1999 年度　　卒業論文

ハードウェア記述言語を用いた PS/2 マウスのコントローラの設計

学籍番号　　：9610179
名前　　：ホー・フィ・クーン
電気通信大学　電気通信学部　電子工学科
木村・齋藤研究室
指導教官　齋藤　理一郎　助教授

提出日　　：平成 12 年 3 月 30 日

もくじ

もくじ

第1章　　序論

1.1 背景

最近、ディジタル回路の設計、開発にハードウェア記述言語（Hardware Description Language HDL）が広く使われている。ハードウェア記述言語にはいろいろな種類があるが、一般に使われているのは VerilogHDL と VHDL である。VHDL の V は(Very High Speed Integrated Circuit) プロジェクトの名前に由来している。当研究室では各前年度から、ウインドウ環境での VHDL を採用している。そして、設計手順も決まっている。

PeakVHDL&FPGA ソフトで VHDL ソースを編集、コンパイル、シミュレーションと設計回路を実際のデバイスで実装するための必要な情報の論理合成までの作業をする。

MAX+plusII ソフトは FPGA(Field Programmable Gate Array)に書き込むための情報の配置配線ツールとコンフィギュレーションツールで使う。

97 年度にグェンさんと松尾さんが行列専用計算機の設計手順を決め、松尾さんは FLEX10K シリーズの EP10K100GC503-4 の FPGA を 2 個と SRAM、DRAM からなる実験基板を作成した。そして、PC と基板のデータ通信のボードも作成した。この基板を使って、VHDL で設計した回路を実装する。PC から結果検証プログラムで基板にデータを送って、基板が計算した結果データを PC に表示する。このモデルは現在、本研究室の専用計算機の設計、開発環境に用いられる。

ここで、上のモデルに FPGA にダウンロードした後、PC に頼らず、自己入出力システムで計算結果を検証するための入出力システムを考えた。まず、マウスのコントローラシステムから設計する。今後、キーボード、VGA などに発展して行くことも考えられる。

マウスのコントローラシステム設計回路の検証のため、マウスコレクターと VGA コネクターが付いている Altera 社が提供した University Program Board (UP1 ボード) あるいは個別の市販 FPGA で、実験基板を作って、検証を行う。

1.2 研究目的

本研究の目的はPS/2マウスのコントローラシステムを設計する。PS/2マウスはマウスの通信基準に従って、システムと交信する。設計するコントローラシステムはマウスの通信アルゴリズムを理解し、マウスから、状態データ（動いたり、ボタンを押されたりなど）を送って来るようにコントロールする。

受け取ったマウスのデータを処理する。表示装置(LED, VGA など)で表示させたり、他の装置を動作させたりをすることを考える。

1.3 論文の構成

第1章：背景を研究の目的について

第2章：PS/2マウスの構造とPS/2マウスのシステムとの通信アルゴリズムについて説明する。

第3章：設計手順と利用するソフト（PeakVHDL&FPGA とMAX+PlusII）の使い方について、カウンタ機能回路の設計から、検証まで、一連操作を通して説明します。

第4章：設計するマウスコントローラシステムについて説明する。システムの主な機能はマウスとコミュニケーション（マウスのメッセジーを理解し、モードセット、データ伝送などの命令をする。）することと、マウスの状態データを処理すること。

第5章：付録に設計のプログラムを記載する。

第２章　マウスとシステムの交信アルゴリズム

2.1 PS/2 マウスの構造

図 2.1 は現在、普及に使われている PS/2 マウスの構造である。その詳細は 図 2.2 この回路構成図で見える。

図 2.1　PS/2 マウスの構造

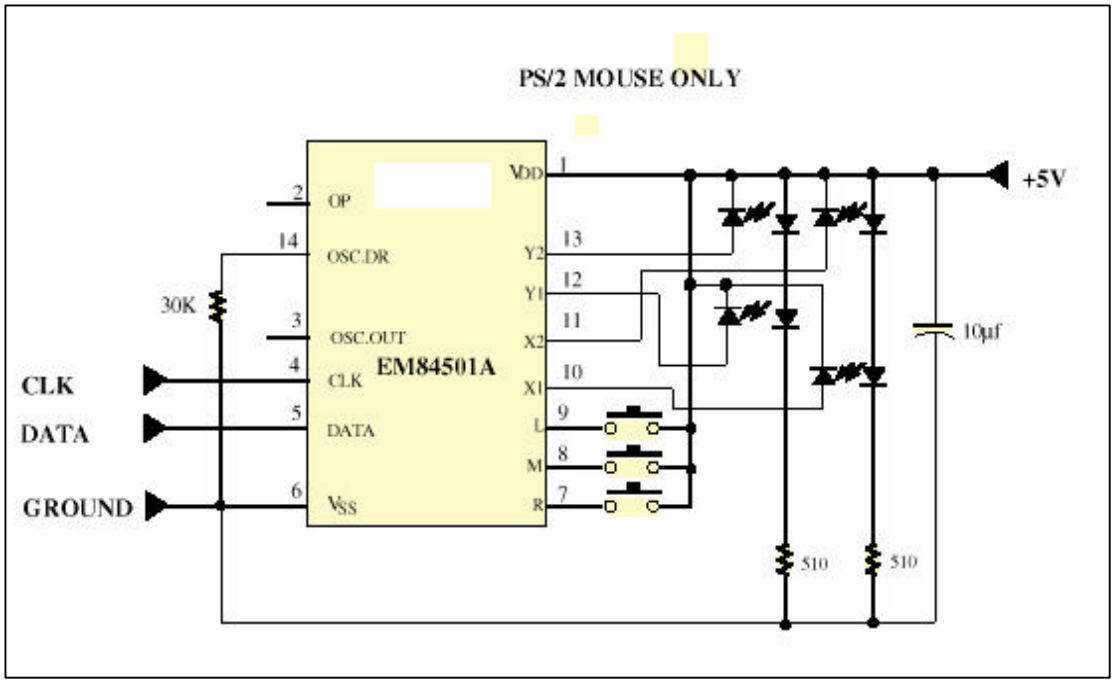


図 2.2　PS/2 マウスの回路構成

図 2.2 中の 10~13（12 と 13 は使用していない）のところはマウスが動くたびに、X と Y 方向のローラが回転する。ローラの円盤に刻んだ空き間を通る光センサーの動作で、回転に相応なディジタル信号を生じる（図 2.3)。

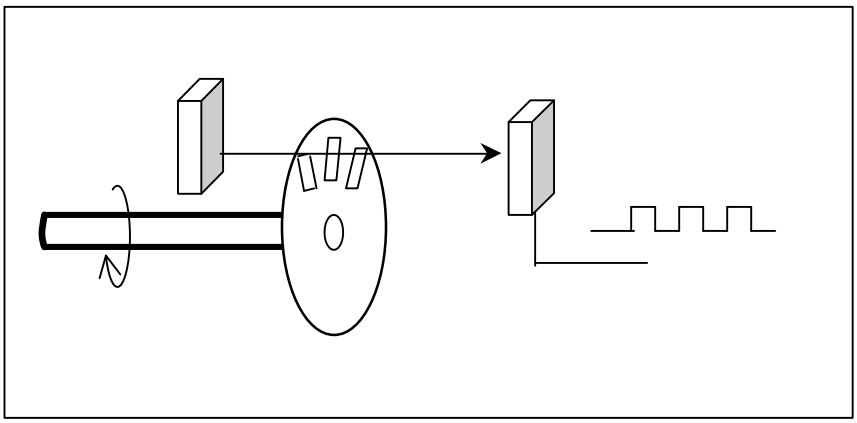


図 2.3　マウスの動き信号の仕組み

図 2.2 中の 7～9 はスイッチで、マウスのボタンと相応（真中のボタンを使用していない）する。CKL と DATA はシステムと通信ための信号である。

2.2 マウスの通信アルゴリズム

2.2.1 オペラショーンモード

マウスはシステムを通信するために 4 つのオペラショーンモードがある。

i) **Reset Mode:**
パワーが ON のとき、あるいは、リセットコマンドが来るときに自己テストのためのモードである。リセット信号を受けた後、マウスは
1）完成コード AA と ID コード 00 を送る（図 2.4)。
2）デフォルト状態をセット：
サンプル速度：100report/s
non-autospeed
stream mode

2dot/count
disable

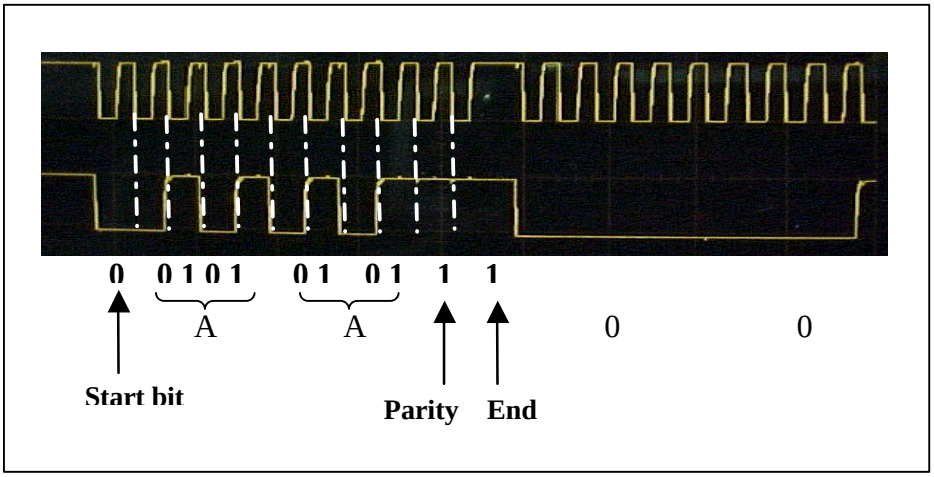


図 2.4　マウスからの AA00 信号

ii) Stream Mode:
以下の条件を満たすならデータを転送する。
1）マウスのスイッチが押される。
2）マウスが動く。

iii) Remote Mode:
リードデータ(read data)コマンドに返事するためのデータ転送。

iv) Wrap Mode:
Reset wrap mode (16 進数で EC)と Reset (16 進数で FF) 以外のすべてのシステムからのデータを送り返す。

2.2.2 PS/2 マウスデータレポット：

i) Stream mode のとき：
データレポートが各サンプルの間にサンプルの終わりのところから送られる。

ii) データ・レポート・フォマット：

表 2.1　データ・レポート・フォマット

Byte	Bit	Description
1	0	左ボタンの状態；　　1= 押す
	1	左ボタンの状態；　　1= 押す
	2	真中ボタンの状態；　　1= 押す
	3	予備 (reserve)
	4	X 方向のデータ符号；　1= negative
	5	Y 方向のデータ符号；　1= negative
	6	X データの overflow；　1= overflow
	7	Y データの overflow；　1= overflow
2	0~7	X データ(D0~D7)
3	0~7	Y データ(D0~D7)

2.2.3 PS/2 マウスデータトランスミーショーン：

i) データをマウスからシステムに送るときにも、システムからマウスに送るときにも、マウスのクロックを利用する。

ii) システムはマウスがデータを送ってもらいたいときに：
データラインを inactive レベル、クロックラインを active レベルにする。

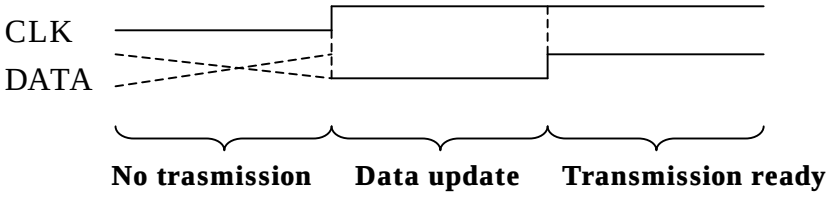
iii) **Data transmission Frame :**

表 2.2　Data transmission Frame

ビット	機能
1	スタートビット　（常に０）
2~9	データビット　（D0~D7）
10	パリティビット　（奇数パリティ）
11	終わりビット　（常に 1）

iv)　Data Output (マウスからシステムに)：

クロックが low **のとき（抑制状態）：データ転送をしない。**
クロックが high **とデータが** low **のとき：データをオップデート（**update**）する。**
クロックとデータが共に high **のとき：転送できる状態になる。**



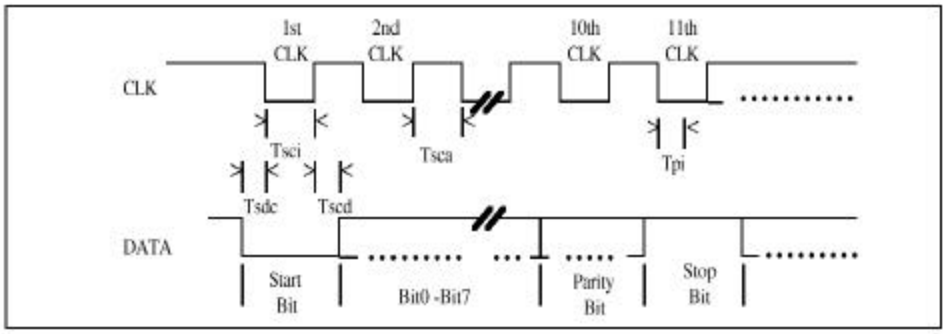
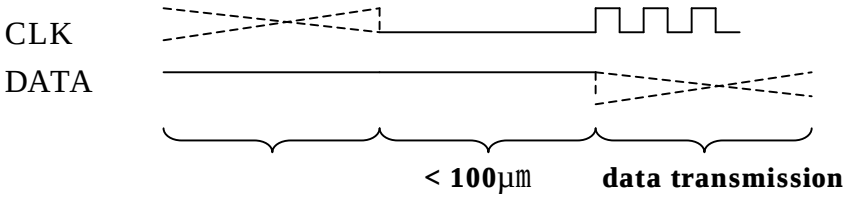


