イベント、同時処理

これまでに学んだ、信号代入文、条件付信号代入文について、
左辺の信号値は、右辺の信号のイベント(値の変化)の発生によって求まる。

- イベントトリガ
- イベント駆動

文の数	内部信号の有無	アーキテクチャ部記述例			
		エンティティ名	抽象度	左辺	右辺
単数	無	mynand2 mynand2xM	論理式	X	A nand B;
		mydec1to2	機能	X	”01” when(A=’0’) else ”10”;
複数	無		論理式	U V	A nand B; not B;
			機能	U V	A when(S=’0’) else B ; C when(A=’1’) else D ;
	有	HA1bit	論理式	Y W C S	A nand B; A or B; not Y; Y and W;
			機能	Y X	A when(S=’0’) else B ; C when(Y=’1’) else D ;

複数の文があるとき、発生したイベントのチェックは、
✓文の順序に関係なく行われる。
✓全ての文に対し、連鎖的に行われる。
⇒ 同時処理

同時処理される各文は、
◆同時処理文と呼ばれる。
◆プロセスと呼ばれる。

順次処理、process文

次の2つの構造を記述できれば、機能記述の幅が広がる。

- ある特定の条件（イベント）判定の後、出力が求まる回路・・・条件分岐（選択構造）を記述
- ある特定の機能を順次繰り返して後、出力が求まる回路・・・繰り返し（反復構造）を記述

これらの構造は、明らかに同時処理でなく、順次処理の形で記述しなければならない。

そのために

A) 順次処理の形で記述する方法 （単に文を並べて記述すると、同時処理文になってしまう・・・）
B) 順次処理を引き起こすイベント指定の方法 （同時処理文では、右辺の信号全てであったが・・・）
C) 選択構造や反復構造自体の記述方法

組み合わせ回路	関係する全ての入力・内部信号を列挙
順序回路	同期信号や制御信号を列挙

A) process文
B) センシティビティリスト
C) if文、case文、for-loop文

```
process（センシティビティリスト）
begin
  if文、case文、for-loop文
end process;
```

順次処理の形で記述する部分

process～end process;
の全体がprocess文

◆同時処理される1つの文
◆1つのプロセス

・・・・ 1つの信号代入文や条件付信号代入文と同じ扱い

（例）組み合わせ回路の機能記述その3

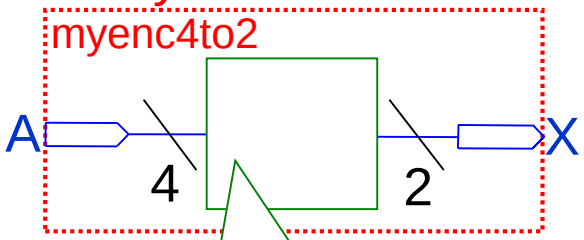
●process文・if文を用いた組み合わせ回路の機能記述

4入力2出力
プライオリティエンコーダ

入出力ポートの記述

```
port(
  A : in std_logic_vector(3 downto 0);
  X : out std_logic_vector(1 downto 0)
);
```

myenc4to2.vhd



回路機能の記述①

```
process(A)
begin
  if(A(3)='1')then
    X<="11";
  elsif(A(2)='1')then
    X<="10";
  elsif(A(1)='1')then
    X<="01";
  elsif(A(0)='1')then
    X<="00";
  else
    X<="ZZ";
  end if;
end process;
```

回路機能の記述②

```
process(A)
begin
  if(A(0)='1')then
    X<="00";
  elsif(A(1)='1')then
    X<="01";
  elsif(A(2)='1')then
    X<="10";
  elsif(A(3)='1')then
    X<="11";
  else
    X<="ZZ";
  end if;
end process;
```

①

入力 A				出力 X	
0	0	0	1	0	0
0	0	1	-	0	1
0	1	-	-	1	0
1	-	-	-	1	1

または

②

入力 A				出力 X	
-	-	-	1	0	0
-	-	1	0	0	1
-	1	0	0	1	0
1	0	0	0	1	1

①と②の違い：
プライオリティの
決め方の違い

if文

●if文

条件を順次チェックし、対応する信号代入文に従って、出力を決定する。

書式

```
if(条件1)then
   信号代入文;
elsif(条件2)then
   信号代入文;
    ⋮
elsif(条件n)then
   信号代入文;
else
   信号代入文;
end if;
```

組み合わせ回路	関係する全ての入力・内部信号を列挙
順序回路	(後述)同期信号や制御信号を列挙

```
process(センシティビティリスト)
begin
```

```
end process;
```

