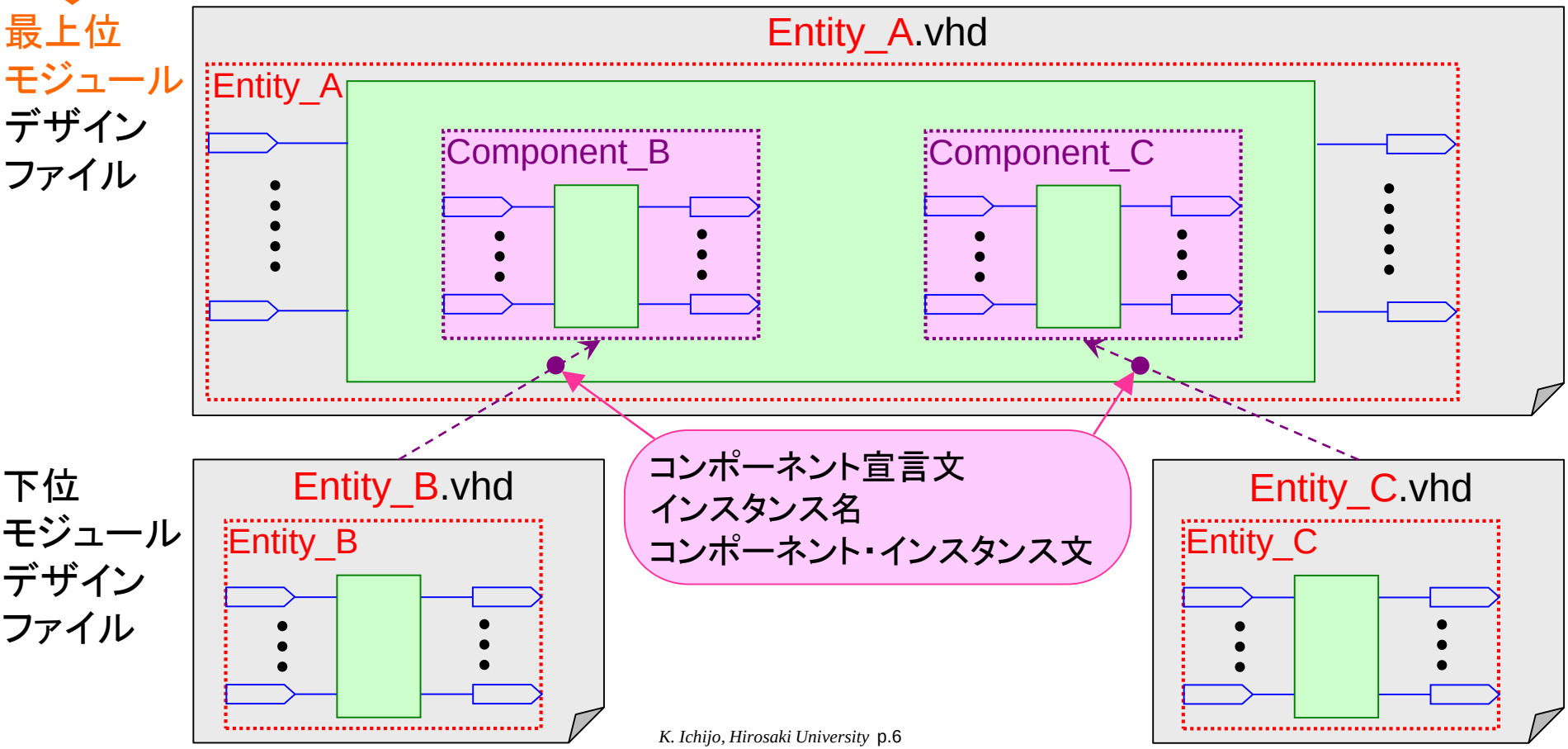
VHDLデザインファイルとモジュール化デザイン

モジュール化デザイン

目的回路を、複数のモジュールに分割して捉え、
各モジュールのデザインファイルを予め用意し、
それらを組み合わせて、目的回路全体のデザインファイルを作成すること。

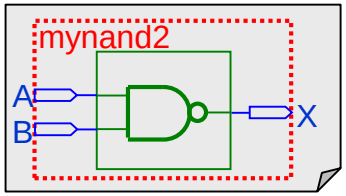
目的回路

（最上位モジュール：目的回路全体を表すモジュール）



（例）モジュール化デザインその1

●既存デザイン（mynand2.vhd）の再利用



【注】
generic宣言を利用していない
→拡張不可能なモジュール化デザイン

入出力ポートの記述

```
port(
  A : in std_logic_vector(2 downto 0);
  B : in std_logic_vector(2 downto 0);
  X : out std_logic_vector(2 downto 0)
);
```