図 2.6　**データアウトプットの波形**

v)　Data Input (システムからマウスに)：

まず、システムはマウスからデータが送られているかどうかをチェックする。もし、送り中と 10 **クロック目の前なら、**override **でクロックを** inactive **にすることができる。**10 **クロック目以後なら、データを受け取る。**
もし、マウスがデータを送っていない、あるいはシステムが override **を選ぶなら、システムからデータを送る準備のためにシステムはクロックを** 100μm **以下に** inactive **にする。**
システムがスタートビット(0**)を送るとマウスはクロックを** active **にしてから、**11 **クロックを出す。マウスはデータラインの** 11 **ビット目をチェックして、もし、**active **ならデータを** low **にして、もう** 1 **回、クロックを出す。**
もし、マウスはエラーを見つかったら、データが high **まで、クロックしてから、**resend **を要求する。**

システムはコマンドあるいはデータを送った後、マウスの返事を要求する。返事を来るまで、次のものを送らない。



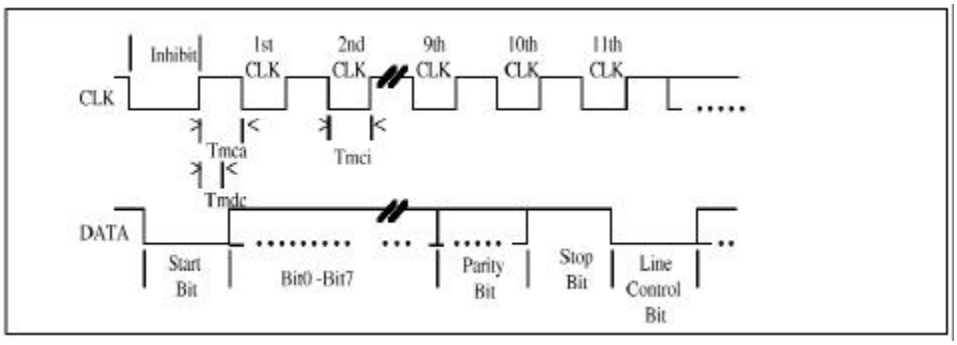


図 2.6　データインプットの波形

2.2.4 PS/2 マウスのエラーハンドリング

i)　無効なインプットあるいは、不正確なパリティをもらうと Resend (FE) コマンドを送る。

ii)　もし、無効な 2 つインプットが連続に来ると、エラーコード(FC)送る。

iii)　Resend コマンド以外の　　コマンドを来ると、保留されるカウンタをすべて消される。

iv)　マウスが Resend (FE) コマンドをもらうと、データの最後のパッケトを送る。

v)　Stream mode のとき、マウスはシステムに 3 バイトのデータ・パケットを送った後、Resend (FE) コマンドをもらうとカウンタを消す前に、3 バイトのデータ・パケットを再送する。

vi)　もし、システムが返事を要求するとトランスミーショーンにエラーを発見すれば、返事は 25ms 内に送られる。

2.2.5 PS/2 マウスコマンド：

システムとマウスの交信のために 16 コマンドがある。
英文字は 16 進数のコード、XX はマウスからのコマンドあるいはマウス状態のデータである。

表 2.3　PS/2 マウスコマンド

Hex code	Command	Mouse echo code
FF	Reset	FA,AA,00
FE	Resent	XX,(XX,XX)
F6	Set Default	FA
F5	Disable	FA
F4	Enable	FA
F3,XX	Set Sampling Rate	FA,FA
F2	Read Device Type	FA,00
F0	Set Remote Mode	FA
EE	Set Wrap Mode	FA
EC	Reset Wrap Mode	FA
EB	Read Data	FA,XX,XX,XX
EA	Set Stream Mode	FA
E9	Status Request	FA,XX,XX,XX
E8,XX	Set Resolution	FA,FA
E7	Set Autospeed	FA
E6	Reset Autospeed	FA

1)　Reset (FF) :
　　このコマンドを受けるとマウスは
　I)　　リッセトをする。
　II)　　FA,AA,00 をシステムに送る。
　III)　デフォルト状態をセット：
　　サンプル速度：100report/s
　　non-autospeed
　　stream mode
　　2dot/count
　　disable

2)　Resend(FE) :
　I)　　マウスは無意味なコマンドを受けるときに Resend コマンドをシス

テムに送る。

II) マウスは Resend コマンドを受けると、データの最後のパケットを再転送する。もし、最後のパケットは Resend コマンドだったら、Resend コマンドのすぐ前のパケットを再転送する。

III) Stream mode のとき、もしマウスは Resend コマンドを受ければ、3 バイトのデータをシステムに送る。

3) Set Default (F6) :

どんなコンディションでも、最初のデフォルトの状態にセットする。

4) Disable (F5) :

Stream mode のとき、このコマンドで、データ転送を中止する。

5) Enable (F4) :

Stream mode のとき、このコマンドでデータ転送が開始する。

6) Set Sampling Rate (F3,XX) :

Stream mode のとき、このコマンドの 2 バイト目の XX の値でサンプル速度をセットする。

Second byte XX	Sample Rate
0A	10/sec
14	20/sec
28	40/sec
3C	60/sec
50	80/sec
64	100/sec
C8	200/sec

7) Read Device Type (F2) :

マウスがはこのコマンドを受けると、いつも FA,00 をシステムに返す。

8) Set Remote Mode (F0) :

Read Data コマンドに返事するときだけ、データを送る。

9) Set Wrap Mode (F2) :

Reset(FF)あるいは Reset Wrap Mode(EC)コマンドが来るまで、Wrap mode が存続する。

10) Reset Wrap Mode (EC) :

このコマンドを受けると、現在モードのすぐ前のモードに戻る。

11) Read Data (EB) :

Remote mode と Stream mode のとき、このコマンドが実行される。マウスが移動しなくても、マウスのボタン状態が変わらなくても、最後のレポートが転送される。Read data コマンドを実行した後、マウスのレジスターに保留されたデータを全部消される。

12) Set Stream Mode (EA) :

Stream Mode をセットする。

13) Status Request (E9) :

このコマンドを受けると、マウスは 3 バイトの状態データをシステムに転送する。データフォーマットは表 2.1 に参考。

14) Set Resolution (E8,XX) :

2 バイト目の XX の値で分解度をセットする。

Second Byte XX	Resolution
00	8 dot/count
01	4 dot/count
02	2 dot/count
03	1 dot/count

15) Set Autospeed (E7) :

Stream mode のとき、サンプルが終わりのところ、X と Y データの値が更新される。Input と Output 間の関連カウント（count）は以下のようにある。

Input	Output
0	0
1	1
2	1
3	3
4	6

5	9
N(>=6)	2*N

16) Reset Autospeed (E6) :

普通のスピードに戻す。

第3章　設計方法

3.1 設計の概要

本研究では、ハードウェア記述言語 VHDL を使って、マウスコントローラの設計を行う。また、プログラムマブルデバイスである FPGA (Field Programmable Gate Array) に設計したコントローラを実装し、実際の動作を検証する。

VHDL はハードウェア記述言語 (Hardware Description Language, HDL) の 1 つである。ハードウェア記述言語は設計したい回路を " 文書 " で記述し、後はコンパイラに自動変換させるという点では C や Pascal などのソフト的な言語と同様であって、容易に記述できる。

ハードウェア記述言語はいろいろな種類があるが、一般に使われているのは VerilogHDL と VHDL である。VHDL の V は(Very High Speed Integrated Circuit) プロジェクトの名前に由来している。当研究室では VHDL を採用している。

FPGA(Field Programmable Gate Array)は書き込み可能な集積回路で、その中に、フリップフロップ、and、or、not、加算器等の基本的な回路素子が数百から、数万個入っている。これらの回路素子の相互接続情報は内部 RAM に保存されている。外部から適当な電気信号を加え、RAM の内容を書き換えることによって簡単に FPGA の機能を変えることができる（外部から信号を加え、FPGA の機能を設定することはコンフィギュレーションという）。FPGA は基盤にあったままで、何回も書き換えることができるという特徴があり、設計した回路を検証するには最適である。

3.2 設計の手順

HDL を LSI 設計の流れを図 3.1 に示す。

まず、HDL により、回路の機能を記述した HDL ファイルを作成する。そして、機能検証（シミュレーション）を行い、機能を正しくなければ、再び HDL ファイルを作成し直す段階へ戻り、同様の手順に従う。機能が正しければ、論理合成を行うことにより HDL ファイルを EDIF(Electronic Design Interchange Format)ファイルに変換する。EDIF ファイルとはデジタル回路を表すフォーマットであり、回路の実際のデバイスで実装するための情報が含まれている。次に、この EDIF ファイルから TTF（Tabular Text File）形式で、表されたデバイスへ

の配置配線ファイルを生成し、この配置配線ファイルを FPGA へダウロードする。最後に機能を実装した FPGA により実装検証をし、正しく動作しなければ、再び、HDL ファイルを作成する段階に戻る。このようにして、実際の機能動作が正当なものとなるまで繰り返し設計を行うことになる。

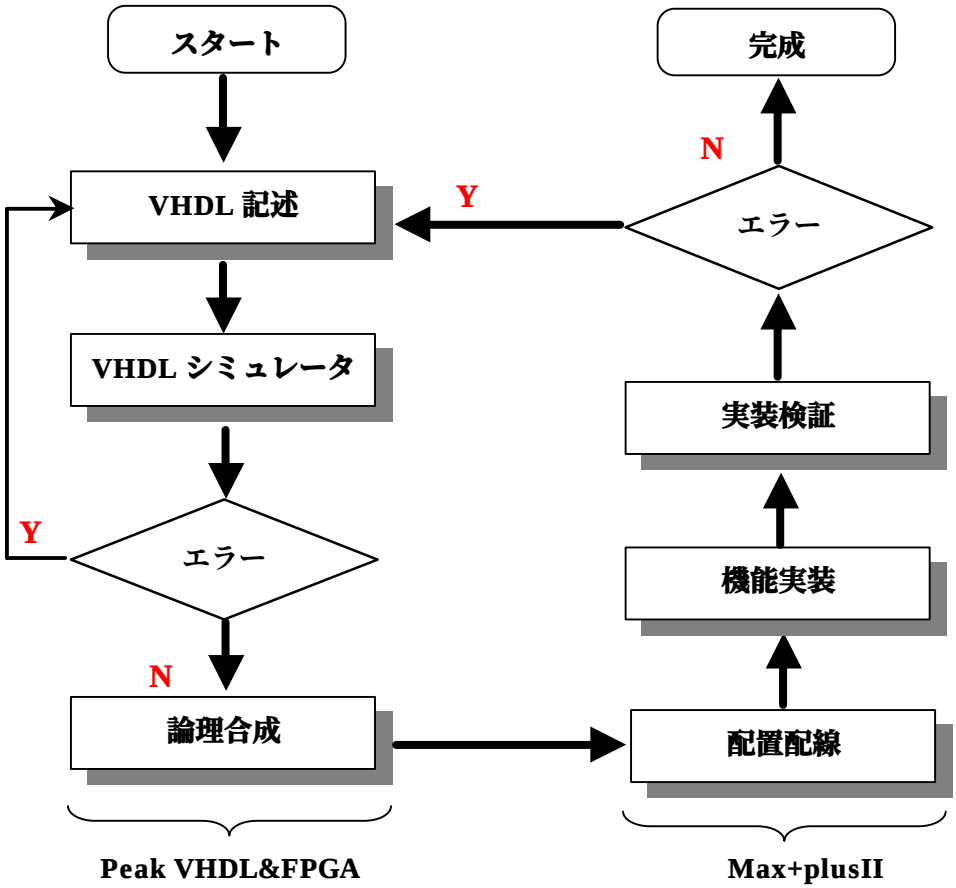


図　3.1 LSI 設計の流れ図

これから、本研究で Altera 社が提供した University Program Board (UP1 ボード)の FPGA で、カウンタの回路を設計する一連の操作を通して、設計の手順を説明する。

*** *ファイルは Wind¥c:¥ho¥fpga¥up1¥count3¥ ディレクトリあるいは UNIX の ~cuong¥fpga¥up1¥count3¥ でにある* ***

3.3 ソフトの使用方法

3.3.1 PeakVHDL&FPGA の使用方法

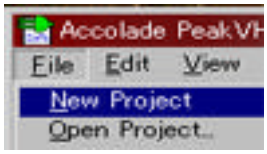
PeakVHDL&FPGA はアメリカ Accolade Design Automation 社が発売した VHDL 処理システムで、主に VHDL コンパイラと論理合成ツールから構成されている。PeakVHDL&FPGA の主な機能は以下に列挙する。