●演算子

【※】信号代入の記号ではない

優先順位別					
	not	*	+	=	and
	abs	/	-	/=	or
	**(冪)	mod	&	<	xor
		rem		>	nand
				<=【※】	nor
				=>	

高 ←優先順位→ 低

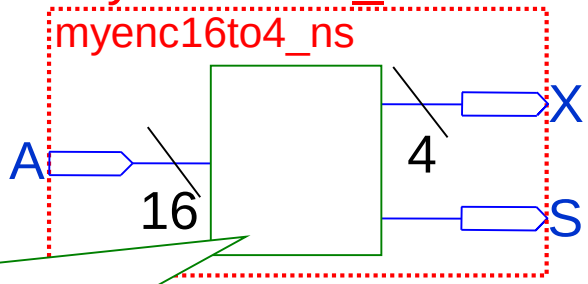
演算機能別	論理	関係	連接	算術
	not	=	&	+
	and	/=		-
	or	<		*
	xor	>		**(冪)
	nand	<=【※】		/
	nor	=>		mod
				rem
				abs

項数別	単項	2項
	not	and
	abs	or
	+	xor
	-	+
		など
		その他
	書式例	
	not A	B and C

■必須課題■

■プロジェクト「myenc16to4_ns」
下の2表のいずれかの動作をする、
16入力5出力（コードは4ビット出力）のプライオリティエンコーダを設計せよ。

myenc16to4_ns.vhd



注1:
プライオリティの決め方は自由
各枠、'0'または'-'

注2:
プライオリティの決め方は自由
各枠、'1'または'-'

入力 A	出力 X				S
15	0	0	0	0	0
14	0	0	0	1	0
13	0	0	1	0	0
12	0	0	1	1	0
11	0	1	0	0	0
10	0	1	0	1	0
9	0	1	1	0	0
8	0	1	1	1	0
7	1	0	0	0	0
6	1	0	0	1	0
5	1	0	1	0	0
4	1	0	1	1	0
3	1	1	0	0	0
2	1	1	0	1	0
1	1	1	1	0	0
0	1	1	1	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0					1

または

入力 A	出力 X				S
15	0	0	0	0	0
14	0	0	0	1	0
13	0	0	1	0	0
12	0	0	1	1	0
11	0	1	0	0	0
10	0	1	0	1	0
9	0	1	1	0	0
8	0	1	1	1	0
7	1	0	0	0	0
6	1	0	0	1	0
5	1	0	1	0	0
4	1	0	1	1	0
3	1	1	0	0	0
2	1	1	0	1	0
1	1	1	1	0	0
0	1	1	1	1	0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1					1

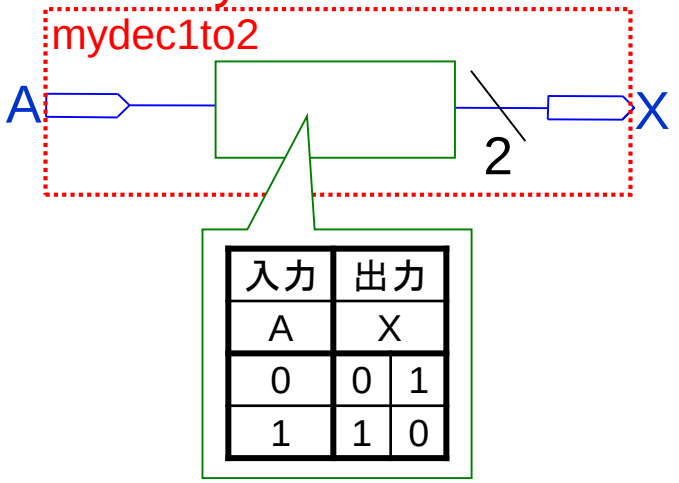
注3：任意に決めてよい。

（例）組み合わせ回路の機能記述その4

●process文・case文を用いた組み合わせ回路の機能記述

1入力2出力デコーダ回路

mydec1to2.vhd



入出力ポートの記述

```
port(
  A : in std_logic;
  X : out std_logic_vector(1 downto 0)
);
```