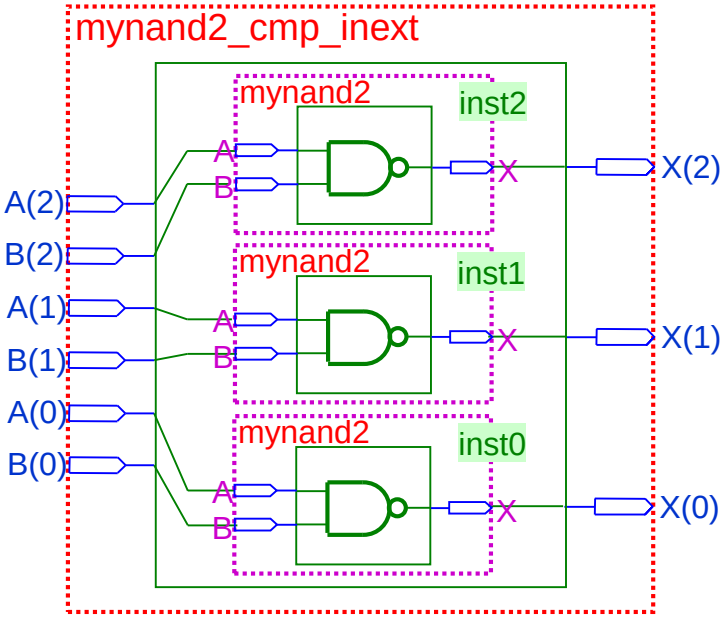
各種宣言の記述

```
component mynand2
  port(
    A : in std_logic;
    B : in std_logic;
    X : out std_logic
  );
end compoment;
```

回路機能の記述

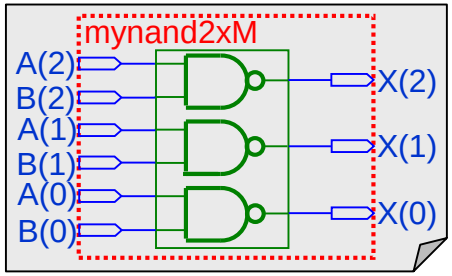
```
ins0 : mynand2
port map(A=>A(0),B=>B(0),X=>X(0));
ins1 : mynand2
port map(A=>A(1),B=>B(1),X=>X(1));
ins2 : mynand2
port map(A=>A(2),B=>B(2),X=>X(2));
```

