2001 年度 卒業論文

ハードウェア記述言語を用いたＰＣの周辺機器の設計

電気通信大学 電気通信学部 電子工学科

9720002　阿部 正之

指導教官 齋藤 理一郎 助教授

提出日 平成 14 年 2 月 7 日

目次

謝辞

本研究および論文作成にあたり、懇切なる御指導、を賜わりました指導教官である
齋藤理一郎助教授に心より御礼の言葉を申し上げます。本研究およびセミナー等で御指導を賜りました木村忠正教授、湯郷成美助教授、一色秀夫助手に厚く感謝の意を表します。また、本研究をするにあたり、さまざまな資産を残して頂いた八木将志様、中島瑞樹様、松尾竜馬様、
グエン・ドゥック・ミン様、山岡寛明氏様、ホー・フィ・クー様、沼知典様、清水信貴様に多大なる感謝をいたします。特に清水信貴様には丁寧に直接指導して頂きました。改めて感謝致します。さらに、木村研究室、湯郷研究室の皆様方にも感謝致します。本研究にあたって、 Max+PlusII を無償で提供して頂きましたアルテラ・ユニバーシティプログラムマネージャー宮田様をはじめ、日本アルテラ（株）にも感謝致します。

第 1 章

序論

1.1 本研究の背景

大規模集積回路 (LSI) を使用することは、高機能な製品を開発するためには必要不可欠なものになっている。 PC から携帯電話などの従来から LSI を使用していた物以外にも最近は、自動車、家電製品などにも使用されている。その結果、 LSI は高集積化と高速化に従い、電子回路の設計はますます複雑で困難な物になってきている。数万ゲートを超える大規模回路では従来のゲートレベルでの方法では短期間で開発することが難しくなってきている。そこで、従来のゲートレベル設計に対する次世代の設計技術として、ハードウェア記述言語 HDL を用いたハイレベル設計手法が実用化されてきている。本研究室では、書き込み可能な LSI(FPGA) と HDL を用いて研究を行っている。
97 年度に松尾 [1] とグェン [2] が行列専用計算機の設計手順を決めた。また、松尾は FLEX10k シリーズの EPF10k100GC503-4 の FPGA を 2 個と SRAM,DRAM からなる実験基板を作成した。昨年までは、この基板を使用して VHDL で設計した回路を実装し PC からの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 に表示するということを行ってきた。しかし、本研究では、 FPGA に回路を実装した後、 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システムを考えた。

1.2 研究目的

本研究の目的は PC のインターフェイスである PS/2 マウスと PS/2 キーボードのコントロールシステムの設計を行う。 PS/2 インターフェイスの通信基準にしたがってシステムと交信する。設計するコントロールシステムはインターフェイスの通信アルゴリズムを理解し、状態データ (動いたり、ボタンを押されたり など) を送られるようにコントロールする。また、 PS/2 インター

フェイスの状態データを出力するシステムとして VGA コントロールの制御も行う。

1.3 本論文の構成

第 2 章において本研究を行う上での各機材、ソフトウェアの説明を記す。第 3 章では VGA のコントロールシステムの設計について説明する。第 4 章においては PS/2 マウスのコントロールシステムの設計について述べる。また、付録としてこの研究で使用する全てのソフトウェア使用法を詳しく解説する。さらに教育用の FPGA 搭載ボードの使用法と PS/2 キーボードのコントロールシステムの設計ついて解説する。

1.4 用語の説明

以下において、本研究を行う上での必要最低限の用語を説明する。

- VHDL
  VHDL(VHSIC Hardware Description Language) はハードウェア記述言語のひとつで、これと論理合成ツール (MaxPlusII など) を用いてハイレベル設計手法による論理回路の設計が、現在主流となってきている。
  VHDL の特徴は、このデジタル回路設計、シミュレーション、合成のすべてを実行するための幅広い構成を持っていること、そして、シミュレーションによる動作検証がしやすく、設計の変更に時間がかからないこと、言語記述がそれほど複雑ではないので、習得が容易である事が言える。
  また、 HDL には Verilog-HDL というのがあり、こちらは C 言語に近い構文の記述ができることや、各記述ブロック毎に各信号の入出力の指定が必要という特徴を持っている。

- FPGA
  FPGA(Field Programmable Gate Array) とは、 PLD(Programmable Logic Device) の一種で、何度でもデジタル回路機能を書込みできるゲート素子である。本研究で使用する FPGA はアルテラ社製の FLEX10k20 である。これは SRAM をベースに作られているので電源を入れるたびにダウンロードする必要がある。

第 2 章

設計方法と使用装置

2.1 設計方法

本研究において VHDL による回路設計の流れ図を図 2.1に示す。まず PeakFPGA という VHDL エディタ で回路機能 を記述し、 HDL ファイル (*.vhd) を作製する。もし、 VHDL 構文の記述がおかしければ、コンパイル中にエラーメッセージが出る。その場合は正しい文章に書き直し、再度コンパイルする。そして、この機能検証を実行するために、テストベンチをおこなう。これは、デバイスの入力ポートにデータを入力する動作を VHDL で記述したものをシミュレーションでデバイス内の各信号の動作を検証することである。このシミュレーションで信号の動作が正しくなければ、 VHDL ファイルを書き直しシミュレーションに戻り、信号の動作が正しくなるまで書き直す。

