ハードウエア記述言語による LSI設計入門

琉球大学工学部情報工学科
和田　知久　wada@ie.u-ryukyu.ac.jp
吉田　たけお tyoshida@ie.u-ryukyu.ac.jp

本日のアジェンダ

- ＶＨDLというハードウエア記述言語を用いて、論理回路設計を体験する

1. ハーフアダーを例にVHDL言語の説明
2. ハーフアダーの動作をシミュレーション
3. ハーフアダーの回路を合成する
4. ALUの記述の紹介
5. ALU回路の合成
6. ALUの動作をシミュレーション (時間があれば)

UNIXアカウント

- 本日各参加者に琉球大学・情報工学科のアカウントを差し上げます。当面は使用できますので、各大学に帰られた後でもご使用ください。（管理はあなたの責任です）
- UNIXとテキストエディタを使う必要があります経験者を含めたグループを作りたく思います
- 学生、教官でサポートしますので、実習楽しんでください

VHDLの歴史

- 1970年代に米国国防省にてVery High Speed ICプロジェクト発足
- 1981年にVHSICプロジェクトの一環として機能記述言語VHDLが提案
- 1986年にIEEEにてVHDLバージョン7.2を基本言語として採択し、標準化スタート
- 1987年にLanguage Reference Manualをリリースして、IEEE-1076Bとして正式に標準化

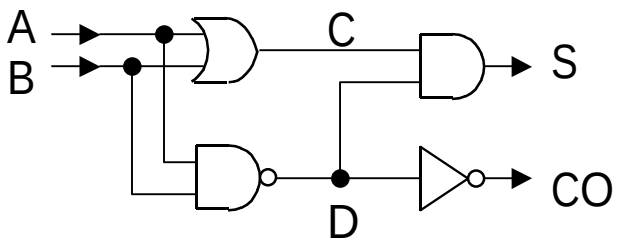
VHDLの適用分野

設計レベル（例：パソコン）	VHDLの記述レベル
マザーボードと周辺機器	アーキテクチャレベル
マザーボード	ビヘビアレベル
Integrated Circuit	RTL（レジスタトランスファレベル）
論理ゲート	ストラクチャレベル

- 本実習では論理合成プログラムによる回路生成が可能なRTLの記述を学ぶ。
- RTLの記述で詳細なデジタル回路の記述ができる。

VHDLの簡単な例題

```
library IEEE;
use IEEE.std_logic_1164.all;
entity HALF_ADDER is
   port ( A, B : in std_logic;
          S, CO : out std_logic);
end HALF_ADDER;
architecture DATAFLOW of HALF_ADDER is
   signal C, D : std_logic;
   begin
   C <= A or B;
   D <= A nand B;
   CO <= not D;
   S <= C and D;
end DATAFLOW;
```



VHDLを簡単に見てゆくと

library IEEE; use IEEE.std_logic_1164.all;	ライブラリ宣言、通常STD_LOGICを使用するので、必ず書く
entity HALF_ADDER is	エンティティ宣言
port (A, B : IN std_logic; S, CO : OUT std_logic);	ポート宣言（入出力信号と信号のデータタイプを定義）
end HALF_ADDER;	エンティティ終了。";"忘れるな！
architecture DATA_FLOW of HALF ADDER is	アーキテクチャ宣言
signal C, D : std_logic;	内部信号を定義
begin C <= A or B ; D <= A nand B ; CO <= not D; S <= C and D; end DATAFLOW;	同時処理文！ コンピュータプログラムとは違う、この場合は4つの同時動作するゲートに対応

std_logicとは

- デジタルの信号を示すデータタイプで、通常‘0’と‘1’を示すが、それ以外に‘X’など、9種類の状態を示す

'U'	Uninitialized	'W'	Weak Unknown
'X'	Unknown	'L'	Weal Low
'0'	Low	'H'	Weak High
'1'	High	'-'	Don't care
'Z'	Hi-impedance		

実習1：ハーフアダーのシミュレーション (1)

