２．論理設計の基礎

1. 目的

必要な機能をユーザ (回路設計者) 自身がプログラム可能な論理 IC であるプログラマブル・ロジック・デバイス (**P**rogrammable **L**ogic **D**evice, 以下 **PLD**) とハードウェア記述言語 **VHDL** を用い，簡単な論理回路を対象としたディジタル回路の設計法を理解する．

2. プログラム可能な論理 IC ~ PLD ~

2.1. PLD の特徴

PLD は，その名のとおり「ユーザ自身が設計したロジック回路を実チップとして動作させることが可能となるプログラマブル・デバイス」であり，現在のディジタル回路設計の世界では，PLD を抜きに語れない時代になっている．比較的最近までは，ASIC の試作のため，あるいは少量生産システムでのみ利用されていたが，最近ではあらゆる組み込み機器で利用されていると言っても過言ではない状況になってきている．また，量販店で販売されているような電気製品にも採用されるようになってきており，多種の PLD が発売され用途に応じて使い分けできるようになってきている．PLD は，大別して CPLD(Complex PLD) と FPGA(Field Programmable Gate Array) がある．

2.2. PLD の基本構造

PLD の内部には，プログラム可能なマクロ・セルと呼ばれる素子と，マクロ・セル間を接続するための配線リソースが多数内蔵されている．PLD の内部構造のイメージを図 1 に示す．

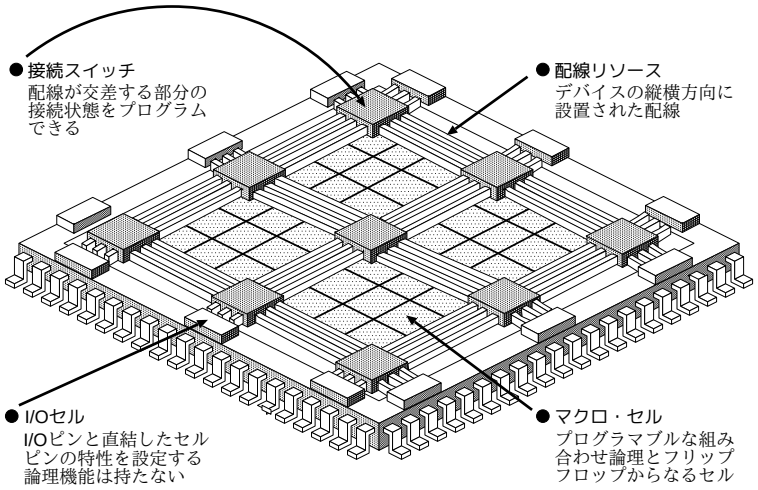


図 1: PLD の内部構造例

● マクロ・セル

マクロ・セルは，組み合わせロジックを実現するプログラム可能なロジック・モジュールと，順序回路を構築するための D フリップフロップ[1]から構成されている．

ユーザが設計した回路は，PLD 設計・開発用ツールによって最終的にマクロ・セル単位に分割される．各マクロ・セル内のロジック・モジュールは，その分割された回路が機能するようにプログラムされる．

● 配線リソース

[1]一種の記憶回路であり，Cp にクロック入力が加えられた瞬間に D 入力の 0/1 の状態が Q 出力にセットされる．

分割されたそれぞれの回路を含むマクロ・セル間は，配線リソースを介して接続される．この配線リソースはあらかじめデバイスの縦横方向に用意されている．

● Ｉ／Ｏセル

デバイスのＩ／Ｏピンと直結したセルである．Ｉ／Ｏセルではデバイス内部のマクロ・セルとは異なり，組み合わせロジックは構成できない．

3. VHDL の概要

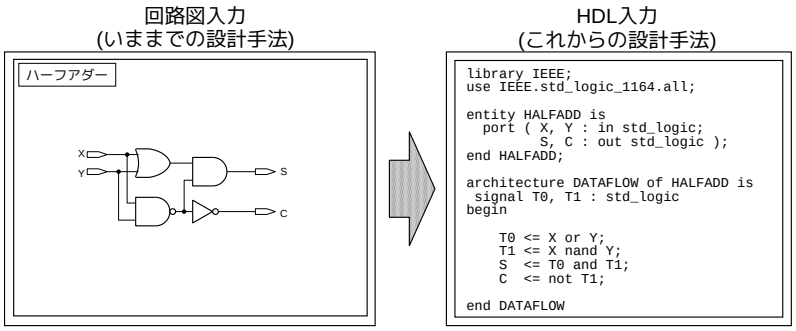
3.1. HDL 設計のメリット

ハードウェア記述言語 (Hardware Description Language, 以下 HDL) は，その名のとおりハードウェアの動作を記述する一種の言語である．