- 設計モデルの階層構造を管理する部分。
- VHDL ソースを編集するためのテキストエディタ。
- VHDL ソースを構文解説しコンパイルするコンパイラ、コンパイラはシミュレーションに必要な情報を生成する。
- 設計した VHDL モデルの動作を検証するシミュレータ。

PeakFPGA は PeakVHDL で記述した VHDL ファイルを実際のディジタル回路に変換するツールである。

3.3.1.1 VHDL ソースの作成手順

- 新しいプロジェクトを作る
  メニューバーから **File** => **New Project** を選択すると、新しいプロジェクトのウインドウが開く。



- プロジェクトを保存する
  新しいプロジェクトを任意のディレクトリに保存する必要がある。メニューバーから **File** => **Save Project As** を選択して、プロジェクト名とディレクトリを入力してから保存する。
- プロジェクトに VHDL ソースを追加する
  メニューバーから **File** =>**New Module** を選択すると図 3.2 のようなダイアログが現れる。このダイアログには 3 つのオプションがある。
  1. **Module Wizard**
     VHDL ソースにある決まった部分を含んだテプレットを自動的に生成してくれる。設計者は必要な部分を追加して、VHDL ソースを完成する。

2. **Test Bench Wizard**

　　上のオプションと同じであるが、テストベンチを VHDL で記述する場合に使う。

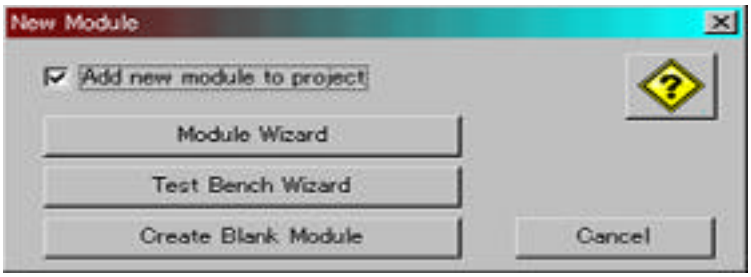


図 3.2　New Module ウィンドウ

3. **Create Blank Wizard**

　　単にテキストエディタを起動する。設計者が VHDL ソースを全部入力する。

- VHDL ソースの作成

  Create Blank Wizard をクリックして、ソースファイル名と保存するディレクトリを入力すると、VHDL ソースファイルのエディタウィンドウが開くと同時に、図 3.4 のように、VHDL ソースファイルがプロジェクトに追加される。

  VHDL ソースファイルのエディタで、VHDL ソースファイルを作成する。

***　*付録の 6.1.2 の count3.HDL ファイルを参考*　***

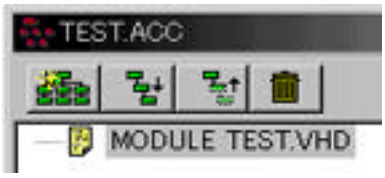


図 3.4　VHDL ソースファイルを追加したプロジェクトウィンドウ

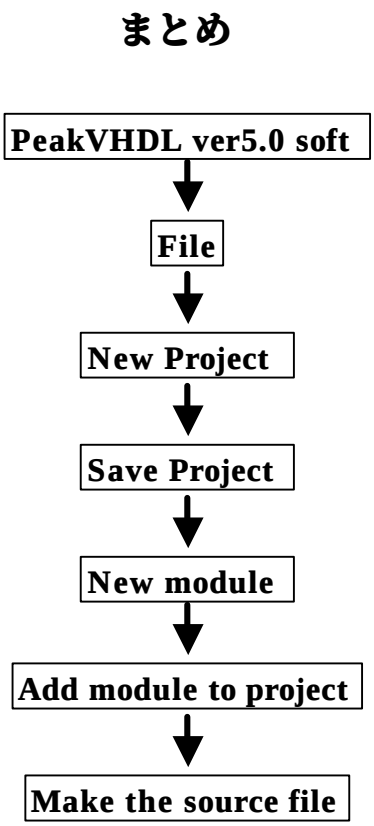


図 3.5　VHDL ソースの作成手順のまとめ

3.3.1.2 VHDL ソースのコンパイル

PeakVHDL&FPGA のコンパイルは VHDL ソースの論理合成を行わず、ソースの構文解析とシミュレータに必要な情報を生成するだけである。

- まず、図 3.5 のように、プロジェクトウインドウのツールバーの上に[プロジェクト再構築]、[プロジェクトの階層構造表示]、[プロジェクトの階層構造隠し]などのボタンをクリックして、[MODULE ... VHD] を選択される状態にする。（VHDL を用いた設計は階層構造である。従って、コンパイルに必要な情報を持つところを選択する必要がある）

図 3.6　コンパイルスタンバイ

- 次に、ツールバーのコンパイルボタン 

をクリックすると、コンパイルが始まる。同時に、メッセージを表示する Transcript の画面が出る。

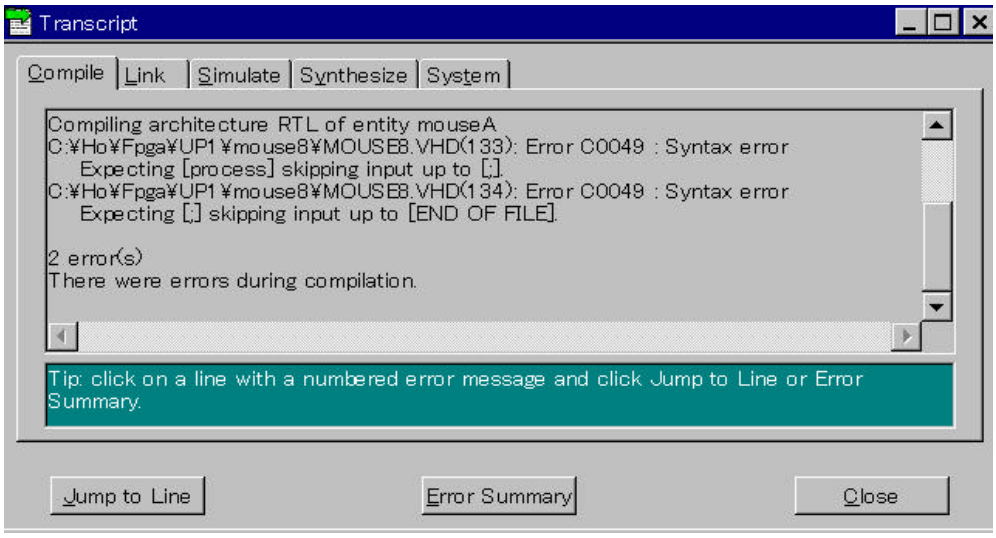


図 3.7　Transcript ウィンドウ

Transcript 画面の中に、コンパイルの進行状況や発見されたエラーなどの情報が出力される。図 3.6 には 2 つのエラーメッセジーがあった。(　)括弧の中の数字はソースファイルのエラーの行数である。エラーメッセジーのところに、カーソルを置いて **Jump to line** のボタンを押すと、ソースファイルのエラーのところにジャンプする。 **Jump to line** のボタンを押すと、エラー解決のアドバイスが表示される。

3.3.1.3 PeakVHDL のシミュレーション

シミュレーションは設計した回路が、正しく動作するかどうか確認するための課程である。実際、回路の適当な入力（テストベクトル）を与え、出力される値を予想値と比較する。

1. **テストベンチ：**

   テストベンチとは入出力ポートを持たない特別な VHDL モデルであり、シミュレーションの必要な入力を生成する。

   テストベンチファイルを作成するにはまず、プロジェクトウインドウから、VHDL ソースファイル(file name.VHD)を選択して、それを、テ

ストベンチに組み込むモジュールとして、指定する。
次に、メニューから **File** ＝＞ **New module ...** で New module ウインドウを開く。New module ウインドウの **Test Bench Wizard** をクリックすると、
図 3.8 のように、テストベンチの入力画面が出る。

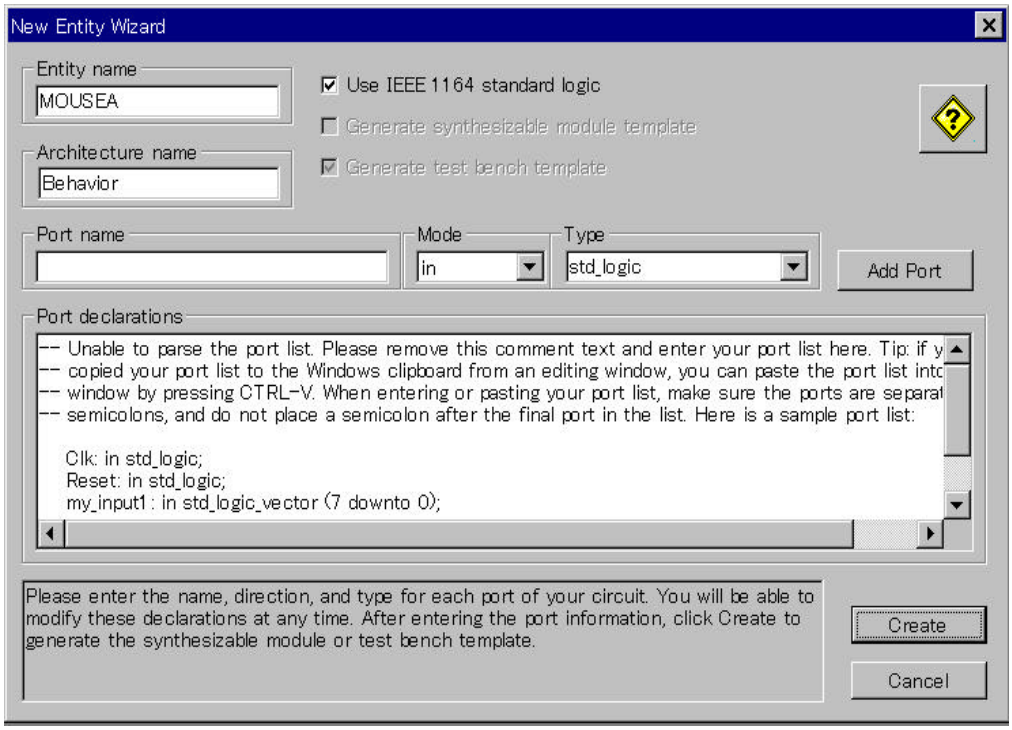


図 3.8　テストベンチ入力画面

テストベンチ入力画面の中に:

- Entity name :　先、選択された VHDL ソースファイルのファイル名を自動的にここにコピーする。
- Architecture name：デフォルト
- Port name , Mode , Type :　テストベンチのものではなく、シミュレーションしたいモジュール（テストベンチに結合されたモジュール）のものである。モジュールの同じものを入力する。
- Port declaration :　モジュールの同じものを入力する。

入力した後、**Create** をクリックすると、テストベンチのエディターウインドウが開く。適当な内容を編集する。
注：　テストベンチ入力画面の各項目は後のエディターウインドウで

入力することもできる。

***　　*付録のテストベンチのファイル TEST_COUNT3.VHD を参考*　　***

2. **シミュレーション：**

テストベンチのファイルが出来上がったあら、コンパイルをする。完成すると、シミュレーションの段階に入る。

- まず、プロジェクトウインドウの中のテストベンチファイルを選択された状態にし、ツールバーのコンパイルボタン  をクリックして、シミュレータが起動する。すると、コンパイルの時のように、進行状態のメッセージを表示する Transcript の画面が出る。エラーがないと、シミュレータは変数、信号など必要な情報を読み込むために、図 3.9 のように、シミュレーション用のオブジェクト選択画面が出る。シミュレーションで、見たいものを選んで、**Close** を押す。