- **ツールのセットアップ**

1. PCにloginする。
2. そこから nirai にrloginする。

```
% rlogin nirai
```

3. テキストエディターを使って、自分のホーム・ディレクトリにある .cshrc ファイルの最後の行に以下の1行を付け加える。

```
source    /usr/synopsys/setup_2000.05/init/synopsys_setup_2000.05
```

4. 変更した設定を有効にするために、niraiにrloginし直す。

実習1：ハーフアダーのシミュレーション (2)

- ディスプレイの設定とファイルのコピー

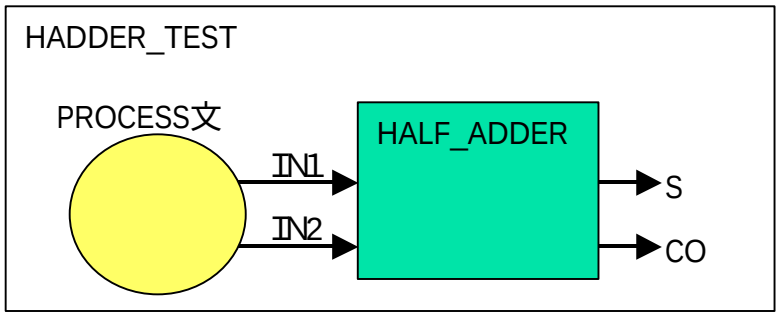
1. PCのローカルなウインドウで、以下のコマンドを実行する
   % xhost nirai
   % xhost nirai-1
2. nirai のウインドウで、以下のコマンドを実行する
   *hostname*はPW01のような自分の使用中のマシン名
   % setenv DISPLAY *hostname*:0.0
3. nirai のウインドウで、作業ディレクトリ (たとえば、vhdl) を作成し、必要なファイルをコピーする
   % mkdir vhdl
   % cd vhdl
   % cp /home/teacher/wada/pub/*.* .

hadder_test.vhd

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

entity HADDER_TEST  is
end HADDER_TEST;

architecture TESTBENCH of HADDER_TEST  is
component HALF_ADDER
      port ( A, B          : in  std_logic;
             S, CO         : out std_logic);
end component;
signal  IN1, IN2,  S, CO  :  std_logic
```




```
begin
   U0: HALF_ADDER port map (IN1, IN2,  S, CO);
   process begin
      IN1<='0' ; IN2<='0' ;
      wait for 10 ns;
      IN1<='0' ; IN2<='1' ;
      wait for 10 ns;
      IN1<='1' ; IN2<='0' ;
      wait for 10 ns;
      IN1<='1' ; IN2<='1' ;
      wait for 10 ns;
      wait;
   end process;
end TESTBENCH;