本研究の場合には、 FPGA でのピンの配置も VHDL ソース作成の時点で指定する必要がある。詳しい説明は、付録の B.1.2 の [VHDL ファイル作成における注意点] に示すが、この場合には、entity の部分で port 文で入出力ピンを指定し、 attribute 文でピン番号を指定する必要がある。

そして、その回路の動作が正しければ、 PeakFPGA で論理合成を行なう。論理合成とは HDL によって記述された回路機能を AND や OR などの論理回路の形に変換することである。これにより HDL ファイルは EDIF(Electronic Design Interchange Format) ファイル (*.edf) に変換される。この EDIF ファイルはデジタル回路をテキストファイルで表したファイルで、回路を実際のデバイスに実装するための情報が含まれている。この EDIF ファイルを目的のデバイスにコンフィグレーションする為には、そのデバイスに合う形に変換しなければならないが PeakFPGA では出来ない。そこで Max+Plus2 というソフトウェアが必要になってくる。この Max+Plus2 でデバイスを指定して再びコンパイルする。すると評価基盤上にある FPGA、 FLEX10K に乗る TTF(True Type Format) ファイル (*.ttf) が作成される。この TTF ファイルを FPGA にダウン

ロードする。最後に機能を実装したFPGAにより実装検証を行い、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りHDLを書き直す。このようにして、実際の機能動作が正当なものになるまで繰り返し行うことになる。作成の流れは図2.1に示す。

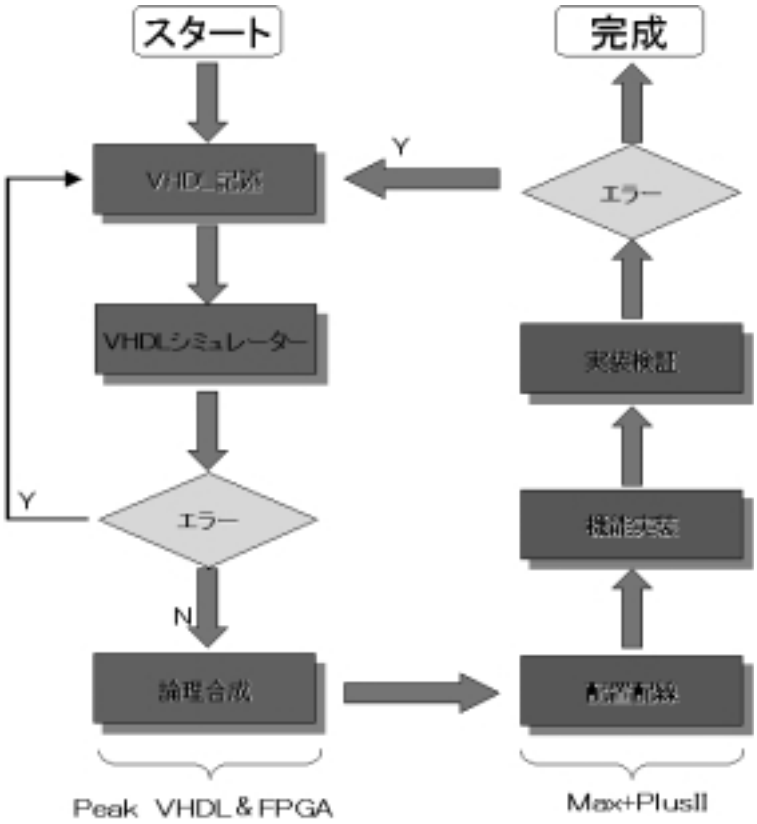


図 2.1: 本研究におけるデジタル回路設計の流れ [1]

2.2 ハードウェア

2.2.1 評価基盤

本研究で使われる評価基板は (株) 日本アルテラ社のユニバーシティ・プログラムに参加し、ご提供頂いた University Program Board(UP1board 図 2.2) と呼ばれる基板を使用する。この評価基板の構成は、 FPGA は 2 つ搭載されている。 1 つは FLEX10k であり、もう 1 つは MAX7000s である。各 FPGA には、 7 セグメントディスプレイがつながっており数字などを表示できる。また、本研究で使用するマウス、 VGA のコネクタが搭載されており、これらは FLEX10k に接続されている。