HDL による設計手法は，現在すでに **ASIC**(**A**pplication **S**pecified **I**ntegrated **C**ircuit：特定用途向け集積回路) などの大規模集積回路の設計でさかんに利用されており，FPGA や CPLD などを使用する比較的小規模な設計にも様々なメリットをもたらす．表 1，図 2 に今までの回路図入力による設計と HDL による設計の比較を示す．

表 1: 回路図入力，HDL 記述各入力による設計

	回路図入力	HDL 入力
1	回路図入力に時間がかかる	テキストで簡単に入力
2	回路変更が困難	回路変更が容易
3	設計者以外では内容を理解しづらい	動作の記述のため内容を理解しやすい
4	特定の半導体メーカのライブラリを考慮して回路図入力	半導体メーカのライブラリに依存せず，どのメーカにも容易に対応可



ハーフアダー(半加算器)の真理値表

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

図 2: 回路図入力，HDL 記述各入力による設計 (ハーフアダー)

3.2. HDL の種類

HDL には，代表的なものとして **VHDL** と **Verilog-HDL** がある．各々を比較した場合，様々な立場によって優劣が左右されるが，本実験では VHDL を使用する．

3.3. VHDL の特徴

VHDL は，さまざまなレベルでの記述が可能である．たとえば，システム全体のアルゴリズムを記述したり (アーキテクチャ・レベルでの記述)，ハードディスクのデータのやり取りや，モータの制御などをモデル化してシステム全体を抽象度の高いレベルで記述したり (ビヘイビア・レベルの記述)，ロジック回路生成が可能なレベルで記述したり (RTL : Register Transfer Level での記述) できる．もちろんロジック・ゲートレベルでの記述も可能である．

4. 実験概要，使用装置

本実験では、教育用端末の Windows 上で動作する **ALTERA 社製 Quartus II 開発ソフトウェア**を使用し、ディジタル回路の設計，合成および論理シミュレーション (Quartus II での動作シミュレーション) を行う．その後、合成した回路を**同アルテラ社製 Altera DE0 開発・学習ボード**に転送し，実際に DE0 ボード上で設計した回路の動作確認を行う．

また、本実験での回路の設計には、**VHDL** と呼ばれるハードウェア記述言語を使用する．今まではディジタル回路の設計には、設計する回路の論理式などから **and** や **or** などのゲートを並べてそれらの間を結線することにより設計されてきた．しかし近年では、**VHDL** や **Verilog-HDL** のようなハードウェア記述言語により、ディジタル回路をソフトウェア的なスタンスから設計できるようになっている．

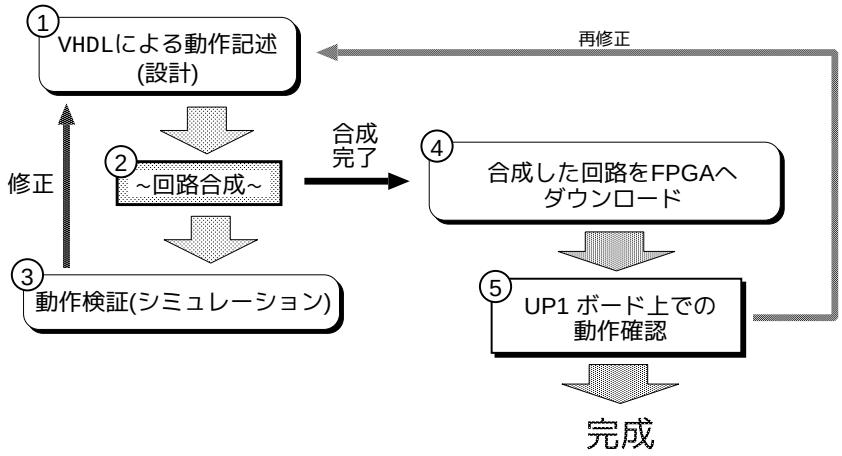


図 3: 本実験でのディジタル回路設計の流れ

5. 実験

5.1. 実験内容

本実験では，ディジタル回路の設計方法を理解することを目的として、2 週間で以下の 3 種類の回路について回路設計～回路合成～論理シミュレーション～DE0 ボード上での動作確認まで行う．