configuration  CFG_ADDER  of HADDER_TEST is
    for TESTBENCH
    end for;
end CFG_ADDER;
```

実習1：ハーフアダーのシミュレーション (3)

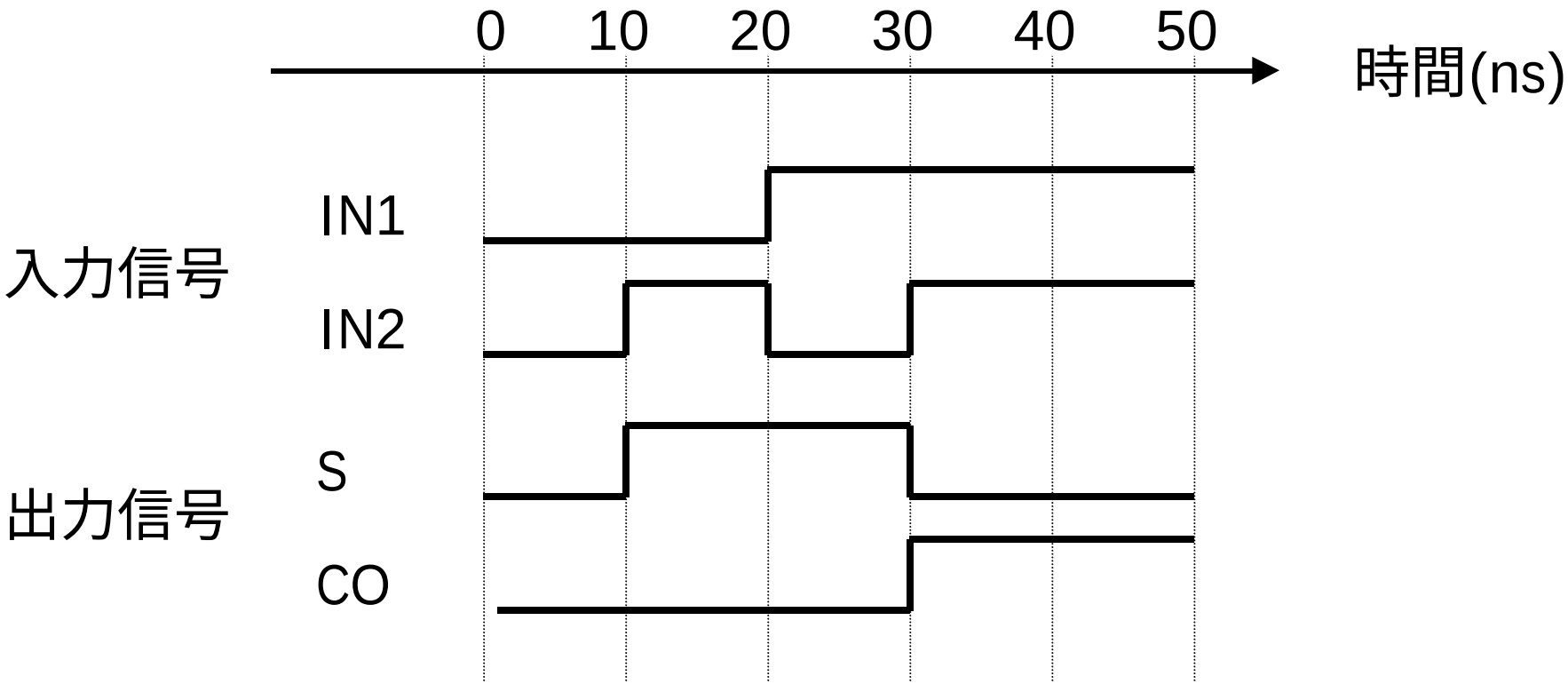
- VHDLファイルの構文解析

1. ２つのVHDL記述の内容を確かめる
   half_adder.vhd　：ハーフアダーの記述
   hadder_test.vhd　：ハーフアダーをテストするための記述：テストベンチ
2. 構文解析する　(VHDLアナライザ)
   % vhdlan　half_adder.vhd
   % vhdlan　hadder_test.vhd

- VHDL Debugger で動作を確認する

1. VHDL Debuggerを起動する
   % vhdldbx　CFG_ADDER &
2. 観測したい信号を指定する
   （VHDL Debugger ウインドウの最下位のＢＯＸで　　trace *’signal
3. シミュレーションの時間を進める
   （VHDL Debugger ウインドウの最下位のＢＯＸで　　run　50
4. 次ページのような波形になればＯＫ！

ハーフアダーの動作波形



実習2: ハーフアダーの回路合成

- design_analyzerの起動とファイルの読み込み

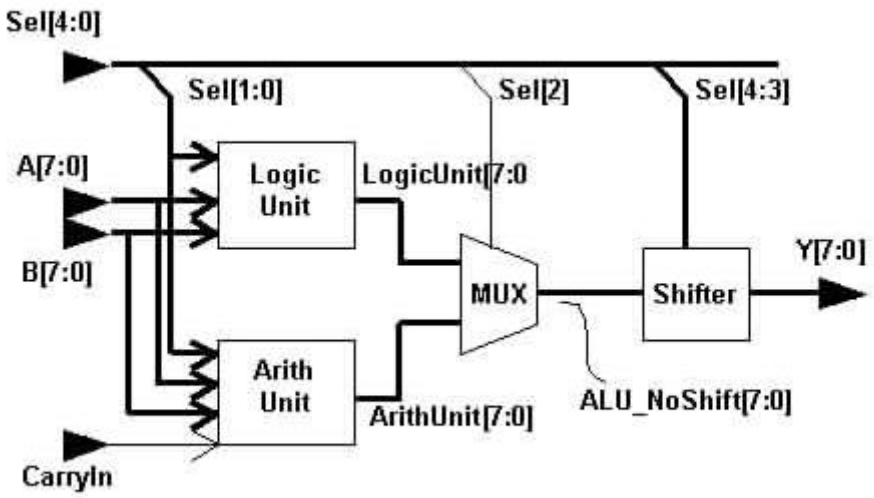
1. デザインアナライザを起動する
   % design_analyzer &
2. メニューからFile->Read…を選択し、
   half_adder.vhd を読み込む
3. 画面中央のHALF_ADDERシンボルをダブルクリックし、1階層下のレベルへ入ると、入出力ピンがわかる
4. さらに中央の箱をダブルクリックし、1階層下のレベルへ入ると、記述どおりのハーフアダー回路が確認できる

- 回路の最適化

1. メニューからTool->DesignOptimizationを選び、OKをクリックする
2. 回路が最適化されたことを確認する

ALUのVHDL記述の紹介

- ALUとは算術（加算や減算）や論理演算を行う回路である
- 今回２つの8ビットの数AとBの入力に対してYを出力する、１４種類の演算を行うＡＬＵの記述を紹介する



ALUの動作

S4	S3	S2	S1	S0	Cin	動作	説明	実行ブロック
0	0	0	0	0	0	Y <= A	Aを転送	Arithmetic Unit
0	0	0	0	0	1	Y <= A+1	Aをインクリメント	Arithmetic Unit
0	0	0	0	1	0	Y <= A + B	加算	Arithmetic Unit
0	0	0	0	1	1	Y <= A + B + 1	キャリー付加算	Arithmetic Unit
0	0	0	1	0	0	Y <= A + Bbar	AとBの１の補数をたす	Arithmetic Unit
0	0	0	1	0	1	Y <= A + Bbar + 1	減算	Arithmetic Unit
0	0	0	1	1	0	Y <= A - 1	デクリメント	Arithmetic Unit
0	0	0	1	1	1	Y <= A	Aを転送	Arithmetic Unit
0	0	1	0	0	0	Y <= A and B	論理積	Logic Unit
0	0	1	0	1	0	Y <= A or B	論理和	Logic Unit
0	0	1	1	0	0	Y <= A xor B	排他的論理和	Logic Unit
0	0	1	1	1	0	Y <= Abar	１の補数	Logic Unit
0	0	0	0	0	0	Y <= A	Aを転送	Shifter Unit
0	1	0	0	0	0	Y <= shl A	Aを左シフト	Shifter Unit
1	0	0	0	0	0	Y <= shr A	Aを右シフト	Shifter Unit
1	1	0	0	0	0	Y <= 0	0を転送	Shifter Unit

alu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity ALU is
   port(Sel     :  in  unsigned(4 downto 0);
         CarryIn :  in  std_logic;
         A, B    :  in  unsigned(7 downto 0);
         Y       :  out unsigned(7 downto 0) );
end entity ALU;