[1] ファイル名:u01mabe/zu/flow.ps

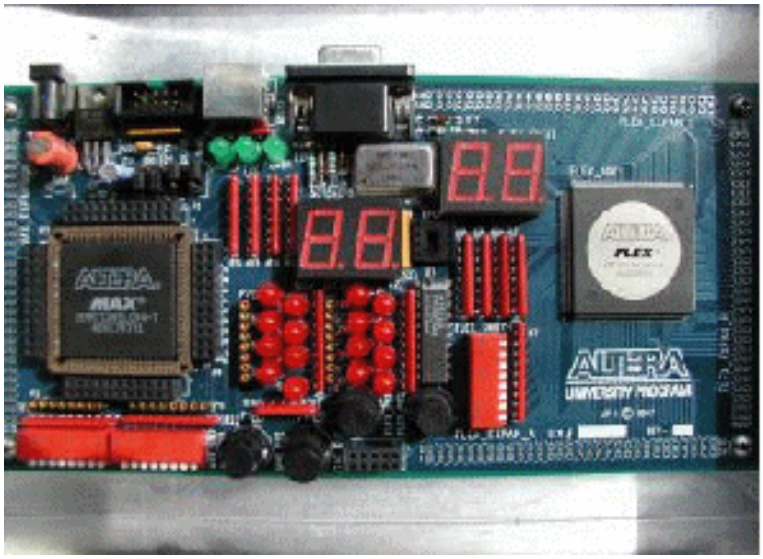


図 2.2: UP1 ボード [2]

また、 FPGA にダウンロードする際は、バイトブラスターというケーブルでダウンロードされる。バイトブラスターは、パソコンと 25 ピンのコネクタで接続され UP1 ボードとは JTAG-IN コネクタという、 10 ピンのコネクタで接続される。また、バイトブラスターは MaxPlusII Ver7 以上でないと使用できない。

2.3 ソフトウェア

2.3.1 PeakVHDL

VHDL を記述するにあたり、シミュレーション、論理合成、シミュレーションが出来るエディタが必要になる。そこでインターリンク社より PeakFPGA を購入した。現在のバージョンは PeakFPGA5.20c である。

2.3.2 Max+Plus2

VHDL エディタにより論理合成して出来た EDF ファイルを目標の FPGA にコンフィグレーションするためにコンパイルする必要がある。そこで Altera 社の MAX+Plus2 を使って行うこととする。現在のバージョンは Max+Plus2 10.0 である。

[2] ファイル名:u01mabe/zu/up1.ps

第 3 章

VGA のコントロールシステムの設計

3.1 目的

UP1 ボードに搭載されている FPGA の FLEX10k デバイスは、 VGA モニターに出力できる。その特徴を生かしてマウスの動きを FPGA をインターフェイスとして、 VGA モニターに出力する。ここでは、 VGA モニターへの出力の基本を学び、実際に思い通りの動作を VGA モニター上に出力する。

3.2 VGA インターフェイス

UP1 ボードには、 15 ピンの D-sub コネクタがある。これを使用して VGA モニターと接続できる。
EPF10k20 デバイスからは、 5 つの信号が定義されていて、それらを介して VGA モニターへ送られる。３つの VGA 信号は、赤、緑、青で残りの２つの信号は水平および垂直同期である。デバイスは、それらの信号を VHDL で操作して思い通りのイメージを VGA モニターへ出力することができる。

表 3.1: VGA 出力のピン設定

信号	D-sub ピン	EPF10k20
赤 (RED)	1	236
緑 (GREEN)	2	237
青 (BLUE)	3	238
GND	6.7.8.10.11	-
水平同期 (HORIZ-SYNC)	13	240
垂直同期 (VERT-SYNC)	14	239

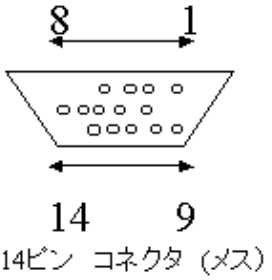


図 3.1: D-sub ピン [1]

3.3 VGA ドライバオペレーション

一般的な VGA モニターは、行と列で分けられたピクセルの格子から成り立っている。 VGA モニターは、縦 480 行、横 640 列を持つ。
それぞれのピクセルは赤、青、緑の 3 色の信号 (RGB 信号) の組み合わせによって、様々な色を写し出すことができる。
VGA モニターは、ピクセルを新しく変化させるときに決まる内部クロックを持っている。このクロックは、 VGA で決められた周波数 25.175MHz で操作している。

[1] ファイル名:u01mabe/zu/vga1.ps

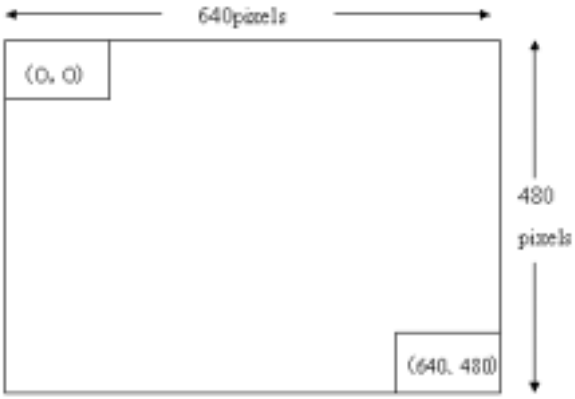


図 3.2: VGA モニター [2]

VGA モニターは水平および垂直同期信号によって画面をリフレッシュする。垂直同期信号を出力して X-Y 平面の起点とみなされる画面の左上の (0,0) からの出力を開始する。そして、水平同期信号を出力して、その画面の 1 行をリフレッシュする。その水平サイクルを 480 行行なった後、また、垂直同期信号を出力して再度画面をリフレッシュする。これを繰り返すことで、 VGA 出力が行なわれる。