- **1 週目**

  【1】真理値表を元に全加算器を設計 (プログラム記述) し，Quartus II 上で論理シミュレーションを行う．論理シミュレーションにより正しい回路が設計されていることを確認した後，設計した回路を UP-1 ボードへダウンロードし，ボード上で動作確認を行う．

  【2】【1】で設計した全加算器を 4 つ用いた，図 4 に示す 4 ビット並列形加算器 (4 ビットリップル加算器) の設計，論理シミュレーションおよび UP-1 ボードでの動作確認を行う．

- **2 週目**

  【3】真理値表を元に 7 セグメント LED と 8P ディップスイッチを使用した 2 進-10 進デコーダの設計および論理シミュレーション，UP-1 ボードでの動作確認を行う．

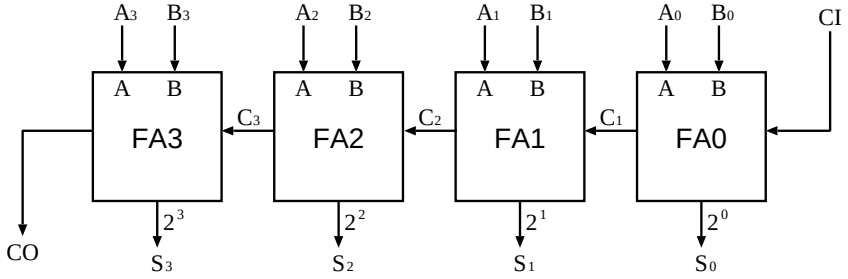


図 4: 4 ビット並列形加算器

以降、本実験テキスト、「Quartus II の使い方」、「DE0 開発・学習ボードの使い方」を合せて参照しながら実験を進めていく．

5.2. 全加算器の設計

(i) 真理値表・論理式

全加算器を設計するために，最初に真理値表 (表 2) から必要な論理式を導き出す．表 2(a) は，普通使われている 0/1 による '**正論理**' の真理値表，表 2(b) は，負論理動作も考慮して信号を L/H 表現とした真理値表である．

表 2 から、全加算器の論理式は次のように導き出される．

$$
\begin{aligned}
S &= X\cdot Y\cdot CI + \overline{X}\cdot\overline{Y}\cdot CI + \overline{X}\cdot Y\cdot\overline{CI} + X\cdot\overline{Y}\cdot\overline{CI} \\
&= (X\cdot Y + \overline{X}\cdot\overline{Y})\cdot CI + (\overline{X}\cdot Y + X\cdot\overline{Y})\cdot\overline{CI} \\
&= \overline{(X\oplus Y)}\cdot CI + (X\oplus Y)\cdot\overline{CI} \\
&= (X\oplus Y)\oplus CI \\
CO &= X\cdot Y\cdot CI + \overline{X}\cdot Y\cdot CI + X\cdot\overline{Y}\cdot CI + X\cdot Y\cdot\overline{CI} \\
&= X\cdot Y\cdot(CI+\overline{CI}) + (\overline{X}\cdot Y + X\cdot\overline{Y})\cdot CI \\
&= X\cdot Y + (X\oplus Y)\cdot CI
\end{aligned}
$$

(·: 論理積、　+: 論理和、　$\oplus$: 排他的論理和)

表 2: 全加算器の真理値表

(a) 0/1 表現 (正論理)

X	Y	CI	S	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

(b) L/H 表現

X	Y	CI	S	CO
L	L	L	L	L
H	L	L	H	L
L	H	L	H	L
H	H	L	L	H
L	L	H	H	L
H	L	H	L	H
L	H	H	L	H
H	H	H	H	H

以上の論理式および表 2 での入出力名を使用し、VHDL により記述 (設計) された全加算器の記述例をサンプル 1 に示す.

```
-- (サンプル 1 ) 全加算器
library IEEE;
use IEEE.std_logic_1164.all;

entity FULLADDER is
  port( A, B, CI : in std_logic;
        S, CO    : out std_logic );
end FULLADDER;

architecture DATAFLOW of FULLADDER is
begin

  S  <= (A xor B) xor CI;
  CO <= (A and B) or ( (A xor B) and CI);

end DATAFLOW;
```

('- -' で始まる行はコメント行)

(ii) Quartus II による回路設計手順

前節のサンプルを用い (正しい論理式であれば、上の同じでなくても良い)，Quartus II でディジタル回路の設計・合成・シミュレーションを行う．次に述べる設計時の各手順については、別紙「Quartus II の使い方」も参照しながら行う．