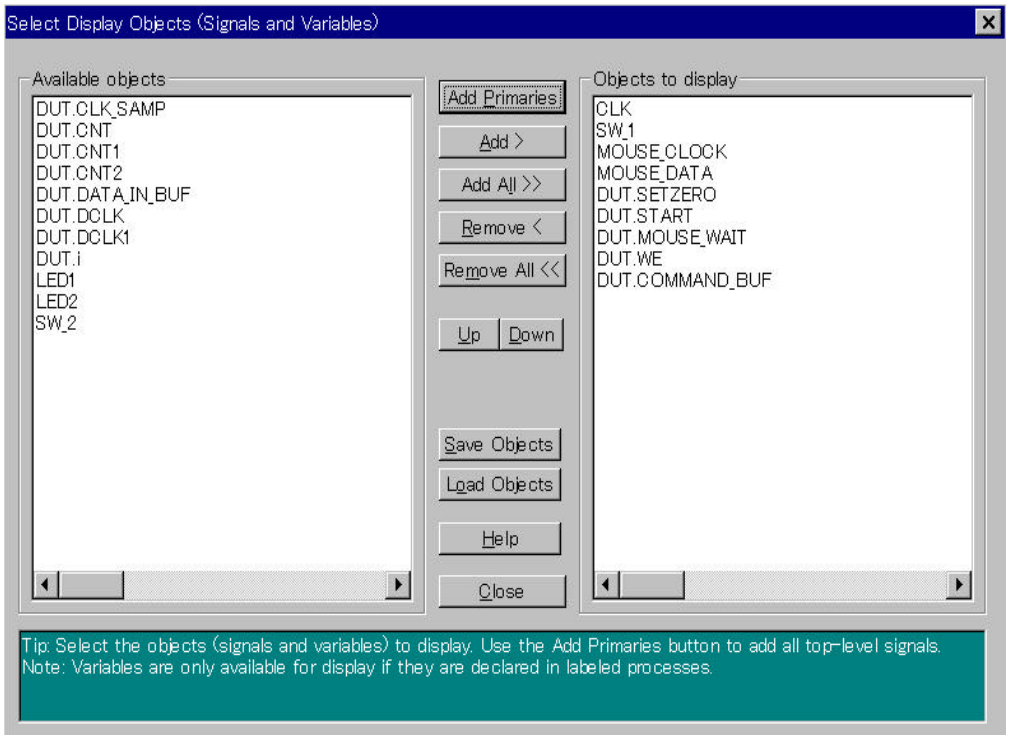


図 3.9　オブジェクト選択画面

- オブジェクトの選択が終わったら、図 3.10 のように、シミュレーション画面が出る。シミュレーションウインドウのツールバーから、時間設定のボタン  をスリックして、図 3.11 のダイアローグで、シミュレーションしたい時間を設定する。（**注：設定時間が長くなると、シミュレーション結果が出るまで、計算時間がかかるから、最低必要の時間だけを設定する**）。

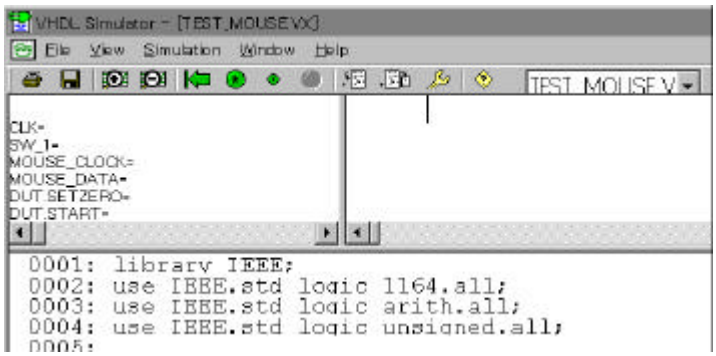


図 3.10　シミュレーションウインドウ

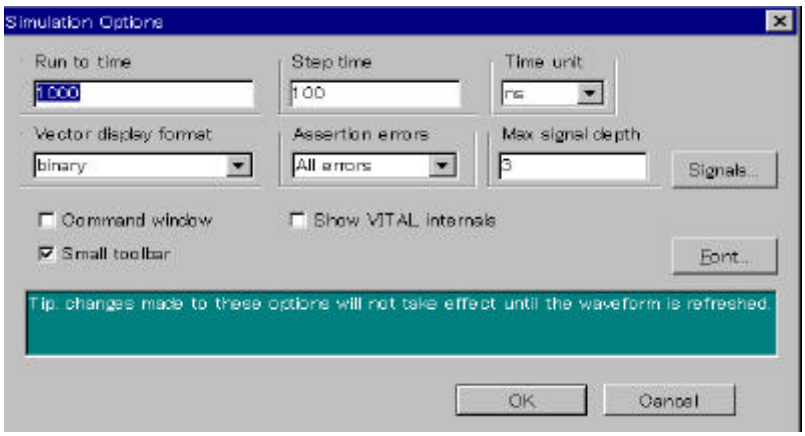


図 3.11　シミュレーション時間設定ダイアログ

- 次にシミュレーションウインドウのツールバーから、実行ボタン  を押して、実行させる。しばらく、シミュレータを計算した後、図 3.12 のように、シミュレーション結果の画面が出る。
  シミュレーション結果はディジタル波形のタームチャート、オブジェクトのカーソルところの値などが表示する。

オブジェクトを追加したいときには追加ボタン  を押して、図 3.9 のオブジェクト選択画面に戻れる。

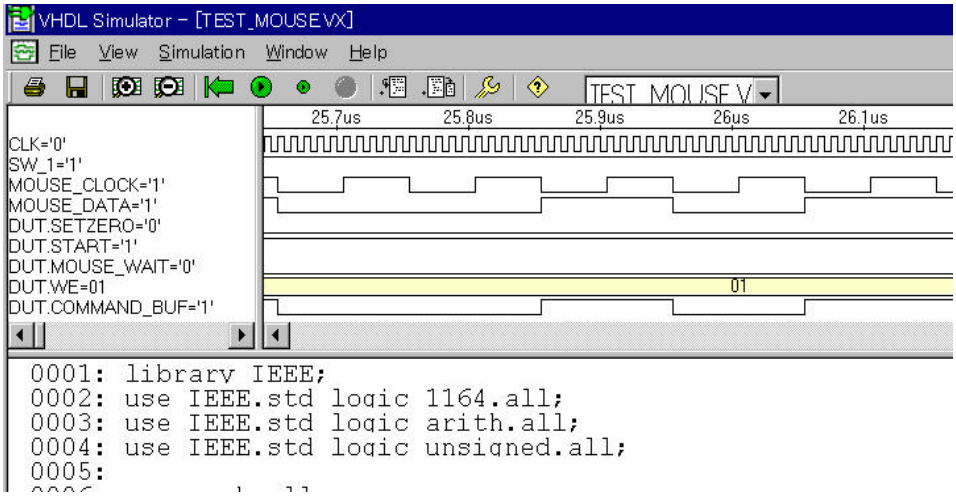
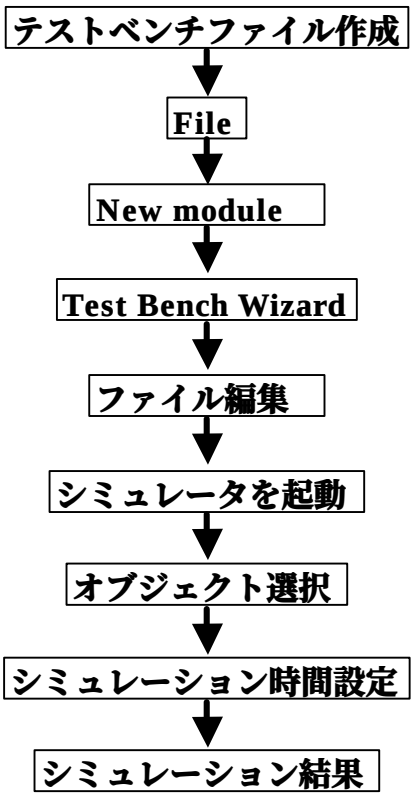


図 3.12　シミュレーション結果のウインドウ

シミュレーションのまとめ



3.3.1.4 論理合成

論理合成（synthesize）はハードウェア記述言語で記述したソースを実際のディジタル回路に変換する過程である。VHDL は wait 文や float 型などのように、本質的にシミュレーション向けで、論理合成不可能、あるいは合成できても効率が非常に悪く、実用性がない部分がたくさん持っている。一方、論理合成はまだ開発途上の技術であるため、このような構文がない VHDL コンパイラを通って、かつ論理レベルでのシミュレーションができても、論理合成できない VHDL ソースが多くある。

PeakVHDL&FPGA の論理合成ツールは VHDL ソースの変換結果として、EDIF（Electronic Design Interchange Format）ファイルを出力する。EDIF はディジタル回路を表す業界標準のファイル形式で、EDIF ファイルの中に、使用する基本部分（セル）の定義とこれらのセルの相互接続関係がテキストフォーマットで書かれている。どのようなセルが使えるかは、EDIF ファイルを処理するツールに依存するので、あらかじめそれを指定しなければならない。

使用するセルのセットを指定するには **Synthesize** ＝＞ **Options** を選択する。図 3.13 のようなオプション設定ウィンドウが表示される。

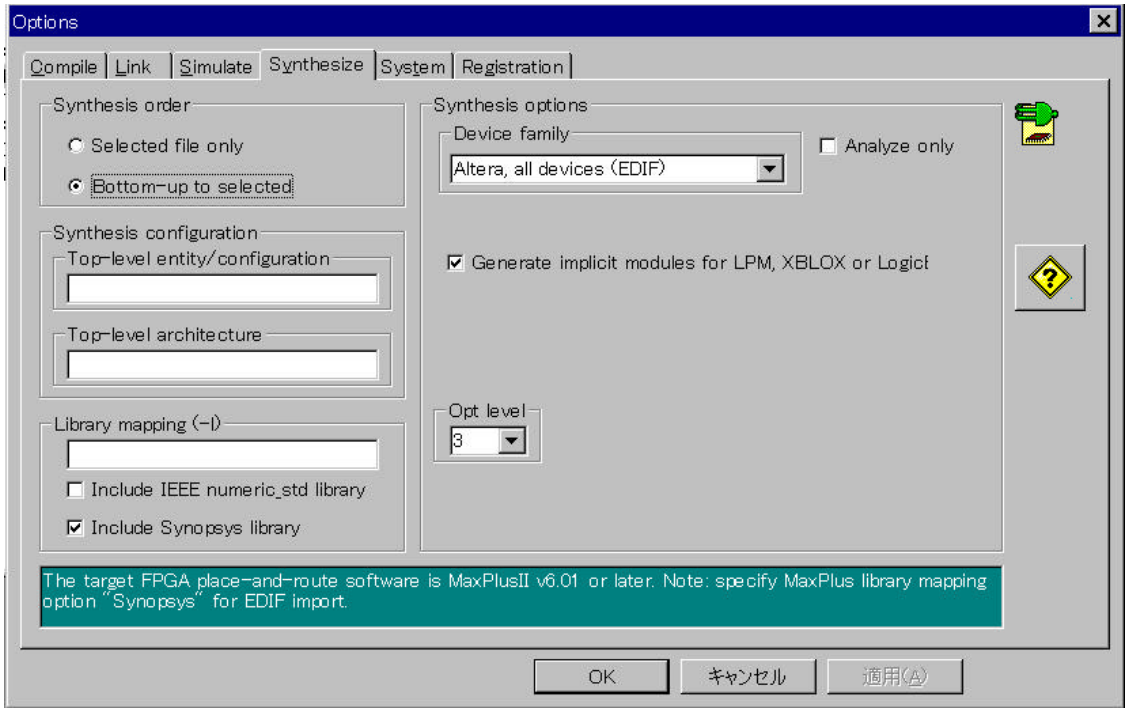


図 3.12　論理合成のオプション設定ウィンドウ

以下はよく使うオプションである：

- Library Mapping VHDL : ソースの中に算術計算などのライブラリによりサポートされる記述があれば、対応するライブラリをチェックする。
- Device Family : このオプションでセルセットを指定する。MAX+plusII で、EDIF ファイルを処理させる場合は *Altera,all devices (EDIF)* を選ぶ。
- Analyze only : 論理合成ツールはVHDL ソースを実際のディジタル回路に変換する前に、そのソースを構文解析し、ソースが変換可能かどうかを確かめる。普通このプロセスに比べて、実際の変換課程の方が多くの時間がかかる。ソースの構文をチェックする、あるいは単にソースが論理合成可能かどうかを知りたい場合はこのオプションをチェックする。

普段、Device Family だけを *Altera,all devices (EDIF)* を指定して、他のオプションはデフォルト値を使う。

Ok ボタンでオプションウィンドウを閉じ、ツールバーの  ボタンで、論理合成を開始する。進行状態やエラーメッセージなどは、VHDL ファイルを同じディレクトリに Metamor.log という名前で保存される。このファイルは **Synthesize** => **View Synthesis Log** で見ることができる。

3.6.2 MAX+plusII の使用方法

MAX+plusII は Altera 社の FPGA をターゲットとしたディジタル回路の設計を支援するソフトウェアである。MAX+plusII は主に以下の機能を持っている。

- 配置配線ツール
  ディジタル回路を表す EDIF ファイルから Altera 社の FPGA 用書き込み情報を生成する。
- 回路図入力ツール
  マウスで回路をグラフィカルに入力できる。入力したい回路図は VerilogHDL あるいは VHDL ソースに自動的に変換できる。
- デバイス（FPGA）にダウンロードする。

本研究では上で説明した PeakVHDL ソフトで作られた論理合成（EDIF ファイル）から TTF（Tabular Text File）形式で、表されたデバイスへの配置配線ファイルを生成し、この配置配線ファイルを FPGA へダウロードする作業を通して、MAX+plusII の使い方を説明する。

3.6.2.1 配置配線ファイルを生成

前節で述べたように、PeakVHDL ソフトで作られた EDIF ファイルは回路を実際のデバイスで実装するために必要な情報がすべてのを含まれている。また、EDIF ファイルから、FPGA に書き込むための情報（TTF ファイル）は必要である。そのため、TTF ファイルを生成する段階をしなくては行かない。

- まず、MAX+plusII ソフトを起動する。メニューバーから **File** => **File** を選択すると、図 3.14 のウインドウが開く。EIDF ファイルを選択する。次に、メニューバーから **File** => **Project** => **Set Project to Current file** をする。複数の EIDF ファイルが同時に開くことが可能から、この操作が必要である。