回路機能の記述

（ふつうの）信号代入文	条件付信号代入文	process文・case文

真理値表 → 論理式
入力 A / 出力 X
0 → 0 1
1 → 1 0
↓
信号代入文
	`X <= "01" when(A='0') else "10";`
	条件付信号代入文
		`process(A)`
		`begin`
		`  case (A) is`
		`  when '0' => X <= "01";`
		`  when others => X <= "10";`
		`  end case;`
		`end process;`

case文

●case文
選択信号の値に対して、
whenで記述された選択肢を順次チェックし、
対応する信号代入文に従って、出力を決定する。

組み合わせ回路	関係する全ての入力・内部信号を列挙
順序回路	（後述）同期信号や制御信号を列挙

書式

```
case （選択信号） is
   when 選択肢 => 信号代入文;
   when 選択肢 => 信号代入文;
     ⋮
   when others => nullまたは信号代入文;
end case;
```

```
process（センシティビティリスト）
begin
  ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
end process;
```

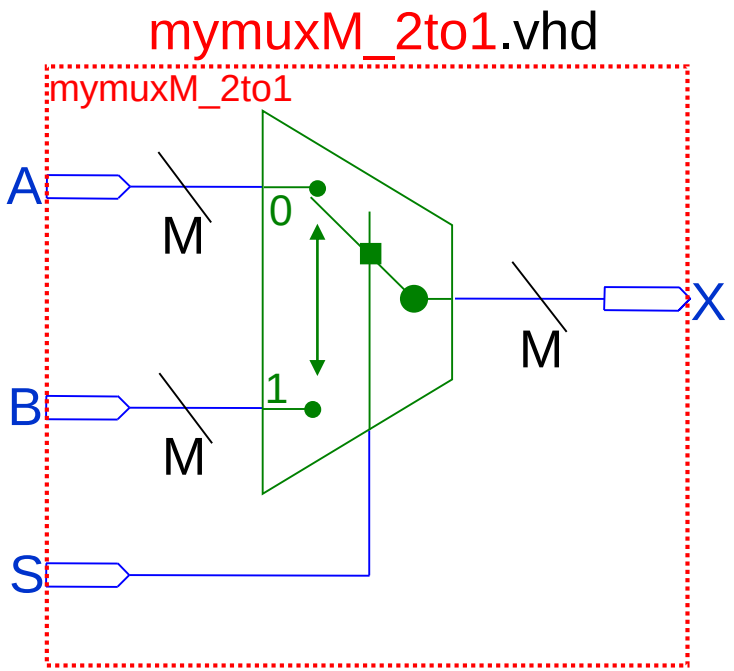
●選択信号の部分に、&演算子で連接した信号を記述する場合は、
　その前に「std_logic_vector'」を付ける。
●選択肢の後の　=>　は関係演算子ではない。
●others選択肢は省略できない。
●others選択肢として記述すべき文がない場合は、nullを記述する。

■必須課題■

■プロジェクト「mymuxM_2to1」
下の表は、
Mビット2入力信号に対するマルチプレクサ（セレクタ）回路の動作表である。
これについて、case文を用いて回路を設計せよ。

なお、Mはジェネリック宣言で指定すること。

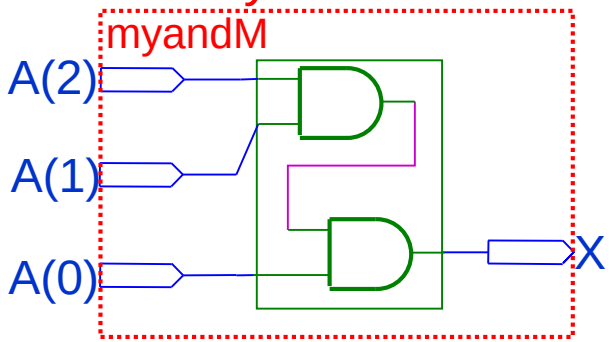
入力	出力
S	X
0	A
1	B



（例）組み合わせ回路の機能記述その5

●process文・for-loop文を用いた組み合わせ回路の機能記述

myandM.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);
   X : out std_logic
);
```

回路機能の記述

各種宣言の記述

```
signal Y : std_logic;
```

内部信号と信号代入文

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

信号代入文

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

process文・for-loop文

```
process(A)
variable tmp : std_logic;
begin
   tmp := A(0);
   for I in 1 to M-1 loop
      tmp := tmp and A(I);
   end loop;
   X <= tmp;
end process;
```

ジェネリック変数利用不能

ジェネリック変数利用可能

for-loop文、変数、変数代入文

●for-loop文：「文」を、変数の範囲指定に従って処理する。

書式

```
for 変数 in 変数の範囲 loop
  「文」
end loop;
```

- 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

組み合わせ回路	関係する全ての入力・内部信号を列挙
順序回路	（後述）同期信号や制御信号を列挙