architecture COND_DATA_FLOW of ALU is
begin

   ALU_AND_SHIFT:
   process (Sel, A, B, CarryIn)
        variable Sel0_1_CarryIn  : unsigned(2 downto 0);
        variable LogicUnit, ArithUnit,
                      ALU_NoShift     : unsigned(7 downto 0);
   begin
        --------------------------
        -- Logic Unit
        --------------------------
        LOGIC_UNIT: case Sel(1 downto 0) is
           when "00"    => LogicUnit := A and B;
           when "01"    => LogicUnit := A or B;
           when "10"    => LogicUnit := A xor B;
           when "11"    => LogicUnit := not A;
           when others  => LogicUnit := (others => 'X');
      end case LOGIC_UNIT;
        --------------------------
        -- Arithmetic Unit
        --------------------------
        Sel0_1_CarryIN := Sel(1 downto 0) & CarryIN;
        ARITH_UNIT: case Sel0_1_CarryIn is
           when "000"   => ArithUnit := A;
           when "001"   => ArithUnit := A+1;
           when "010"   => ArithUnit := A+B;
           when "011"   => ArithUnit := A+B+1;
           when "100"   => ArithUnit := A + not B;
           when "101"   => ArithUnit := A-B;
           when "110"   => ArithUnit := A-1;
           when "111"   => ArithUnit := A;
           when others  => ArithUnit := (others => 'X');
      end case ARITH_UNIT;
        --------------------------
        -- Mutiplex
        --------------------------
        LA_MUX: if (Sel(2) = '1') then
           ALU_NoShift := LogicUnit;
      else
           ALU_NoSHift := ArithUnit;
      end if LA_MUX;
        --------------------------
        -- Shift operation
        --------------------------
        SHIFT: case Sel(4 downto 3) is
           when "00"    =>  Y <= ALU_NoSHift;
           when "01"    =>  Y <= Shift_left(ALU_NoShift, 1);
           when "10"    =>  Y <= Shift_right(ALU_NoShift, 1);
           when "11"    =>  Y <= (others => '0');
           when others  =>  Y <= (others => 'X');
      end case SHIFT;
   end process ALU_AND_SHIFT;
end architecture COND_DATA_FLOW;
```