  **** Wind¥c:¥ho¥fpga¥up1¥count3¥ ディレクトリあるいは UNIX の ~cuong¥fpga¥up1¥count3¥ で COUNT3.EDF ファイルを選択する ****

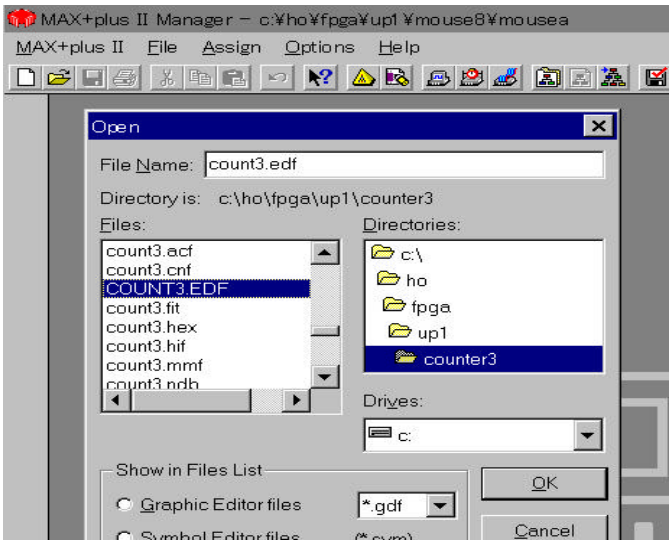


図 3.14　EDIF ファイルの選択ウィンドウ

- メニューバーから **Assign** => **Device** クリックする。図 3.15 のようにデバイス選択の画面が出る。ダウンロードしたい FPGA デバイスの番号と同じものをこのウィンドウから選択する。

  ****　UP1 ボード上の FPGA のデバイスは FLEX10K の EPF10K20RC240-4 であるから、その番号を選ぶ　****

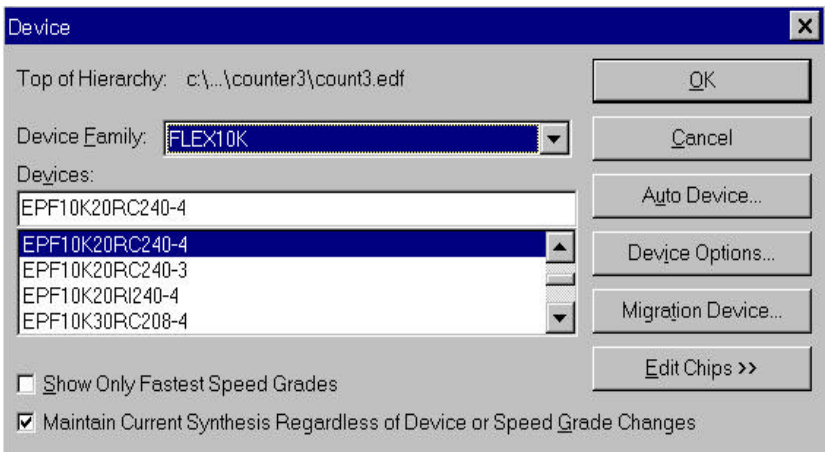


図 3.15 デバイス選択ウィンドウ

- メニューバーから **MAX+plusII** => **Compiler** をクリックする。図 3.16 のようにコンパイル画面が出る。**Start** ボタンでコンパイルを開始する。

コンパイルが成功したら、同時に TTF ファイルも生成された。ここで、TTF ファイルを生成する段階は終了である。

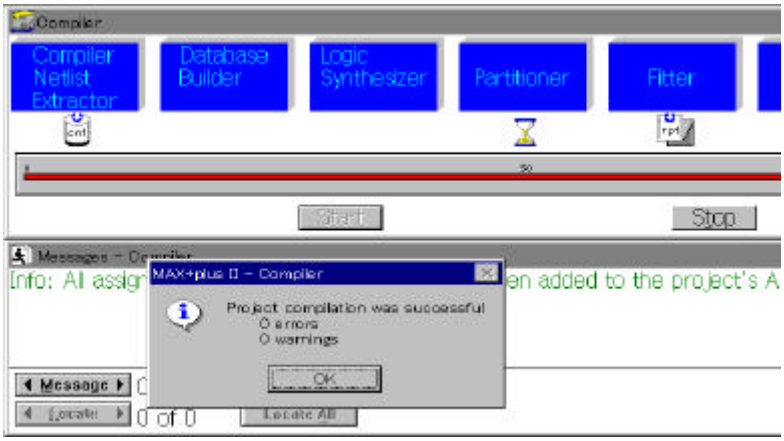


図 3.16　コンパイルウィンドウ

3.6.2.2 FPGA にダウロード

この段階は回路を実際のデバイスで実装するため、配置配線（TTF）ファイルを FPGA にダウンロードすること。

- メニューバーから **MAX+plusII** => **Programmer** をクリックする。図 3.17 のようにプログラマ画面が出る。

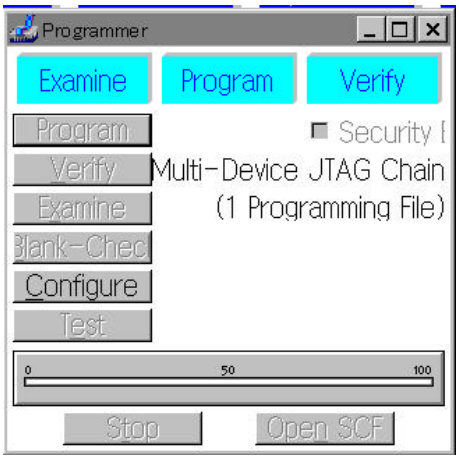


図 3.17　プログラマウィンドウ

- メニューバーから **JTAG** => **Multi – Device JTAG Chain Setup ...** をクリックして、図 3.18 のウインドウが開く。ここで、適当なダンロ

ードファイルを選択しなければならない。図 3.18 の中に「Programming File Names:」のところはこれからダンロードファイル名を表示する。当たらなかったら **Delete** で消す。
Select Programming File ... のボタンを押すと、ファイル選択のウインドが開く、このウィンドからダンロードしたいファイルを選択して、**Add** のボタンを押す。次に **Detect JTAG Chain Info** を押して、デバイスボードの電源とケッブルの接続状態をチェックする。異常がなかったら **OK** を押す。

**** Wind¥c:¥ho¥fpga¥up1¥count3¥ ディレクトリあるいは UNIX の ~cuong¥fpga¥up1¥count3¥ で count3.sof ファイルを選択する ****

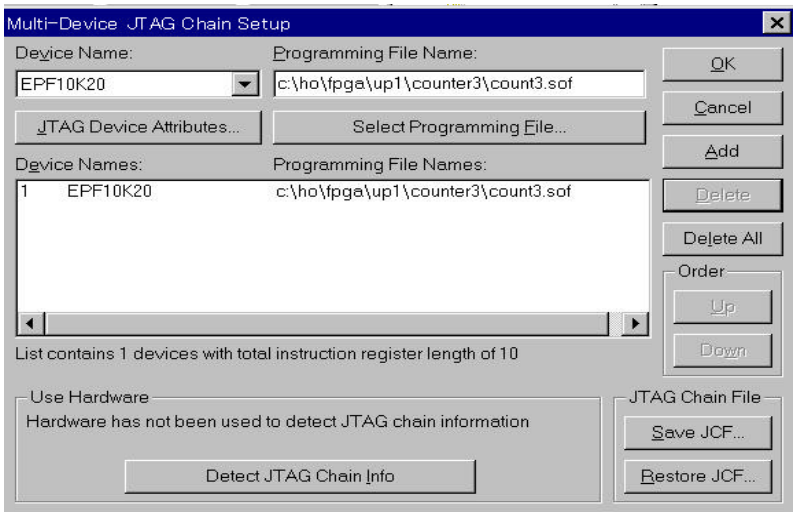


図 3.18 ダウンロードファイル選択ウインドウ

- プログラマウィンドウ（図 3.17）の **Configure** を押して、ダウロードを開始する。

この段階が終了したら、設計したい回路は次実際のデバイスに実装ができた。そのあと、動作のテストをして、検証を行う。

第 4 章　　マウスのコントローラシステムの設計

4.1　マウスとの通信

マウスはシステムと通信するために 4 つのモードと 16 のコマンド（表 2.3）が用意してある。コマンドは 11 ビットの単位で、1 パケットに分けられる。そのフォーマットは表 4.1 と図 4.1 のようにある。

表 4.1　　Data transmission Frame

ビット	機能
1	スタートビット　（常に 0）
2~9	データビット　（D0~D7）
10	パリティビット　（奇数パリティ）
11	終わりビット　（常に 1）

- 最初、マウスの電源が ON にすると、マウスは自動的にデフォルト状態をセットと図 4.1 のようにコード AA00 を送ったあと（2.2.1 オペラションモード 2.2.1 の節を参考）、システムからの返事を待つ状態になる。

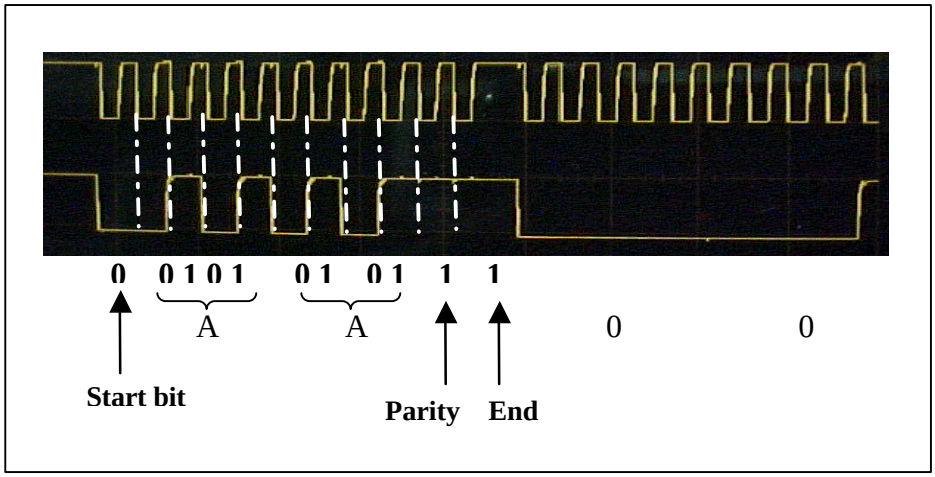
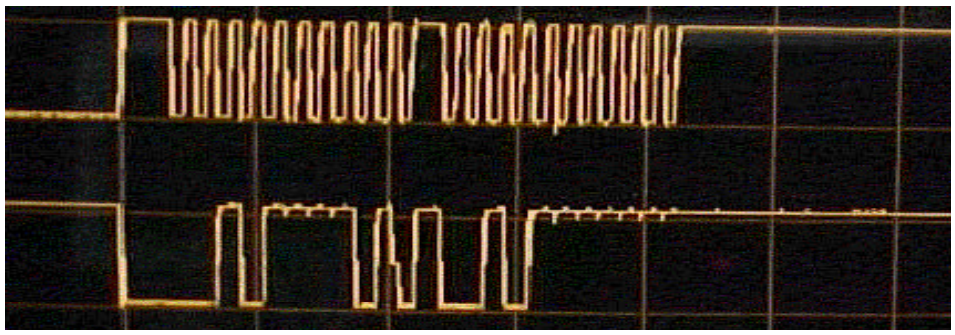


図 4.1　マウスからの AA00 の信号

- デフォルトの状態で Stream mode をセットされてある。コントローラシステムは データ伝送を命令するために enable Command (F4) 図 4.2）をマウスに送ると、そのあと、マウスが動いたり、ボタンを押されたりすれば、3 つパケットずつのデータをシステムに転送する 図 4.3）。

注意：マウスからシステムに転送するときにも、システムからマウスにコマンドを送るときにも信号はマウスのクロックと同期しなくてはいけない。ただし、マウスからシステムに転送するときに Rissing Clock と、システムからマウスにコマンドを送るときに Falling Clock と同期する。



4　　F　　A　　F

図 4.2　システムの enable Command (F4)とマウスの返事（FA）

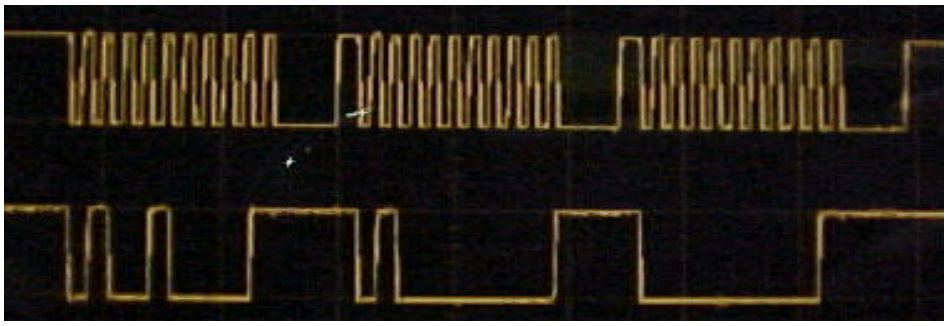


図 4.3　マウスの 3 パケットデータ（ビットの意味は表 2.1 に参照）

4.2　　システムの設計

- **まず、マウスのクロックと、同期の信号を生成する。そのため、データ信号の制御で、マウスにクロックを発生させなくてはいけない。その制御は** DATA **信号を** 100μm **以内 high レベル（システムの発信機を利用）にしてから、**Low **レベルに落とすと、マウスがクロックを発生する（図** 4.4)。