```
process（センシティビティリスト）
begin
  ：
  ：
end process;
```

【注】
変数の範囲指定には、MSB/LSBの概念は不要。
どちらかと言えば、to による指定の方が自然といえる。

●変数：process文中でのみ有効。

●変数代入文：変数の値を定める。

書式

```
process（センシティビティリスト）
variable 変数名 ： 型 ;
begin
  変数名 := 定数または式;
  --変数代入文はif文、case文、for-loop文の内外に書ける
end process;
```

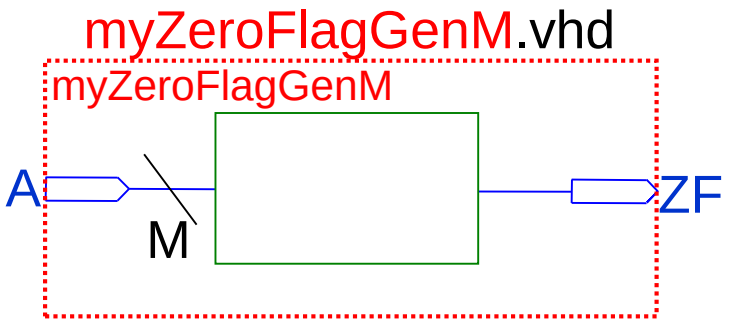
- 先頭文字が半角英文字の半角英数文字列。
- 大小文字を区別しない。
- VHDL予約語は使えない。

■必須課題■

■プロジェクト「myZeroFlagGenM」
下の表は、Mビット入力信号に対するゼロフラグを生成する回路の動作表である。
これについて、for-loop文を用いて回路を設計せよ。

なお、Mはジェネリック宣言で指定すること。

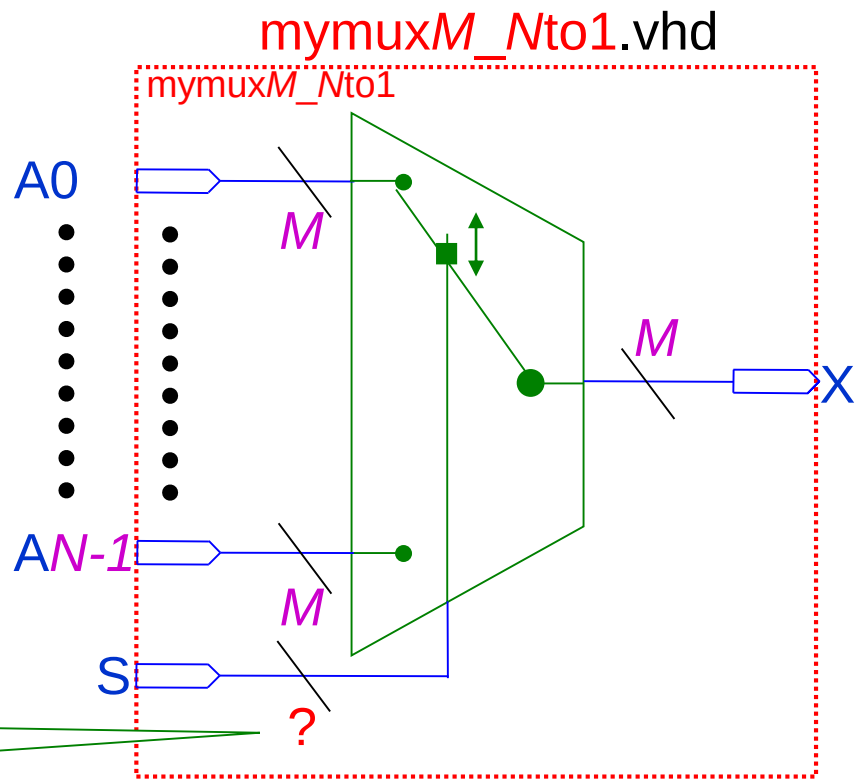
入力	出力
A	ZF
Mビット全て0→ 0 0 ・・・ 0	1
その他	0



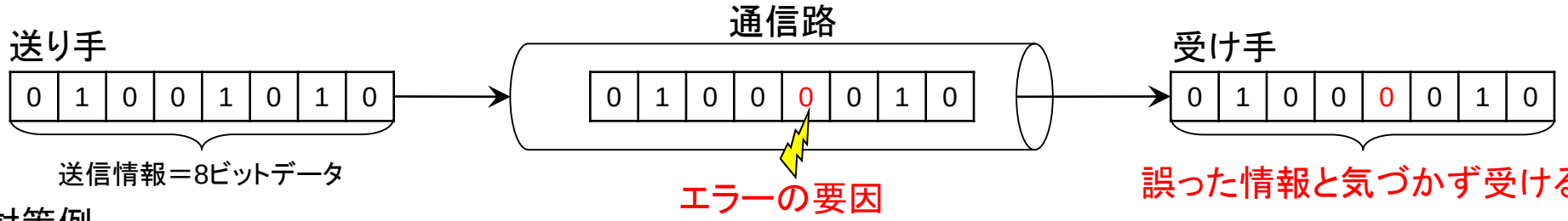
▲追加課題▲

▲プロジェクト「mymux*M*_*N*to1」
M（≧2）と*N*（≧3）を適当に定めたマルチプレクサ回路を、
case文を用いて設計せよ。

なお、
・*M*はジェネリック宣言で指定してもよい。
・*N*は固定した値を用いること。
・*N*が2の冪乗でない場合、
　Sへの入力値が*N*以上のときの
　出力Xの値は、適当に定めてよい。



▲追加課題▲



対策例

1. エラーの要因を取り除く。
2. 誤り検査を行い、誤りが検出されたら再送要求。【基本的方法の一つ】
3. 誤り訂正技術を導入する。

◆1ビットパリティを用いたパリティ検査

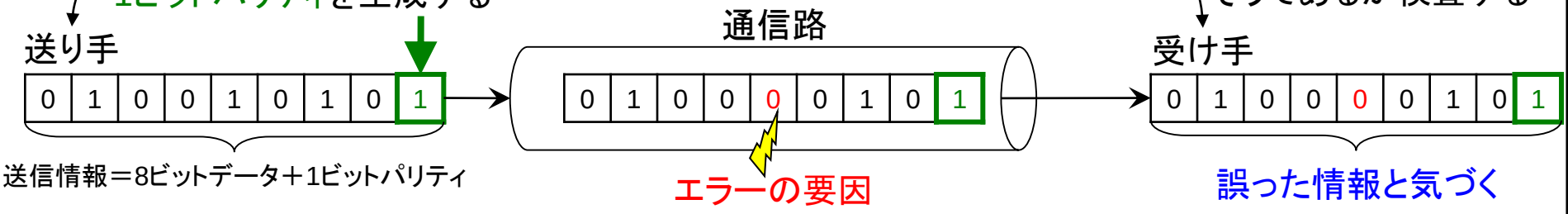
A) データ中の1の数が奇数ならば、パリティビット＝1

B) データ中の1の数が偶数ならば、パリティビット＝1

約束「送信情報には必ず偶数個の1が含まれています」