3.4 VGA タイミング

VGA モニターを適切に動かすために、特別なパルスを特別な時間でデータを返さなければならない。水平および垂直同期パルスは、色のデータを返している間、モニターを同期する特別な時間に発生しなければならない。図 3.3に水平および垂直同期信号のサイクルを示す。

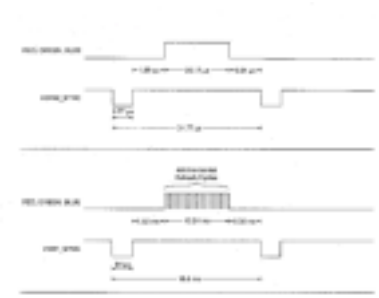


図 3.3: 水平および垂直同期信号のサイクル [3]

垂直同期について、最初に VERT-SYNC を'0' に下げる。これが画面のリフレッシュを開始する信号になる。 64 μ s 後にこの信号を'1' に戻す。それから 1.02ms 後に水平同期を始めます。

[2]ファイル名:u01mabe/zu/vga.ps

[3]ファイル名:u01mabe/zu/timing.ps

水平同期について、まず、 HORIZ-SYNC を'0' に下げる。これが画面の 1 行をリフレッシュを開始する信号です。 3.77 μ s 後にこの画面を'1' に戻す。それから 1.897 μ s 後に RGB 信号を出力する。
RGB 信号出力ではオシレータのクロック 1 周期で 1 ピクセルを表示すると考えて良い。
RGB1 列の出力が終わったら、水平同期を閉じるために 0.94 μ s 待つ。この水平同期を 480 回 (15.25ms) した後、垂直同期を閉じるために 0.35ms 待つ。これで画面 1 回分のリフレッシュサイクルが行なわれる。垂直同期サイクルの周期は 16.6ms となっている。
VGA のリフレッシュサイクルおよび周波数を下に示す。
T(pixel)(1 ピクセルを出力する時間) = 40ns
T(row)(1 行を出力する時間) = 31.77 μ s
T(screen)(1 画面を出力する時間) = 16.6ms
f(rr)(水平同期周波数) = 31.5kHz
f(sr)(垂直同期周波数) = 60Hz

3.5 結果

VGA モニターの基本を学ぶために、いろいろな VGA の制御を行った。以下に VGA の制御の主な例を示す。

- 7segVGA
  この VGA の制御は、 UP1 ボード上にある 7 セグメントディスプレイと VGA モニターを同時に制御する目的で行った。この制御は、 UP1 ボード上の 7 セグメントディスプレイと VGA モニター上に同時に 00 から 99 までカウントアップする。これは過去に作った UP 1 上の 7 セグメントディスプレイをカウントアッププログラムを元に、 VGA モニター上にも 7 セグメントディスプレイを作り、数字一つずつを制御して作成した。図 3.4に VGA 出力結果を示す。

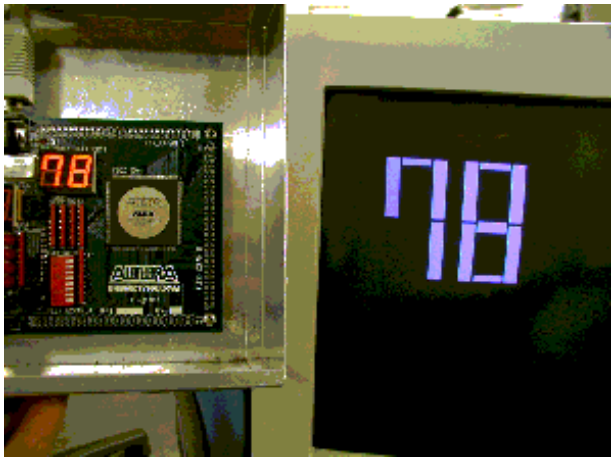


図 3.4: VGA モニターへの出力結果 [4]

図 3.4では分かりにくいが、7 セグメントディスプレイと VGA 出力はカウントアップし、同期も取れている。

- move24.48

  この VGA の制御は、 VGA 上で左右に動く物体を表示する目的で行った。まず、 24*48 の出力を表示するため 24*48 をビットで表示するプログラムを作成した。また、四角のある物体が左右に動くプログラムを作成した。そのあと二つのプログラムを組み合わせに左右に動く 24*48 ビットのプログラムを完成させた。 VGA モニターへの出力結果を図 3.5に示す。

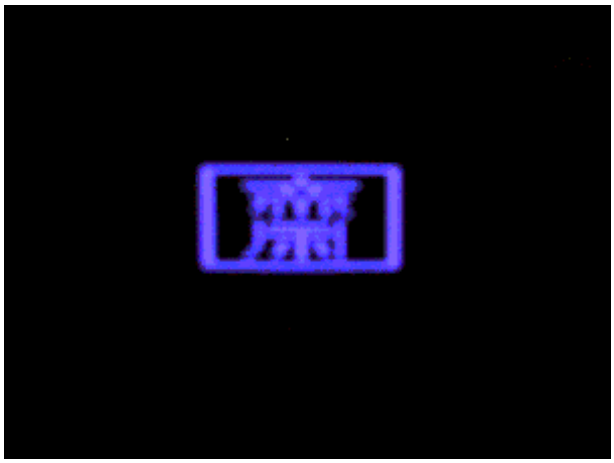


図 3.5: VGA モニターへの出力結果 [5]