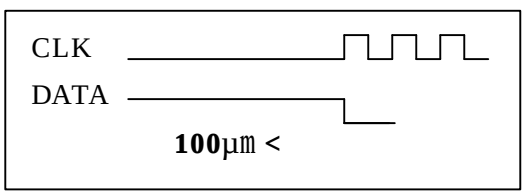


図 4.4 クロックの発生

- マウスからのデータを Buffer に保留。
  マウスから送ってもらったコマンド、返事あるいは状態データなどを解読するために、一時、Buffer に保留する必要がる。VHDL は vector 型で Buffer を用意できる。

```
signal DATA_IN_BUF : std_logic_vector(33 downto 0) ;
```

保留する部分は以下のようになる。

```
I <= 0;
if falling_edge( MOUSE_CLOCK ) then

        DATA_IN_BUF(I) <= MOUSE_DATA;
        I <= I + 1;
end if;
```

上の部分プログラムでマウスクロックを下がるたびに、同期のマウスデータを Buffer に保留される。

- マウスにコマンドを送る
  システムはコマンドで、マウスをコントロールする。
  まず、中間 Buffer に Command を用意する。

```
signal COMMAND_BUF : std_logic ;
constant COMMAND      : std_logic_vector(0 to 9):= "1001011101";
```

プログラムの中にマウスクロックを上がるたびに、マウスクロックと同期の信号を送る。

```
I <= 0;
if falling_edge( MOUSE_CLOCK ) then
        COMMAND_BUF <= COMMAND(I);
        I <= I + 1;
if I = 9 then
I <= 0;
```

4.3　結果

4.3.1　マウスクロックに同期の信号を生成すること。

マウスの通信アルゴリズム（図 2.6 に参照）によって、
DATA 信号を 100μm 以内 **high** レベル（システムの発信機を利用）に

してから、Low レベルに落とすと、マウスがクロックを発生する（図 4.4 に参照)。このマウスクロックを利用して、マウスクロックが下がるたびにデータをマウスに入力する。以下の部分ソースはマウスクロックの同期信号をマウスに入力する部分である。

```
constant COMMAND     : std_logic_vector(0 to 8):= "010111101";
--コマンドを設定する。
MOUSE_DATA <= '1' when WE = "00" else COMMAND_BUF  when WE =
"01" else 'Z' when WE = "10" else '0';
CHECK_DATA <= '1' when WE = "00" else COMMAND_BUF  when WE =
"01" else '1' when WE = "10" else '0';
--リセット、マウスクロックを発生させる、マウスに入力するモード、
マウスから出力を受け取るモードを切り換える部分。
elsif falling_edge( MOUSE_CLOCK ) then
        SETZERO <= '0';
        if WE = "01" then
                COMMAND_BUF <= COMMAND(I);
                I <= I + 1;

                if I = 9 then
                        I <= 0 ;
                        START <= '0';                    -
                end if;
--マウスに入力するモードで、マウスクロックに同期コマンドをマウス
に入力する。
```

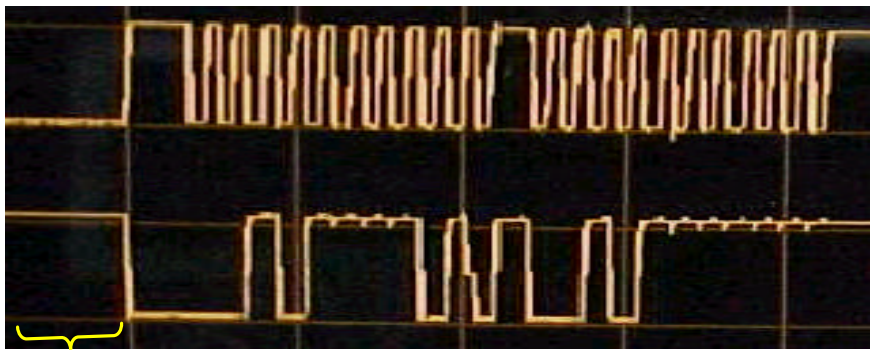


図 4.5　マウスクロックと同期コマンド

図 4.5 の ⌐¬ ところは 100μm 以上 CLK が Low、DATA が High を制御する。後、DATA を Low に下がるとマウスクロックが発振される。このクロックと同期の F4 コマンドをマウスに入力する。その後、マウスからの出力をうけとるモードに切り換えて、マウスの返事ＦＡを受ける。

4.3.2 コマンドを送って、マウスをコントロールすること。

上の 4.3.1 に説明したマウスクロックと同期のコマンドの送り方でソースの中の「**constant COMMAND : std_logic_vector(0 to 8):= "010111101";**」のところで 16 のコマンド（表 2.3 に参照）の中適当のコマンドを設定することができる。現在のシステムは 1 個のコマンドしか入力できない。

4.3.3 マウスからの状態（動いたり、ボタンを押されたり）データを受け、Buffer に保留する。

マウスは電源が ON するとき、自動的にデフォルトの状態（サンプル速度：100report/s、non-autospeed、stream mode、2dot/count、disable）をセットする。この状態の中 Enable(F4)コマンドをマウスに送るとマウスは図 4.6 のように状態（動いたり、ボタンを押されたり）データをシステムに送る。

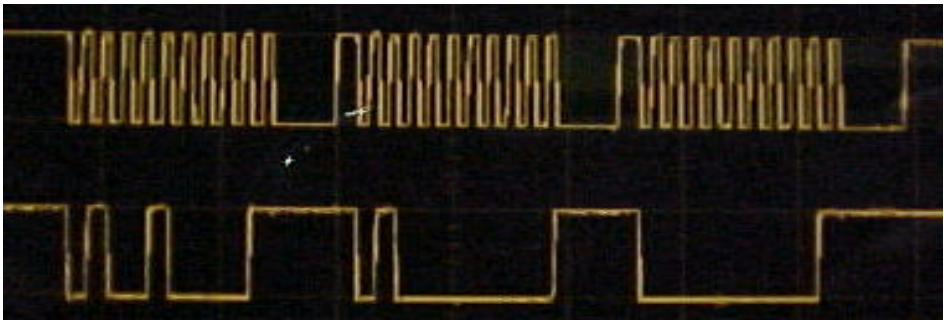


図 4.6　マウスの状態データ（ビットの意味は表 2.1 に参照）

システムはマウスクロックを利用して、データを Buffer に保留される。

signal DATA_IN_BUF : std_logic_vector(33 downto 0) ;

Buffer を用意する。

```
I <= 0;
if falling_edge( MOUSE_CLOCK ) then

        DATA_IN_BUF(I) <= MOUSE_DATA;
        I <= I + 1;
end if;
```

--マウスクロックを下がるたびに、同期のマウスデータを Buffer に保留される。

4.4　結果検証

4.4.1　検証装置

- UP1 ボード
  本研究室で Altera 社が提供した。20000 ゲットの FPGA とセグメント LED、スイッチ、Mouse Connector ,VGA Commector などから構成された UP1 ボード（図 4.5）を利用した。
  UP1 ボードは専用ケーブルで、コンピュータ本体のプリンタコネクターに接続して、MAX+PlusII ソフトで、ダウンロードできる。

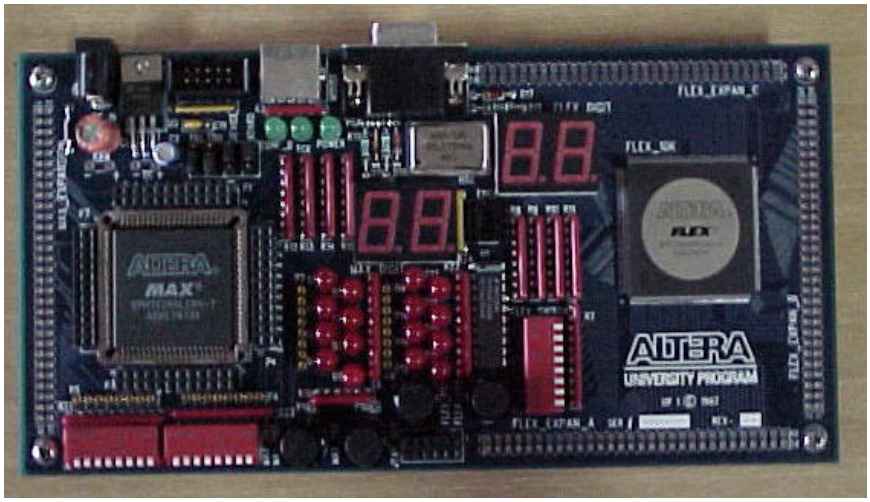


図 4.5　UP1 ボード

- 自作検証ボード
  図 4.6 のように 10000 ゲットの FPGA とセグメント LED、スイッチ、Mouse Connector などから検証ボードを作成した。

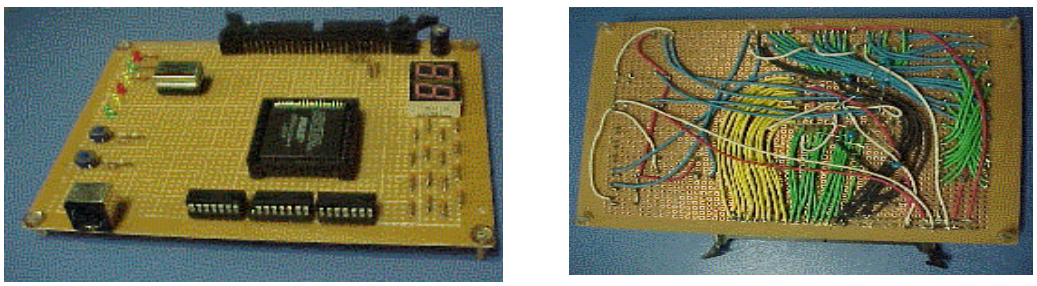


図 4.6 自作ボード

4.4.2　　自作検証ボードの設計図

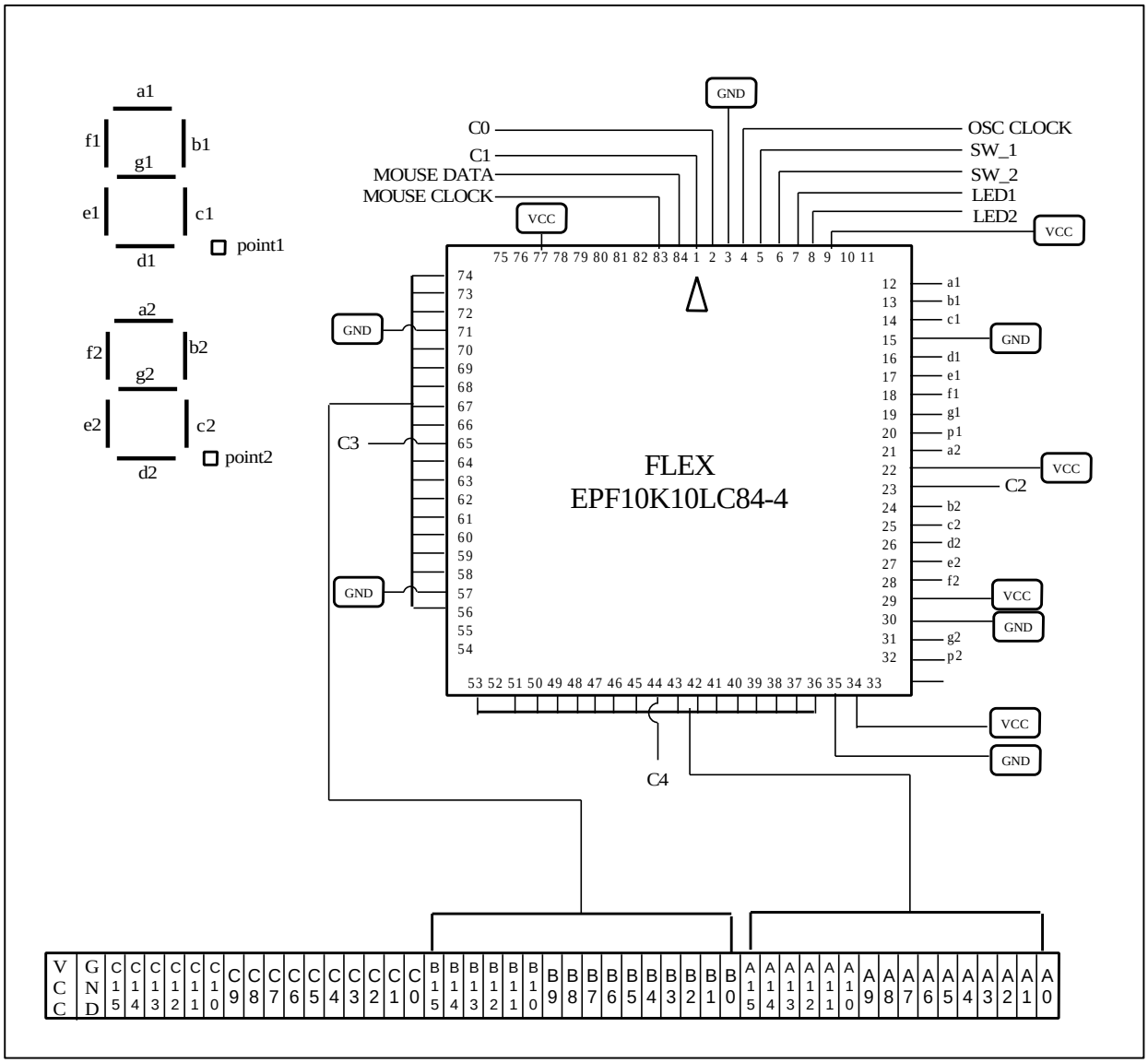


図 4.7　ボード設計図

4.2.2.1 基板の諸元

表 4.1 検証基板の諸元

名称	マウスコントローラ検証基板
FBGA	FLEX EPF10K10LC84-4
CLOCK	クロックオシレータより供給か
外部端子	A ポート B ポート C ポートそれぞれ 16 ビットと VCC GND
コネクタ形状	50pin ヘッダ
その他	スイッチ x2、セグメント LED x2