そうなるように
1ビットパリティを生成する

そうであるか検査する



▲プロジェクト「myparity1genM」
上記の検査方法(A)を採用したとき、
Mビットデータから1ビットパリティを生成する回路を
for-loop文を用いて設計せよ。
なお、Mはジェネリック宣言で指定できるようにすること。

✓ヒント：M=2の場合のデータとパリティの関係を表にしてみよ。
✓1ビットパリティを用いた検査方法は、2ビット以上の誤りは検出できない。
✓対策例の2や3の詳細を含む分野を情報理論、符号理論という。
✓対策例の2や3に相当する技術は見えない所で様々に使われている。

これまでの例の整理

Top-Level Entity名	スライド	論理回路種別	Process文	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	―	―		―
myor3	VHDL-02	組み合わせ	―	信号代入	論理	○	―		―
mynand2xM	VHDL-02	組み合わせ	―	信号代入	論理	―	○	値渡しなし	―
mynand2_cmp_inext	VHDL-02	組み合わせ	―	信号代入 （下位Level ）	論理	―	―		○（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	―	○	値渡しあり ・Top-Level (for-generate文)	○（その3）
myadderM	VHDL-03	組み合わせ	―	信号代入	算術(+)	―	―		―
myshift1	VHDL-03	組み合わせ	―	信号代入	連接(&)	―	―		―
mydec1to2	VHDL-03	組み合わせ	―	条件付信号代入	関係	―	―		―
myenc4to2	VHDL-04	組み合わせ	○(if)	信号代入	関係	―	―		―
mydec1to2	VHDL-04	組み合わせ	○(case)	信号代入	なし	―	―		―
myandM	VHDL-04	組み合わせ	○(for-loop)	変数代入	論理	―	○	値渡しあり (for-loop文)	―

バリエーション
がやや増えた

次のステップ
でさらに学習

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

さまざまなバリエーション
で学習済

適宜応用する