mynand2_cmp_inext.vhd



（例）モジュール化デザインその2

●既存デザイン（mynand2xM.vhd）の再利用



【注】
generic宣言を利用しているが、
or部分のデザインには活かされていない
→拡張不可能なモジュール化デザイン

ジェネリック宣言の記述

```
generic(M:integer:=3);
```

入出力ポートの記述

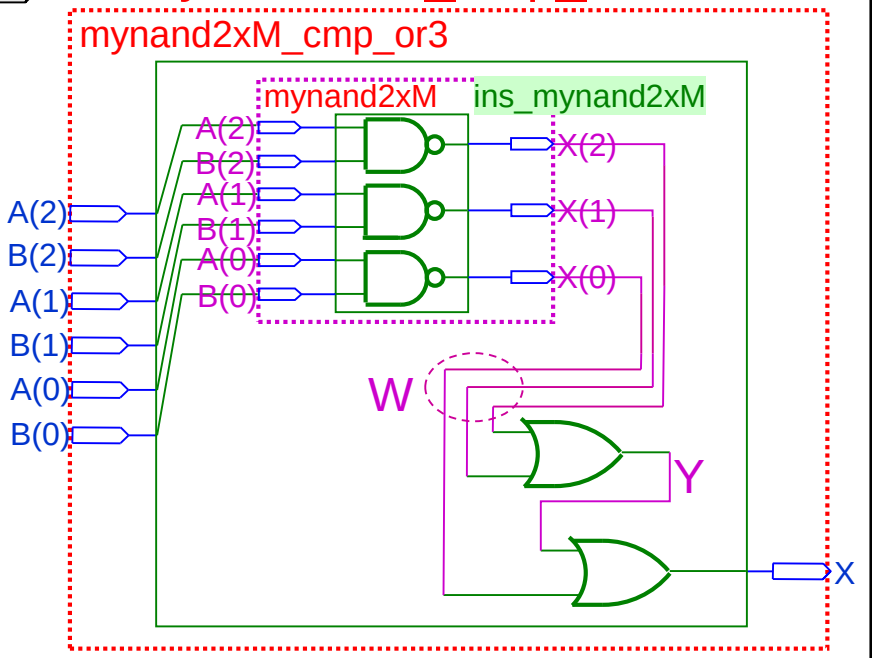
```
port(
  A,B : in std_logic_vector(M-1 downto 0);
  X : out std_logic
);
```

各種宣言の記述

```
component mynand2xM
  generic(M:interger:=3);
  port(
    A : in std_logic_vector(M-1 downto 0);
    B : in std_logic_vector(M-1 downto 0);
    X : out std_logic_vector(M-1 downto 0)
  );
end compoment;

signal Y : std_logic;
signal W : std_logic_vector(M-1 downto 0);
```

mynand2xM_cmp_or3.vhd



回路機能の記述

```
ins_mynand2xM : mynand2xM
generic map(M=>M)
port map(A=>A,B=>B,X=>W);

Y <= W(2) or W(1);
X <= Y or W(0);
```

component宣言文、instance名、component instance文

●component宣言文：既存デザインを使用（再利用）することを宣言

書式

```
component コンポーネント名
  generic(宣言1; ･･･ 宣言i; ･･･ 宣言N);
  port(ポート1; ･･･ ポートi; ･･･ ポートN);
end component;
```

コンポーネントとして用いるデザインにて定めたエンティティ名

コンポーネントとして用いるデザインにて宣言されている通りに書く。

component宣言内のgeneric宣言内の変数名

```
generic map(変数名=>上位変数名,･･･)
```

●コンポーネント・インスタンス文：宣言したコンポーネントの使用法を定める

書式

```
インスタンス名 ： コンポーネント名 ジェネリックマップ ポートマップ;
```

- 先頭文字が半角英文字の半角英数文字列。
- 大小文字を区別しない。
- VHDL予約語は使えない。

- コンポーネント名と異なる名前をつける。
- コンポーネント・インスタンス文が複数ある場合、それらの間で重複しないようにインスタンス名をつける。

```
port map(信号名1=>○,･･･,信号名i=>○,･･･,信号名N=>○)
```

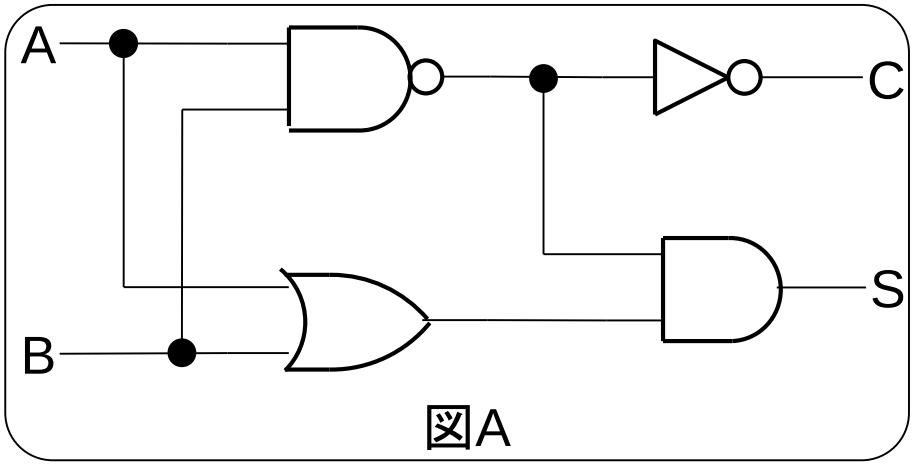
component宣言内のport文中の入出力信号名

上位の入出力信号名または内部信号名

【注】generic map 及び port map 中の「=>」を関係演算子と混同しないように

■必須課題■

■プロジェクト「FA1bit」
プロジェクト「HA1bit」で記述した図Aのデザインファイルを再利用して、
すなわち階層化設計を行って、1ビット全加算器のデザインを記述し、
設計製作フローを進めよ。