4.2.2.1 基板の詳細

このボードは松尾さんが製作した ISA BUS 16bit I/O Card と言うインターフェカード（松尾さんの論文参照）を利用して、コンピュターを交信する。
コンフィグレーションの時、インターフェースの信号 C0 C1 C2 B14 B15 を用いる。

表 4.2 コンフィグレーション Pin

C0	DATA0
C1	CLK 選択
C2	nCONFIG
B14	nSTATUS
B15	CONF_DONE

パソコンとの通信は A ポート(A0~A15)、B ポート(B0~B13)、C ポート(C3,C4)を用いる。
実行プログラムは松尾さんが製作したプログラムを利用する。

4.2.3 オシロスコープ

4 チャネルのオシロスコープ（図 4.7）で、通信ディジタル波形を見るために、マウスとシステムの間に接続する。実際に見たい信号は Mouse CLK と Mouse DATA であるから、2 チャネルだけで十分。

信号の active 幅は 5v であるから、チャンネルの V/dir が 5v を設定すればよい。

トリガー信号は Mouse CLK と Mouse DATA の信号のどちらを使用しても良いが、普段、CLK の信号をトリガーとする。トリガーレベルは 2V ぐらいにセットする。

Mouse CL の周波数は 200μm であるから、time/dir を 200μm を設定すればよい。

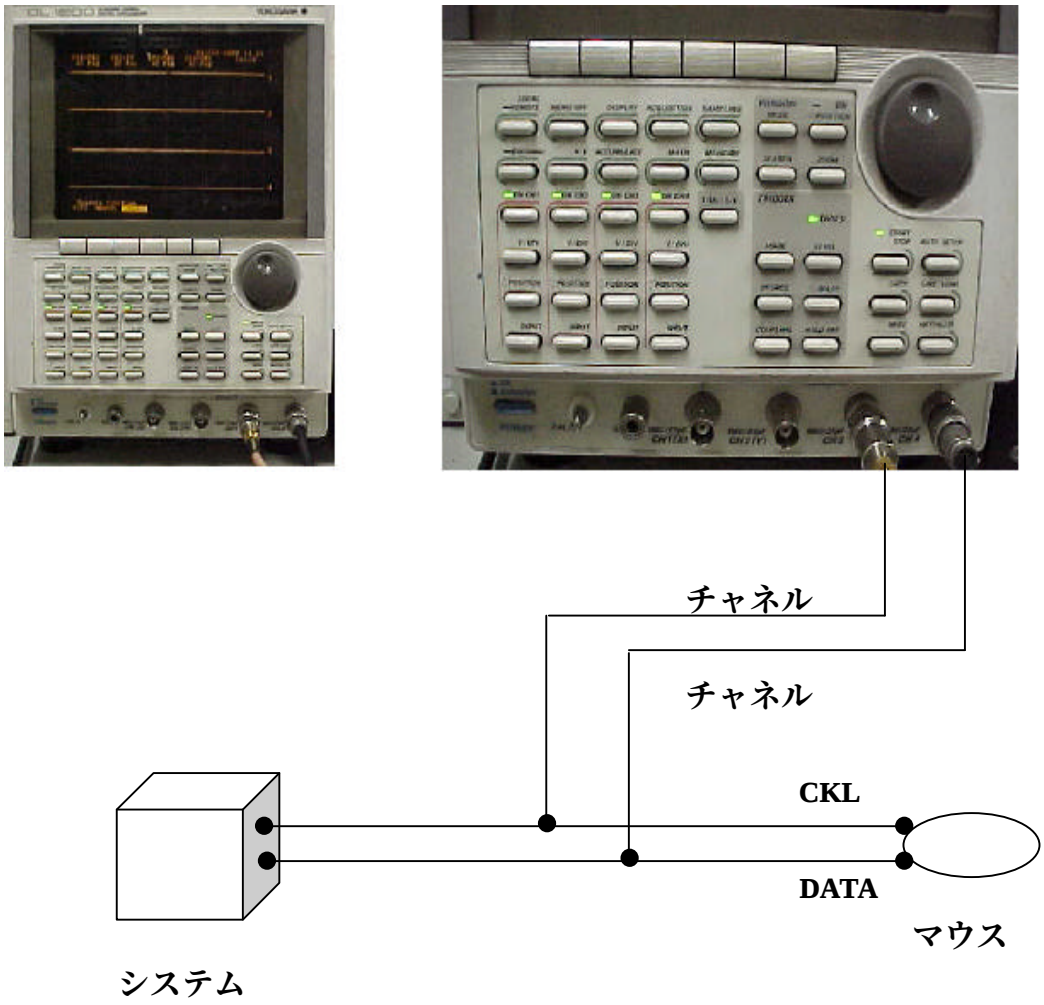


図 4.7　オシロスコープの接続回路図

4.4.4　検証結果

- オシロスコープで、図 4.1、4.2、4.3 のようにマウスとシステムの通信のコマンドとデータなどのディジタル波形で、確認できた。

第 5 章　結論

本研究は次の結果を得られた：

- システムとマウスの交信アルゴリズムを理解できた
- マウスコントロールコマンドでマウスをコントロールできた
- マウスからの状態データを受け取ることができた。

残り課題：

- 設計したシステムはまだ自動的にマウスからの信号を解読して、適切のコマンドを出すことができていない。（現在のシステムは 1 個のコマンドしか出せない）
- 受け取ったデータはまだ表示装置に表示させることができていない。

謝辞

この論文の作成及び本研究の進む過程に毎週、丁寧に報告を聞き、すっかり指導をしてくれました指導教官の齋藤理一郎助教授のおかげで、多少の研究結果が出てきました。ここに齋藤助教授に心よりご感謝の言葉を申し上げます。

そして、本研究にセミナを通して、意見と指導をしてけれました木村忠正教授、湯郷成美助教授、一色秀雄先生に深謝の意を申し上げます。

最後、研究室の沼知典先輩、松尾竜馬先輩にいろいろなことを教えてくれまして、心から感謝します。他の研究室のメンバーにも、よい研究雰囲気で、有意義の学部最後の１年をすごすことができまして、ご感謝します。

第 6 章　付録

6.1　カウンタのプログラム

6.1.1　FLEX8000 のカウンタ

```
-- 可変速 20 進アップカウンタ (FLEX8000)
-- Wind¥c:¥ho¥fpga¥counter¥count

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity countup3 is
port (
        SW_1,SW_2,CLK : in std_logic;
        CARRY : out std_logic;
        LED : out std_logic_vector(7 downto 0)
    );

attribute pinnum of LED    : signal is   "15,16,18,46,35,37,39,40";
attribute pinnum of SW_1   : signal is   "27";
attribute pinnum of SW_2   : signal is   "19";
attribute pinnum of CLK    : signal is   "50";
attribute pinnum of CARRY : signal is   "45";

end countup3;

architecture RTL of countup3 is

signal CLK_2     : std_logic_vector(20 downto 0); -- クロック分周
signal CLK_3     : std_logic;                     -- クロック分周
```

```
signal DCLK       : std_logic;                          -- 遅延クロック
signal CNT        : std_logic_vector(3 downto 0);   -- LED 表示カウント数
signal CNT_1      : std_logic_vector(3 downto 0);   -- SW_1 のカウント数
signal CNT_2      : std_logic_vector(3 downto 0);   -- SW_2 のカウント数
signal SW_STATE : std_logic_vector(1 downto 0);   -- スイッチの状態
signal CRY        : std_logic;                          -- 内部参照用キャリー
signal N          : std_logic_vector(3 downto 0);   -- カウント速度
signal ST         : std_logic;                          -- スイッチの状態

begin

-- クロック分周
  process begin
    wait until CLK'event and CLK = '1';   -- クロックの立上がり
    CLK_2 <= CLK_2 + '1';
  end process;

-- カウント速度決定
  process ( N, CLK_2 ) begin
    case N is
      when "0000" => DCLK <= CLK_2(8);     -- 最大速
      when "0001" => DCLK <= CLK_2(9);
      when "0010" => DCLK <= CLK_2(10);
      when "0011" => DCLK <= CLK_2(11);
      when "0100" => DCLK <= CLK_2(12);
      when "0101" => DCLK <= CLK_2(13);
      when "0110" => DCLK <= CLK_2(14);
      when "0111" => DCLK <= CLK_2(15);
      when "1000" => DCLK <= CLK_2(16);
      when "1001" => DCLK <= CLK_2(17);   -- 最小速
      when others => DCLK <='X';
    end case;
  end process;

  CLK_3 <= CLK_2(15);           -- カウント速度決定カウンタ用クロック
  SW_STATE <= SW_1 & SW_2;     -- スイッチの状態
```

```
-- 遅延クロック同期カウンタの動作
  process begin
    wait until DCLK'event and DCLK = '1';   -- 遅延クロックの立上がり
    case SW_STATE is
      when "00" =>                          -- SW_1,SW_2 OFF
        CNT_1 <= "0000";                    -- カウント初期化
        CRY <= '0';                         -- キャリー初期化

      when "01" =>                          -- SW_1 ON ,SW_2 OFF
        if ( CNT_1 = "1001" ) then          -- 最大数のとき
          CNT_1 <= "0000";                  -- カウント初期化
          CRY <= not CRY;                   -- 桁上げ
        else
          CNT_1 <= CNT_1 + "0001";          -- カウントアップ
        end if;

      when others => null;
    end case;
  end process;

-- カウント速度決定カウンタの動作
  process begin
    wait until CLK_3'event and CLK_3 = '1'; -- 遅延クロックの立上がり
    case SW_STATE is
      when "00" =>                          -- SW_1,SW_2 OFF
        N <= "0000";                        -- カウント速度の初期化
        CNT_2 <= "0000";                    -- カウント初期化

      when "10" =>                          -- SW_1 OFF, SW_2 ON
        if N = "1001" then                  -- カウント速度最大のとき
          N <= "0000";                      -- カウント速度の初期化
          CNT_2 <= "0000";                  -- カウント初期化
        else
          N <= N + '1';                     -- カウントアップ(スピードアップ)
          CNT_2 <= CNT_2 + '1';             -- カウントアップ
```

```
        end if;

      when others => null;                    -- その他のときは何もしない
    end case;
  end process;

-- スイッチの状態を確認
  process ( SW_1 , SW_2 ) begin
    if ( SW_1 = '0' ) then          -- SW_1 ON
      ST <= '0';
    elsif ( SW_2 = '0') then        -- SW_2 ON
      ST <= '1';
    end if;
  end process;

  CNT <= CNT_1 when ST = '0' else   -- SW_1 ON なら CNT_1 を、
         CNT_2 ;                    -- SW_2 ON なら CNT_2 をLEDに表示

-- カウンタの値を LED に表示
  process ( CNT ) begin
    case CNT is
      when "0000" => LED <= "11111110";   -- " 0 "を表示
      when "0001" => LED <= "00110000";   -- " 1 "を表示
      when "0010" => LED <= "11101101";   -- " 2 "を表示
      when "0011" => LED <= "01111001";   -- " 3 "を表示
      when "0100" => LED <= "10110011";   -- " 4 "を表示
      when "0101" => LED <= "01011011";   -- " 5 "を表示
      when "0110" => LED <= "11011111";   -- " 6 "を表示
      when "0111" => LED <= "01110000";   -- " 7 "を表示
      when "1000" => LED <= "11111111";   -- " 8 "を表示
      when "1001" => LED <= "01110011";   -- " 9 "を表示
      when others => LED <= "XXXXXXXX";   --  不定
    end case;
  end process;
  CARRY <= '0' when ST = '1' else CRY;    -- SW_1 ON のとき、キャリー表示
end RTL;
```