【1】作業用フォルダの作成 (「**Quartus II の使い方**」 **P4)**

コンピュータの画面の左下の丸い部分 (スタート) から「コンピュータ」を開き、さらにその中の「ネットワークの場所」にある windows ディスクを開き、その中に班のアルファベット名のフォルダを作成する．日本語等の全角文字は使用できません．

【2】**Quartus II** の起動 (「**Quartus II** の使い方」 **P5)**

デスクトップ上の 「Quartus II」という名前の青いアイコンをダブルクリックして起動．

(重要！) デスクトップ上には **Quartus II** のアイコンが２つあるが、必ず **Quartus II 9.1** の方を使用する．

【3】プロジェクトの設定 (「**Quartus II** の使い方」 **P7~ P15)**

設計する回路の名前を定義する．「Quartus II の使い方」に沿って作業すること．

- (使い方 P6) : プロジェクト名は '**fulladder**' とします。プロジェクトは、作成した作業用フォルダ内に設定する．
- (使い方 P7) : そのまま「Next」をクリック
- (使い方 P8) : ウインドウ上の Family: は **Cyclone III** を選択し、その後下の Available devices: で **EP3C16F484C6** を選択して「Next」をクリック。
- (使い方 P9) : そのまま「Next」をクリック

【4】回路設計 **(HDL** 記述入力**)** (「**Quartus II** の使い方」 **P16~ P17)**

サンプルプログラムを VHDL 入力ウィンドウ (エディタ) を用いて入力する．一行目のコメント文 「**–(サンプル 1) 全加算器**」 は記述しない．プログラムは '**fulladder.vhd**' として保存する．

【5】設計回路の合成 (「**Quartus II** の使い方」 **P18~ P19)**

ここで、設計 (入力) したプログラムのコンパイルを行い、プログラム内の入出力宣言や論理式から、実際の回路 (and や or などのゲートにより構成される回路) に合成する．何らかのエラーが出た場合には，エラーの箇所を修正しエラーが出なくなるまで再度コンパイルしてチェックする．このとき，幾つかのワーニングが出る場合があるが，この課題ではとりあえずワーニングは無視．

【6】入出力ピンの配置 (「**Quartus II** の使い方」 **P20~ P22)**

設計 (合成) される回路は、最終的には DE0 ボード上の FPGA に書き込まれ、その後動作確認が行われる．しかし動作確認を行うためには，FPGA の適切なピンにより信号の入出力しなければならず、ここではそのためのピンの割り当てを行う．「Quartus II の使い方」には二種類の割り当て方法が記載してあるが，どちらか一方のやり易い方で実行する．ピン配置が終ったら、再度コンパイルを実行する．この課題では，表 3 に示すピン番号を使用する．

【7】論理シミュレーション (「**Quartus II** の使い方」 **P23~ P32)**

Quartus II のシミュレーション・ツールを用い、入力した回路の論理シミュレーションを行う．ここでのシミュレーション結果において真理値表どおりの結果が得られなかった場合は，再びプログラムの変更と回路合成を行い (場合によってはピンの再配置も)，再度シミュレーション結果を確認する．なお、シミュレーションの際の **Grid Size**' は'**20ns**'、'**End Time**' は '**200ns**' とする事．論理シミュレーションの入力データは真理値表の入力信号を用い、各入力を 1 グリッド毎に順番に入力する．

表 3: 全加算器のピン配置

信号名	ピン番号	機能
X	J6	スライドスイッチ SW0
Y	H5	スライドスイッチ SW1
CI	H6	スライドスイッチ SW2
S	J1	LED0
CO	J2	LED1

【8】設計した回路の **DE0** ボードへの実装 (「**Quartus II** の使い方」**P33~ P36)**

以上の作業が終ったら、Quartus II からパソコンに接続された DE0 ボードへ回路の転送 (ダウンロード) を行う．その後、ピン配置したスライドスイッチと LED を用いて動作確認を行う．ここで目的の動作が行なわれることが確認されたら，この回路についての一通りの設計が終了．