[4] ファイル名:u01mabe/zu/vgakekka.ps

[5] ファイル名:u01mabe/zu/vgakekka1.ps

図 3.5では分かりにくいが、 24*48 ビット表示ができ VGA モニターの端から端まで左右に動く制御ができた。

第 4 章

PS/2 マウスのコントロールシステムの設計

4.1 目的

現在の AT 互換機のマウス・インターフェイスとしては、 PS/2 から採用されたものが一般的に採用されている。本研究では、その PS/2 マウスの構造や交信アルゴリズムについて理解してマウスの動作を UP1 ボードをインターフェイスとしてマウスの状態データをコントロールする。また、マウスの状態データを LED や VGA に表示することを目的として研究を行った。

4.2 PS/2 マウスの構造

本研究では、現在もっとも多く使われている PS/2 マウス (図 4.1) を使用する。 PS/2 マウスの構造を以下に示す。その詳細は図 4.2に示す。

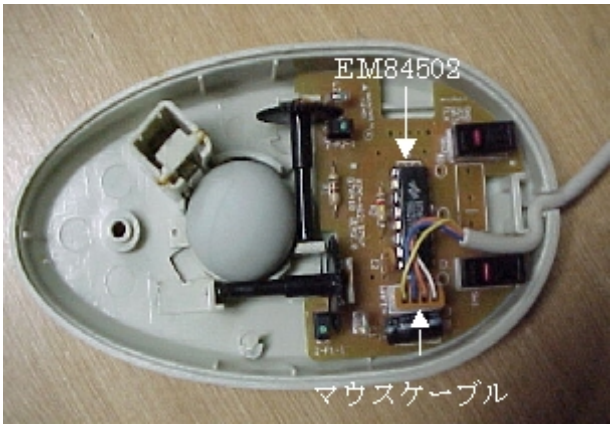


図 4.1: PS/2 マウスの構造 [1]

[1]ファイル名:u01mabe/zu/mouse.ps

マウスの内部は図 4.1のようになっている。中央にゴム製のボール、マウスの IC として EM84502、また、 PC との通信には 4 本のケーブルがつながっている。 4 本のケーブルにはそれぞれ、 5V 電源、 GND、マウスクロック、マウスデータがマウスコネクタに接続されている。

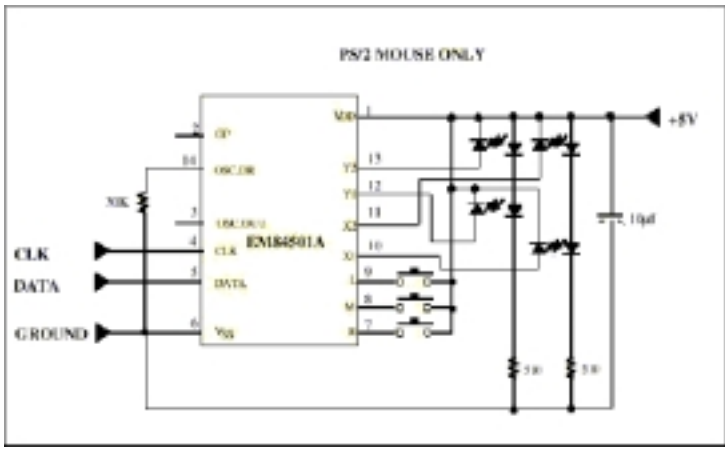
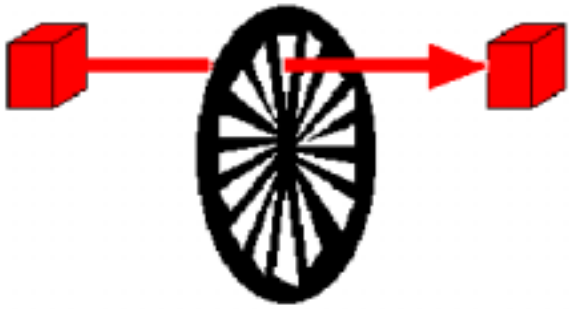


図 4.2: PS/2 マウスの回路構造 [2]

マウスの中の構造はボールおよびそれに付けられた 2 つの検出装置と、システムとやり取りするための回路基板から成り立っている。ボールに取りつけられた検出装置は互いに直角をなしているので、縦と横つまりあらゆる 2 次元の動きを検出することが可能である。

- ボールの動きの検出方法

図 4.3のように等間隔で穴の空いた円盤および発光素子、受光素子で構成されている。マウスが動かすとボールが回転し、それに合わせて X と Y 方向の円盤が回転する。すると発光素子から出た光は円盤を通過、遮断を繰り返す。受光素子は円盤に刻んだ空き間を通る光センサーを受け取るとそれをディジタル信号に変換する。したがって、受光素子から得られるディジタル信号はパルス信号となり、その間隔は円盤の回転速度に対応している。つまり、マウスを速く動かせばパルスの間隔は狭まり、ゆっくり動かせば広がる。