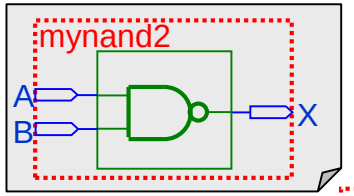


図A

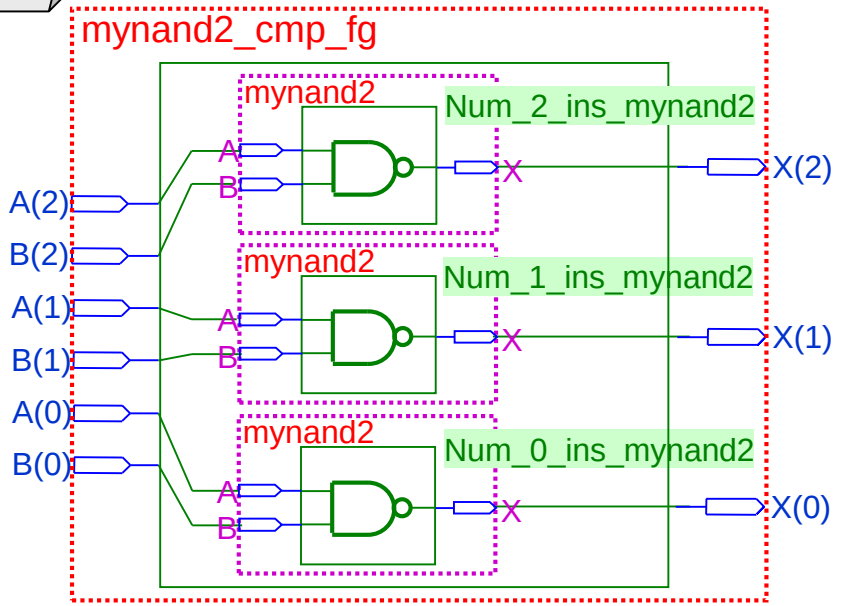
（例）モジュール化デザインその3

●既存デザイン（mynand2.vhd）の再利用
●記述の複製　for-generate文

【注】
generic宣言とfor-generate文を利用
→拡張可能なモジュール化デザイン



mynand2_cmp_fg.vhd



ジェネリック宣言の記述

```
generic(M:integer:=3);
```

入出力ポートの記述

```
port(
  A : in std_logic_vector(M-1 downto 0);
  B : in std_logic_vector(M-1 downto 0);
  X : out std_logic_vector(M-1 downto 0)
);
```

各種宣言の記述

```
component mynand2
  port(
    A : in std_logic;
    B : in std_logic;
    X : out std_logic
  );
end compoment;
```

回路機能の記述

```
Num : for I in 0 to M-1 generate
  ins_mynand2 : mynand2
  port map(A=>A(I),B=>B(I),X=>X(I));
end generate Num;
```

・容易に記述
・拡張可能な記述

```
ins0 : mynand2
port map(A=>A(0),B=>B(0),X=>X(0));
ins1 : mynand2
port map(A=>A(1),B=>B(1),X=>X(1));
ins2 : mynand2
port map(A=>A(2),B=>B(2),X=>X(2));
```