(注) 「**Quartus II** の使い方」**28** ページ冒頭にあるジャンパの設定については無視して下さい。

最後の動作確認の際は，担当の方に確認してもらうこと．
また，印刷したプログラム、シミュレーション結果は、レポート提出の際に添付する．

5.3. 4 ビット並列加算器の設計

前で設計した全加算器をサブ・モジュールとして利用し，図 4 に示す 4 ビット並列加算器を設計する．記述例をサンプル 2 に示す．但し，このサンプルはまだ不完全なので、このサンプルを用いて回路を完成させる．

```
-- (サンプル２) 4 ビット並列加算器
library IEEE;
use IEEE.std_logic_1164.all;

entity FULLADDER4 is
  port( A0, A1, A2, A3 : in std_logic;
        B0, B1, B2, B3 : in std_logic;
        CI             : in std_logic;
        S0, S1, S2, S3 : out std_logic;
        CO             : out std_logic );
end FULLADDER4;

architecture STRUCTURE of FULLADDER4 is

component FULLADDER
  port( A, B, CI : in std_logic;
        S, CO    : out std_logic );
end component;

signal C1, C2, C3 : std_logic;

begin

  FA0 : FULLADDER port map ( A0, B0, CI, S0, C1 ); -- 0 ビット目の全加算器
  FA1 : ～     以降，1 ビット目から 3 ビット目までの全加算器について
  FA2 : ～     0 ビット目と同様に記述する．この際，隣の全加算器との
  FA3 : ～     入出力の接続に注意して記述する．

end STRUCTURE;
```

(i) 4 ビット並列加算器の設計手順

Quartus II の起動から DE0 ボードを用いた動作確認までの手順は前の全加算器の場合と同じであるが，異なる点を以下に示す.

【1】プロジェクト名、プログラムファイル名、およびプログラムでのエンティティ名は '**FULLADDER4** とする.

【2】プロジェクトは全加算器と同じフォルダに作成するが，このとき「Quartus II の使い方」の P6 で 「Next」をクリックした後に出てくるダイアログ画面では「いいえ」をクリックして次に進む.

【3】「Quartus II の使い方」の P6 で fulladder,vhd を追加する (右側の「Add All」ボタンをクリック)

【4】サンプル 2 の FA1, FA2, FA3 には FA0 と同じ書式の記述になるが，FA0 〜 FA3 は 実験書図 5 の F0 〜 FA3 に対応する．よって，桁上りの伝播などに留意して記述していく必要がある.

【5】論理シミュレーションの入力データには、2 通りのデータを用います。1 つは '9 + 5' で、もう 1 つは最後に桁上りが発生する (CO = '1' となる) 加算入力を使用する.

※ 4 ビット並列加算器は、担当の方に動作確認をしてもらう事.

5.4. 7 セグメント LED を用いた 2 進-10 進デコーダの設計

「2 進-10 進デコーダ回路」は、DE0 ボードにある 7 セグメント LED(1 つ) と 4 つのスライドスイッチを用い、スイッチの 4 桁 2 進数を 0〜9 までの 10 進数に変換して表示する回路である．

○設計上の注意

この課題は、全加算器と同じ手法で真理値表を用いて回路設計を行う．但し，DE0 ボードでは LED の **消灯/点灯** と信号値の対等が， 使う LED の種類によって異なる．全加算器、4 ビット並列加算器で使用した LED では「**消灯：信号'0'**、**点灯：信号 '1'**」であったが，7 セグメント LED では逆で「**消灯：信号 ' 1'**、**点灯：信号 '0'**」となっている．そこで、次の表 4 のような表現の真理値表を用います。

. . . . . .

(i) 真理値表

表 4 は、スライドスイッチが上の時を'**OFF**、下の時を'**ON**' と考え、7 セグメント LED が消灯の時を '**OFF**'、点灯の時を'**ON**' と考えて作成した真理値表である．

表 4: ON/OFF 表現による真理値表

スイッチ入力				7 セグメント **LED**						
I3	**I2**	**I1**	**I0**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
OFF	OFF	OFF	OFF	ON	ON	ON	ON	ON	ON	OFF
OFF	OFF	OFF	ON	OFF	ON	ON	OFF	OFF	OFF	OFF
OFF	OFF	ON	OFF	ON	ON	OFF	ON	ON	OFF	ON
OFF	OFF	ON	ON	ON	ON	ON	ON	OFF	OFF	ON
OFF	ON	OFF	OFF	OFF	ON	ON	OFF	OFF	ON	ON
OFF	ON	OFF	ON	ON	OFF	ON	ON	OFF	ON	ON
OFF	ON	ON	OFF	ON	OFF	ON	ON	ON	ON	ON
OFF	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF	OFF
ON	OFF	OFF	OFF	ON	ON	ON	ON	ON	ON	ON
ON	OFF	OFF	ON	ON	ON	ON	ON	OFF	ON	ON