[2] ファイル名:u01mabe/zu/mouse0.ps

図 4.3: 信号発生の仕組み [3]

4.3 マウスの通信アルゴリズム

4.3.1 オペレーションモード

PS/2 マウスにはシステムと通信するために 4 つのオペレーションモードがある。以下にそれぞれのオペレーションの説明をする。

i) Reset Mode

このモードは、電源が ON の時、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。

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

ii) Stream Mode

以下の条件を満たすならデータをシステム側に転送する。

1) マウスのスイッチが押される。

2) マウスが動く。

iii) Remote Mode

read deta コマンドに返事するためのデータ転送

iv) Wrap mode

reset wrap mode コマンド (16 進数で EC) と reset コマンド (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。

[3]ファイル名:u01mabe/zu/mouse1.ps

4.3.2 PS/2 マウスデータ

i)Stream Mode のとき

データは各サンプルの間隔の終りに送られる。

ii)Remote Mode のとき

データは read data command に応じて送られる。

以下にマウスの bit の仕組みを示す。

表 4.1: マウス フォーマット [4]

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)

4.3.3 PS/2 マウスデータの転送システム

マウスにはマウスクロックとマウスデータがある。マウスはシステムにマウスクロックとマウスデータを入力し、システムは 5V 電源と GND をマウスに与える。
マウスデータは、マウスからシステムに送る時とシステムから受けるときは、マウスクロックに同期して送らなければならない。
また、マウスデータは 11 ビットを１パケットとし出力される。マウスクロックの１ビット目は、スタートビットと呼ばれるもので常に 0 を示す。また、 10 ビット目はパリティビットで奇数パリティを示す。 11 ビット目はエンドビットと呼ばれるもので常に 1 を示す。残りの 2-9 ビットはデータビットであり、ここでマウスコマンドが表される。マウスコマンドについては後で説明する。以下にマウスデータの bit の仕組みを示す。

[4]ファイル名:u01mabe/zu/format.ps

表 4.2: マウスの bit の仕組み[5]

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）

- DATA OUTPUT(マウスからシステムに)

  ここでは、マウスからシステムにデータの output について説明する。

  まず、マウスに電源が入るとマウスデータ、マウスクロック共に high の状態となり、転送できる状態になる。マウスデータはマウスクロックと同期して送られるがマウスクロックが low のときデータを転送しない。

  また、マウスクロックが high、マウスデータが low のときマウスデータを update する。図 4.4にデータ転送の例を示す。

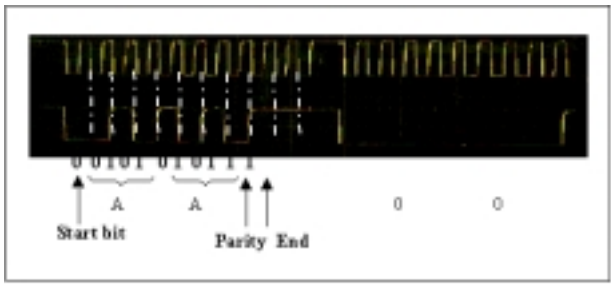


図 4.4: データ転送の例 [6]

- DATA INPUT(システムからマウスに)

  まず、システムはマウスからマウスデータが送られているかどうかチェックする。もし、マウスクロックが送られている途中か、 10 クロック目の前なら、 override でマウスクロックを inctive level にする。 10 クロック以後ならば、データを受け取る。また、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 100 μm 以下に inactive にする。

  マウスはスタートビットが送られるとマウスクロックを active にしてから 11 ビットのマウスクロックを出力する。マウスはクロックの 11 ビット目をチェックし、もし、 active ならマウスデータを low にしてもう 1 回クロックを出力する。

  システムはマウスからコマンドを入力されると、コマンドに応じたエコーコードを出力する。

[5] ファイル名:u01mabe/zu/trans.ps

[6] ファイル名:u01mabe/zu/hakei.ps

4.3.4 PS/2 マウスコマンド

PS/2 マウスにはシステムとの交信のために 16 個のコマンドが用意されている。コマンドは 16 進数であり、 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。
マウスからシステムにに FF,FE などのコマンドが送られると、システムは送られたコマンドに応じたエコーコードをマウスに送り返す。
以下にコマンドの一覧と説明を示す。

表 4.3: コマンド [7]

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

- Reset(FF)
  このコマンドを受けるとマウスは
  - すべてのコマンドをリセットし、マウスをデフォルト状態をセットする。
  - デフォルト状態
    デフォルトの状態とは、以下のものの状態になる。
    サンプル速度:100report/s , non-autospeed,stream mode,2dot/count,disable
- Resend(FE)
  - 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。
  - 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、最後のパケットが 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。
  - Stram mode のとき、もしマウスが 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。

[7] ファイル名:u01mabe/zu/command.ps

- Set Default(F6)
  - マウスがどんな状態であろうと、 Set Default コマンドを受ければマウスはデフォルトの状態をセットする。
- Disable(F5)
  - マウスが Stream mode のとき、このコマンドでデータ転送を中止する。
- Enable(F4)
  - マウスが Stream mode のとき、このコマンドでデータ転送が開始される。
- Set Sampling Rate(F3,XX)
  - マウスが Stream mode のとき、このコマンドは、下記の中で示されるコマンド XX によって示されたサンプル速度をセットする。