実習3：ALUの回路合成

- design_analyzerの起動とファイルの読み込み

1. デザインアナライザを起動する
   % design_analyzer &
2. メニューからFile->Read を選択し、
   alu.vhd を読み込む
3. 画面中央のALUシンボルをダブルクリックし、1階層下のレベルへ入ると、入出力ピンがわかる
4. さらに中央の箱をダブルクリックし、1階層下のレベルへ入ると、記述をシンボル化したALU回路が確認できる

- 回路の最適化

1. メニューからTool->DesignOptimizationを選び、OKをクリックする
2. 回路が生成されたことを確認する

test_alu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity TEST_ALU is
end TEST_ALU;

architecture TESTBENCH of TEST_ALU is
component ALU
   port ( Sel      : in   unsigned(4 downto 0);
          CarryIn  : in   std_logic;
          A, B     : in   unsigned(7 downto 0);
          Y        : out  unsigned(7 downto 0));
end component;
signal   SIN          : unsigned(4 downto 0);
signal   CIN          : std_logic;
signal   AIN,BIN,YIN   : unsigned(7 downto 0);
begin
        U0: ALU port map (SIN,CIN,AIN,BIN,YIN);
        process begin
         AIN <= "00001111";  BIN <= "11110000";
           wait for 10 ns;
           SIN <= "00000"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00000"; CIN <= '1';
           wait for 10 ns;
           SIN <= "00001"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00001"; CIN <= '1';
           wait for 10 ns;
           SIN <= "00010"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00010"; CIN <= '1';
           wait for 10 ns;
           SIN <= "00011"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00011"; CIN <= '1';
           wait for 10 ns;
           SIN <= "00100"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00101"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00110"; CIN <= '0';
           wait for 10 ns;
           SIN <= "00111"; CIN <= '0';
           wait for 10 ns;
           SIN <= "01000"; CIN <= '0';
           wait for 10 ns;
           SIN <= "10000"; CIN <= '0';
           wait for 10 ns;
           SIN <= "11000"; CIN <= '0';
           wait for 10 ns;
           wait;

        end process;
end TESTBENCH;

configuration CFG_ALU of TEST_ALU is
   for TESTBENCH
   end for;
end CFG_ALU;
```

(オプショナル) 実習4：ALUのシミュレーション

- VHDLファイルの構文解析

1. ２つのVHDL記述の内容を確かめる
   alu.vhd　：ALUの記述
   test_alu.vhd　：ALUをテストするための記述：テストベンチ
2. 構文解析する　(VHDLアナライザ)
   % vhdlan alu.vhd
   % vhdlan test_alu.vhd

- VHDL Debugger で動作を確認する

1. VHDL Debuggerを起動する
   % vhdldbx　CFG_ALU &
2. 観測したい信号を指定する
   (VHDL Debugger ウインドウの最下位のＢＯＸで　　trace *' signal
3. シミュレーションの時間を進める
   RUNボタンの右横のＢＯＸに、２０　と記入し、RUNボタンを数回押す

これで終了です！

- VHDLは如何でしたか？
- VHDL以外にVerilogHDLというハードウエア記述言語もあり、この2つが現在有名です

- 研修ご苦労さまでした！