(ii) 論理式の導出

次に、表 4 から論理式を作成する．

DE0 ボードは、前に述べたようにスイッチが上の時が信号**'1'**、下の時が信号'0'、7 セグメント LED は信号**'1'**が消灯、信号**'0'** の時が点灯なので、入力・出力とも '**OFF = '1'**、**ON = '0'** として論理式を導出していく．

この場合、例えば出力信号**'1'** の組み合せで論理式を導出すると '**a**' の論理式とその VHDL 表記は次のようになる．

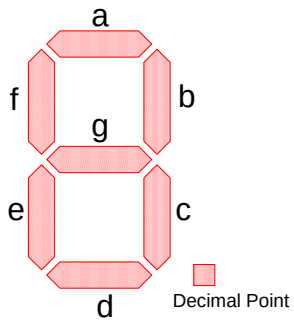
$a = \overline{I0} \cdot I1 \cdot I2 \cdot I3 + I0 \cdot I1 \cdot \overline{I2} \cdot I3$　　(←論理式)

a <= (**not** I0 **and** I1 **and** I2 **and** I3) **or** (I0 **and** I1 **and not** I2 **and** I3)　　(← VHDL 表記)

同様に、以下 '**b**' から '**g**' までの論理式を求めていく (必要に応じて論理式の簡単化も可)。

(iii) VHDL による 2 進-10 進デコーダ回路の設計

以下、Quartus II を使用して設計を行いますが、設計作業は**全加算器**の時と同じである．ピン配置は、表 5 を使用すること．



信号名	ピン番号	機能
I0	J6	スライドスイッチ SW0
I1	H5	スライドスイッチ SW1
I2	H6	スライドスイッチ SW2
I3	G4	スライドスイッチ SW3
a	E11	7 セグメント LED HEX0_D0
b	F11	7 セグメント LED HEX0_D1
c	H12	7 セグメント LED HEX0_D2
d	H13	7 セグメント LED HEX0_D3
e	G12	7 セグメント LED HEX0_D4
f	F12	7 セグメント LED HEX0_D5
g	F13	7 セグメント LED HEX0_D6

図 5: 7 セグメント LED を用いた 2 進-10 進デコーダのピン配置

(重要) 回路名 (プロジェクト名 = *entity* 名 = ソースファイル名) は、必ず半角英文字で始まる名前にすること．(**7segment** など、数字で始まる名前は不可)

(iv) 論理シミュレーション

真理値表の I0 ～ I3 を入力信号として論理シミュレーションを行う．

(v) UP-1 ボードでの動作確認

各入力値で適切なセグメント位置が点灯するようにすること．具体的には，例えば入力値が 10 進数 '1' の場合には，セグメント 'b' と 'c' が点灯するように設計されていれば良い．

※ 2 進-10 進デコーダ回路は、担当の方に動作確認をしてもらう事.

6. レポートについて

- (重要) 実験書 ページ ii の ~ 実験に関する注意事項~ にある 2.実験レポートについて【2】の表紙の記入項目は 絶対に 忘れない事．未記入項目がある場合、適正に評価されないことがある．

レポートの提出期限は実験終了日から 2 週間後まで。提出するレポートには，

○ 目的、実験装置
○ 実験方法：簡単でいいですが、実際に行なった実験作業について 流れがわかるように書く事．
○ 実験結果・考察：印刷したプログラム、シミュレーション結果を添付する．
○ 検討課題 (結果・考察・課題)：

を書いて提出する．なお、レポートに添付する実験結果 (印刷物) は

- 全加算器の Quartus II での論理シミュレーション結果
- 4 ビット並列加算器のプログラムファイル
- ~~4 ビット並列加算器の論理シミュレーション結果~~
- 7 セグメント LED のプログラムファイル
- 7 セグメント LED の論理シミュレーション結果

.....

~~(課題) シミュレーション結果では、設定した入力信号に対して出力結果がずれて (遅れて) 表示される．この入出力間の遅延の原因について考察する．~~

参考文献

[1] “ＶＨＬＤによるハードウェア設計入門”，長谷川裕恭，ＣＱ出版社

[2] “A VHDL Primer ＶＨＤＬ言語入門”，Jayaram Bhasker，ＣＱ出版社

[3] “ＦＰＧＡ活用チュートリアル”，ＣＱ出版社

[4] “ＦＰＧＡ／ＰＬＤ設計スタートアップ”，ＣＱ出版社