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

- Read Device Type(F2)
  - マウスはこのコマンドを受け取ると、 FA、 00 をシステムに返す。
- Set Remote Mode(F0)
  - Read Dada コマンドに返事するときだけ、マウスデータを送る。
- Set Wrap Mode(EE)
  - Reset(FF) コマンドあるいは、 Reset Wrap Mode(EC) コマンドが来るまで、 Wrap Mode が存続する。

- Reset Wrap Mode(EC)
  - マウスは、このコマンドを受けると現在のモードの一つ前のモードに戻る。
- Read Data(EB)
  - Remote Mode と Stream Mode のとき、このコマンドが実行される。このモードのときマウスが移動しなくても、またマウスのボタンが押されなくてもマウスデータは転送される。
- Set Stream Mode
  - このコマンドを受けるとマウスは、 Stream Mode をセットする。
- Status Repuest(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。状態データについては表 3.1 を参照
- Set Resolution(E8,XX)
  - コマンド XX の値に応じて分解度をセットする。

コマンド XX	Resolution
00	8 dot/count
01	4 dot/count
02	2 dot/count
03	1 dot/count

- Set Autospeed(E7)
  - Stream Mode のとき、 X,Y 方向のデータが更新される。
- Reset Autospeed(E6)
  - 普通のスピードに戻す。

4.4 結果

マウスの通信アルゴリズムにしたがって、 mouseCLK を 100 μ s 以内に High レベルから Low レベルに下げて、 mouseDATA を発生させることを行った。結果を下に示す。

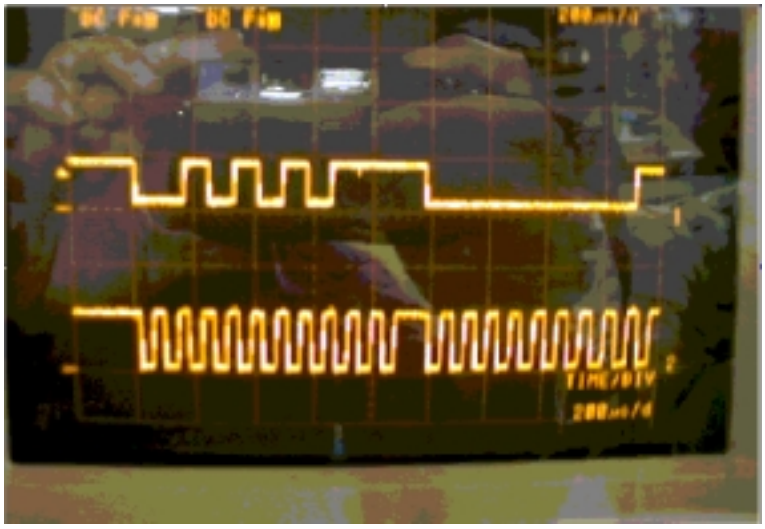


図 4.5: 結果 上が DATA、下が CLK[8]

これは、 UP1 ボード上にあるボタンを押すごとに mouseCLK が、 100 μ s 以内に下がり mouse-DATA が発生する。この mouseDATA は、 AA00 というマウスからのリセット信号である。 このあとマウスに Set Stream Mode コマンド (EA) を送り Stream Mode にした後、 Enable コマンド (F4) を送るとマウスのデータ転送が開始する。

[8]ファイル名:u01mabe/zu/kekka.ps

第 5 章

考察と今後の提案

UP1 ボードを用いて VGA モニター上への出力は制御できるようになった。この研究の最終目的であるマウスの状態データを VGA モニターに出力させるまでには至らなかったが、 mouseCLK を 100 μ s 以内に下げ mouseDATA を発生させるところまで出来た。
今後の課題はマウスにコマンドを送りマウスを自由に制御することが課題である。 mouseDATA が発生するので、その状態にマウスコマンドを送ればマウスを制御出来るであろう。また、マウスが制御出来るようになったら、 VGA モニターへの制御は完成しているのでそれを参考にし、最終的に VGA モニターへ出力することが必要である。さらに、 PS/2 キーボードにも取り組み、自由に制御させてみてはどうだろうか。 PS/2 キーボードは PS/2 マウスのコントロールシステムと似ているので PS/2 マウスの制御が出来れば、 PS/2 キーボードの制御も出来るであろう。

参考文献

[1] 松尾竜馬, “行列計算専用大規模集積回路の開発” ,1997 年度卒業論文

[2] グェン・ドゥック・ミン, “ハードウェア記述言語を用いた行列計算専用プロセッサの設計” ,1997 年度卒業論文

[3] EMC “EM84502 PS/2mouse controller”