6.1.2 UP1 ボードのカウンタ

```
-- Wind¥c:¥ho¥fpga¥up1¥count3¥UP1COUNT3.VHD

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity count3 is
   port ( SW_1,SW_2,CLK : in std_logic;
            -- CARRY : out std_logic;
              LED1 : out std_logic_vector(7 downto 0);
        LED2 : out std_logic_vector(7 downto 0)
          );

attribute pinnum of LED2     : signal is   "6,7,8,9,11,12,13,14";
attribute pinnum of LED1     : signal is   "17,18,19,20,21,23,24,25";
attribute pinnum of SW_1    : signal is   "28";
attribute pinnum of SW_2    : signal is   "29";
attribute pinnum of CLK     : signal is   "91";
--attribute pinnum of CARRY : signal is   "45";

end count3;

architecture RTL of count3 is

signal CLK_2       : std_logic_vector(22 downto 0);   --
signal DCLK         : std_logic;                                --
signal CNT1          : std_logic_vector(3 downto 0);     --
signal CNT2          : std_logic_vector(3 downto 0);     --
signal SW_STATE : std_logic_vector(1 downto 0);     --
```

```
begin

--
  process begin
    wait until CLK'event and CLK = '1';     --
    CLK_2 <= CLK_2 + '1';
         SW_STATE <= SW_1 & SW_2;
         if SW_STATE = "11" then
         CLK_2 <= "00000000000000000000000";                    -- -   -
         end if;
  end process;
  DCLK <= CLK_2(22);

  process begin
    wait until DCLK'event and DCLK = '1';
    case SW_STATE is
        when "00" =>                      -- SW_1,SW_2 ON
        CNT1 <= "0000";                  -- -   -
        CNT2 <= "0000";                  --

      when "01" =>                       -- SW_1 ON ,SW_2 OFF
        -- count up
        CNT1 <= CNT1 + "0001";
            if (CNT1 = "1001")then
                        CNT1 <= "0000";
                    CNT2 <= CNT2 + "0001";
        end if;
                if (CNT2 = "1001")then
                        CNT2 <= "0000";
                end if;

      when "10" =>                       -- SW_1 OFF, SW_2 ON
        -- count down
        CNT1 <= CNT1 - "0001";
            if (CNT1 = "0000")then
                        CNT1 <= "1001";
```

```
                    CNT2 <= CNT2 - "0001";
            end if;
                    if (CNT2 = "0000")then
                            CNT2 <= "1001";
                    end if;

     --   others => null;              --
       end case;
     end process;
     -- The count processing for first LED
     process ( CNT1 ) begin
       case CNT1 is
             when "0000" => LED1 <= "00000010";   -- " 0 " Out put
             when "0001" => LED1 <= "10011110";   -- " 1 "
             when "0010" => LED1 <= "00100100";   -- " 2 "
          when "0011" => LED1 <= "00001100";   -- " 3 "
             when "0100" => LED1 <= "10011000";   -- " 4 "
             when "0101" => LED1 <= "01001000";   -- " 5 "
             when "0110" => LED1 <= "01000000";   -- " 6 "
             when "0111" => LED1 <= "00011110";   -- " 7 "
             when "1000" => LED1 <= "00000000";   -- " 8 "
             when "1001" => LED1 <= "00011000";   -- " 9 "
             when others => LED1 <= "XXXXXXXX";   -- Not thing
       end case;
     end process;
  --The count processing for second LED

  process ( CNT2 ) begin
       case CNT2 is
             when "0000" => LED2 <= "00000010";   -- " 0 " Out put
             when "0001" => LED2 <= "10011110";   -- " 1 "
             when "0010" => LED2 <= "00100100";   -- " 2 "
          when "0011" => LED2 <= "00001100";   -- " 3 "
             when "0100" => LED2 <= "10011000";   -- " 4 "
             when "0101" => LED2 <= "01001000";   -- " 5 "
             when "0110" => LED2 <= "01000000";   -- " 6 "
```

```
            when "0111" => LED2 <= "00011110";   -- " 7 "
            when "1000" => LED2 <= "00000000";   -- " 8 "
            when "1001" => LED2 <= "00011000";   -- " 9 "
            when others => LED2 <= "XXXXXXXX";   -- Not thing
      end case;
   end process;


end RTL;
```

6.2　マウスコントローラシステム

6.2.1　UP1 ボードの VHDL ソースファイル

```
-- Wind¥c:¥ho¥fpga¥up1¥mouse¥Mouse7¥mouse7.VHD
--Mouse oparation VHDL source file
-- file name "mouse7.vhd"

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

                    --~~~~~~~~~~~~~~~~~~
entity mouseA is
   port ( MOUSE_DATA,MOUSE_CLOCK,CHECK_DATA : inout std_logic;
                     SW_1,SW_2,CLK : in std_logic;
                     LED1 : out std_logic_vector(7 downto 0);
          LED2 : out std_logic_vector(7 downto 0)
        );
```

```
attribute pinnum of MOUSE_DATA    : signal is   "120";
attribute pinnum of MOUSE_CLOCK   : signal is   "109";
attribute pinnum of CHECK_DATA    : signal is   "129";

attribute pinnum of SW_1    : signal is   "28";
attribute pinnum of SW_2    : signal is   "29";
attribute pinnum of CLK     : signal is   "91";
attribute pinnum of LED1    : signal is   "6,7,8,9,11,12,13,14";
attribute pinnum of LED2    : signal is   "17,18,19,20,21,23,24,25";

end mouseA;

                    --~~~~~~~~~~~~~~~~~~~~
architecture RTL of mouseA is


signal DCLK                 : std_logic_vector(31 downto 0);
signal CLK_SAMP   : std_logic;
signal CNT1         : std_logic_vector(3 downto 0);
signal CNT2         : std_logic_vector(3 downto 0);
signal WE           : std_logic_vector(1 downto 0);
signal START        : std_logic;
signal SETZERO    : std_logic;
signal I            : integer range 0 to 30;
signal CNT                  : integer range 0 to 2518;
-- constant COMMAND    : std_logic_vector(0 to 9):= "1111111111"; --
constant COMMAND    : std_logic_vector(0 to 8):= "010111101"; --"F4"( 0
0010 1111 11    '4F')"Enable"
-- constant COMMAND    : std_logic_vector(0 to 9):= "1001011101"; --"E9"( 0
1001 0111 01)"Status Request
--constant COMMAND          : std_logic_vector(0 to 29) :=
"10010111011001011011001011101";
signal MOUSE_WAIT : std_logic ;

signal COMMAND_BUF : std_logic ;
```

```
signal DATA_IN_BUF : std_logic_vector(33 downto 0) ;

signal DCLK1      : integer range 0 to 5035;
                  --~~~~~~~~~~~~~~~~~~
begin

MOUSE_DATA <= '1' when WE = "00" else COMMAND_BUF when WE =
"01" else 'Z' when WE = "10" else '0';
CHECK_DATA <= '1' when WE = "00" else COMMAND_BUF when WE =
"01" else '1' when WE = "10" else '0';
--~~~~~~~~
MOUSE_CLOCK <= '0' when WE = "00" else 'Z';

process (SW_1, CNT, START) begin
if SW_1 = '0' then
        MOUSE_WAIT <= '1';
elsif CNT = 2517 then     -- wait 100 us
        MOUSE_WAIT <= '0';
end if;
end process;

process ( SW_1, CNT, CLK ) begin

if SW_1 = '0' or CNT = 2518 then
        CNT <= 0;
elsif rising_edge( CLK ) then
        if MOUSE_WAIT = '1' then
                CNT <= CNT + 1;
        end if;
end if;
end process;

process (SW_1, MOUSE_WAIT, MOUSE_CLOCK, WE, I) begin
if SW_1 = '0' then
        for J in 0 to 33 loop
                DATA_IN_BUF(J) <= '0';
```

```
            end loop;
            COMMAND_BUF <= '0';
            I <= 0;
            START <= '1';
            SETZERO <= '1';

    elsif falling_edge( MOUSE_CLOCK ) then
            SETZERO <= '0';
            if WE = "01" then
                    COMMAND_BUF <= COMMAND(I);
                    I <= I + 1;

                    if I = 9 then
                            I <= 0 ;
                            START <= '0';                    -
                    end if;

            elsif WE = "10" then

                    DATA_IN_BUF(I) <= MOUSE_DATA;
                    I <= I + 1;
                    if I = 33 then
                            START <= '1';
                            I <= 0 ;
                    end if;

            end if;
    end if;
    end process;

    process (SW_1,MOUSE_WAIT,START,SETZERO) begin
    if SW_1 = '1' then
            CNT1 <= "0010";
            CNT2 <= "0010";

            WE <= "00";
```

```
end if;

if SW_1 = '0' then
        WE <= "00";

        CNT1 <= "1001";
        CNT2 <= "1001";
elsif MOUSE_WAIT = '1' then
                WE <= "00";
elsif MOUSE_WAIT = '0' then
        if SETZERO = '0' then
                if START = '1' then
                        WE <= "01";
                else
                        WE <= "10";
                end if;
        else
                WE <= "11";
        end if;
else

end if;
end process;



--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -- The count processing for first LED
  process ( CNT1 ) begin
    case CNT1 is
          when "0000" => LED1 <= "00000010";   -- " 0 " Out put
          when "0001" => LED1 <= "10011110";   -- " 1 "
          when "0010" => LED1 <= "00100100";   -- " 2 "
        when "0011" => LED1 <= "00001100";   -- " 3 "
          when "0100" => LED1 <= "10011000";   -- " 4 "
          when "0101" => LED1 <= "01001000";   -- " 5 "
```

```
            when "0110" => LED1 <= "01000000";   -- " 6 "
            when "0111" => LED1 <= "00011110";   -- " 7 "
            when "1000" => LED1 <= "00000000";   -- " 8 "
            when "1001" => LED1 <= "00011000";   -- " 9 "
            when others => LED1 <= "XXXXXXXX";   -- Not thing
    end case;
  end process;

--The count processing for second LED
process ( CNT2 ) begin
    case CNT2 is
            when "0000" => LED2 <= "00000010";   -- " 0 " Out put
            when "0001" => LED2 <= "10011110";   -- " 1 "
            when "0010" => LED2 <= "00100100";   -- " 2 "
       when "0011" => LED2 <= "00001100";   -- " 3 "
            when "0100" => LED2 <= "10011000";   -- " 4 "
            when "0101" => LED2 <= "01001000";   -- " 5 "
            when "0110" => LED2 <= "01000000";   -- " 6 "
            when "0111" => LED2 <= "00011110";   -- " 7 "
            when "1000" => LED2 <= "00000000";   -- " 8 "
            when "1001" => LED2 <= "00011000";   -- " 9 "
            when others => LED2 <= "XXXXXXXX";   -- Not thing
    end case;
  end process;


end RTL;
```

6.2.2　シミュレーションのテストベンチファイル

```
-- Wind¥c:¥ho¥fpga¥up1¥mouse¥Mouse7¥MOUSE7-TEST.VHD
-- テストベンチファイル
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
use IEEE.std_logic_unsigned.all;

use work.all;

entity testMouse is                    --testMouse is the file name ( can change )
end testMouse;

architecture RTL of testMouse is
--architecture TESTBENCH of testMouse is

component mouseA is              --  [MouseA] is the filename of simulation
  port ( MOUSE_DATA,MOUSE_CLOCK : inout std_logic;              -- same
the port of source file
                    SW_1,SW_2,CLK : in std_logic;
--                  CARRY : out std_logic;
                    LED1 : out std_logic_vector(7 downto 0);
          LED2 : out std_logic_vector(7 downto 0)
        );
end component;

--signal MOUSE : std_logic;
signal MOUSE_DATA :std_logic;
signal MOUSE_CLOCK : std_logic;
signal SW_1 : std_logic;
signal SW_2 : std_logic;
signal CLK : std_logic;
signal LED1 : std_logic_vector(7 downto 0);
signal LED2 : std_logic_vector(7 downto 0);

begin

DUT : mouseA port map ( MOUSE_DATA, MOUSE_CLOCK, SW_1, SW_2,
CLK, LED1, LED2 );      --in () write all signal name

process begin
CLK <= '0';
```

```
wait for 5 ns;
CLK <= '1';
wait for 5 ns;
end process;

process begin
wait for 100ns;
SW_1 <= '0';
wait for 100ns;
SW_1 <= '1';
MOUSE_CLOCK <= 'Z';
MOUSE_DATA <= 'Z';
wait for 25155 ns;
--MOUSE_DATA <= 'Z';
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '0';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
```

```
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '0';
for I in 0 to 10 loop
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '0';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
--MOUSE_DATA <= '1';
end loop;
wait for 22194 ns;
SW_2 <= '0';
for I in 0 to 15 loop
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
```

```
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';

end loop;
wait for 25194 ns;
SW_2 <= '1';
for I in 0 to 21 loop
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
MOUSE_DATA <= '0';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
wait for 50ns;
MOUSE_CLOCK <= '0';
wait for 50ns;
MOUSE_CLOCK <= '1';
```

```
MOUSE_CLOCK <= '1';
MOUSE_DATA <= '1';
end loop;


wait;
end process;
end RTL;
--end TESTBENCH;
```