モジュール化デザインその1
mynand2_cmp_inext

⟺ 基本的には右のように記述する。

for-generate文

●for-generate文：「文」を、変数の範囲指定に従って自動的に複製生成する

書式

```
ラベル名 : for 変数 in 変数の範囲 generate
  「文」
end generate ラベル名;
```

- ➢先頭文字が半角英文字の半角英数文字列。
- ➢大小文字を区別しない。
- ➢VHDL予約語は使えない。

- ➢先頭文字が半角英文字の半角英数文字列。
- ➢大小文字を区別しない。
- ➢VHDL予約語は使えない。

- ➢データ型などの宣言は不要

「α以上β以下」（昇順）⇒ α to β
「α以下β以上」（降順）⇒ α downto β

【注】
変数の範囲指定には、MSB/LSBの概念は不要。
従って、どちらかと言えば、高水準言語の流儀に沿ったtoによる指定の方が自然といえる。

●ラベル名が「LB」
●変数の範囲が「α to β」
●文がコンポーネント・インスタンス文で、インスタンス名が「ins_mycomp」

このとき

実際に複製生成される、コンポーネント・インスタンス文のインスタンス名は
「LB_α_ins_mycomp」～「LB_β_ins_mycomp」

間違いではないが、無駄なfor-generate文の使用例：mynand2xM.vhdの回路機能の記述

元々の記述

```
X <= A nand B;
```

明らかにこちらの記述量が少ない

意味は同等

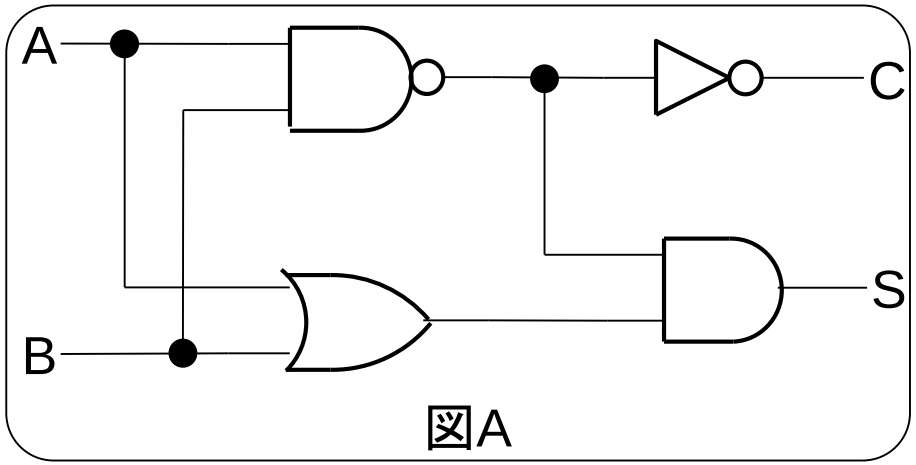
for-generate文を用いた記述

```
Num : for I in 0 to M-1 generate
  X(I)<= A(I) nand B(I);
end generate Num;
```

■必須課題■

■プロジェクト「FA4bit」
プロジェクト「FA1bit」で記述したデザインファイルを再利用して、
すなわち階層化設計を行って、4ビット全加算器のデザインを記述し、
設計製作フローを進めよ。

【注】
・for-generate文を使用せよ。
・ビット数「4」をジェネリック宣言で指定せよ。
・プロジェクト「FA1bit」で記述したデザインファイル自体が、
プロジェクト「HA1bit」で記述した図Aのデザインファイルを再利用
していることに留意せよ。



図A

これまでの例の整理

Top-Level Entity名	スライド	論理回路種別	代入文	演算子	内部信号	generic宣言		モジュール化デザイン
mynand2	VHDL-01	組み合わせ	信号代入	論理	ー	ー		ー
myor3	VHDL-02	組み合わせ	信号代入	論理	○	ー		ー
mynand2xM	VHDL-02	組み合わせ	信号代入	論理	ー	○	値渡しなし	ー
mynand2_cmp_inext	VHDL-02	組み合わせ	信号代入 （下位Level ）	論理	ー	ー		○（その1）
mynand2xM_cmp_or3	VHDL-02	組み合わせ	信号代入 （Top/下位Level ）	論理	○	○	値渡しあり ・Top-Level (signal宣言文) ・下位Level (generic map)	○（その2）
mynand2_cmp_fg	VHDL-02	組み合わせ	信号代入 （下位Level ）	論理	ー	○	値渡しあり ・Top-Level (for-generate文)	○（その3）

バリエーション
なし

先のステップ
で学習

バリエーション
なし

次のステップ
で学習

さまざまなバリエーション
で学習した

以降、適宜応用する

【注】値渡しあり：
generic宣言した変数を、architecture部（各種宣言の記述／回路機能の記述）でも利用していることを意味する。