[4] 宮崎 仁 “PS/2 キーボードインターフェイス”、 1987 年 10 月トランジスタ技術

[5] 吉田 功 “キーボードとマウスの構造” ,1995 年 10 月トランジスタ技術

[6] 桜井 至,”HDL によるデジタル設計の基礎”、 1997 年

付録 A

プログラムソース

A.1 7seg VGA

```
%
% ここから 7seg vga のプログラム
%
-----------------------------------
-- VGA Driver degitr by sw  --
-- 01/5/23   display,7segument --
-- c/m-abe/vga/7segvga/7segvga.vhd
-- m-abe        --
-----------------------------------

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 newVGA5 is
port ( CLK: in std_logic;
       RED   : out std_logic; -- color signal red
       GREEN : out std_logic; -- color signal green
       BLUE  : out std_logic; -- color signal blue
       HORIZ : out std_logic; -- synchronize signal horizontal
       VERT  : out std_logic; -- synchronize signal vertical
       DIGIT_2 : out std_logic_vector(7 downto 0); -- synchronize signal vertical
       DIGIT_1: out std_logic_vector(7 downto 0); -- synchronize signal vertical
       SW  : in std_logic); -- synchronize signal vertical

attribute pinnum of CLK : signal is "91";
attribute pinnum of RED : signal is  "236";
attribute pinnum of GREEN : signal is  "237";
attribute pinnum of BLUE : signal is  "238";
attribute pinnum of HORIZ : signal is  "240";
attribute pinnum of VERT : signal is  "239";
attribute pinnum of DIGIT_1 : signal is "6,7,8,9,11,12,13,14";
attribute pinnum of DIGIT_2 : signal is  "17,18,19,20,21,23,24,25";
attribute pinnum of SW : signal is  "28"; -- sw_1

end newVGA5; --franch

architecture RTL of newVGA5 is
signal HORIZ_SYNC : std_logic;    -- 水平方向の信号
signal HORIZ_CNT : integer range 0 to 800;  -- 水平方向の範囲
signal VERT_SYNC : std_logic;     -- 垂直方向の信号
signal VERT_CNT : integer range 0 to 523;   -- 垂直方向の範囲
```

```
signal RED1 : std_logic;          -- 水平方向の RED 信号
signal GREEN1 : std_logic;         -- 水平方向の GREEN 信号
signal BLUE1 : std_logic;         -- 水平方向の BLUE 信号
signal RED2 : std_logic;          -- 垂直方向の RED 信号
signal RED3 : std_logic;
signal CLK_2 : std_logic_vector(22 downto 0) ;  -- 分周のクロック
signal DCLK : std_logic;
signal LED  : std_logic_vector(3 downto 0):= "0000";
signal LED1  : std_logic_vector(3 downto 0):= "0000";
signal GREEN2 : std_logic;          -- 垂直方向の GREEN 信号
signal BLUE2 : std_logic;           -- 垂直方向の BLUE 信号
signal HYOUJI : std_logic_vector(7 downto 0);-- これは 1 なら表示 0 なら表示しない
signal HYOUJI1 : std_logic_vector(7 downto 0);
signal SX : std_logic_vector(7 downto 0);    -- これは十の位の水平方向の信号
signal SY : std_logic_vector(7 downto 0);    -- これは一の位の水平方向の信号
signal SS : std_logic_vector(7 downto 0);
signal TX : std_logic_vector(7 downto 0);    -- これは十の位の垂直方向の信号
signal TT : std_logic_vector(7 downto 0);    -- これは一の位の垂直方向の信号

begin

-- 水平方向の制御
process begin
wait until CLK'event and CLK = '1';
if ( HORIZ_CNT = 799 ) then            -- HORIZ_CNT が 799 ならば
HORIZ_CNT <= 0;                -- HORIZ_CNT を 0 に戻す
else
HORIZ_CNT <= HORIZ_CNT + 1;    -- それ以外なら HORIZ_CNT に 1 足す
end if;

end process;

-- 水平方向の制御
process begin
wait until CLK'event and CLK = '1';    -- CLK が 変化したら
case HORIZ_CNT is                      --HORIZ CNT が
when 0 =>                      -- 0 ならば
HORIZ_SYNC <= '0';     --HORIZ SYNC を 0
when 150 =>                    -- 150 ならば
HORIZ_SYNC <= '1';     --HORIZ SYNC を 1
when others =>                 -- それ以外は 無し
null;
end case;
end process;

-- 水平方向 RGB OUTPUT の制御
process begin
wait until CLK'event and CLK = '1';   -- CLK が変化したら
case HORIZ_CNT is                     -- HORIZ CNT が
when 202 =>               -- 202 ならば
 SX(4) <= '1';        --SX(4) を 光らせる
 SX(5) <= '1';        --SX(5) を 光らせる
when 218 =>                -- 218 ならば
 SX(4) <= '0';        --SX(4) を 光らせない
 SX(5) <= '0';        --SX(5) を 光らせない
when 222 =>                -- 222 ならば
 SX(0) <= '1';        --SX(0) を 光らせる
 SX(3) <= '1';        --SX(3) を 光らせる
 SX(6) <= '1';        --SX(6) を 光らせる
when 258 =>                  -- 258 ならば
 SX(0) <= '0';        --SX(0) を 光らせない
 SX(3) <= '0';        --SX(3) を 光らせない
 SX(6) <= '0';        --SX(6) を 光らせない
when 262 =>                -- 262 ならば
   SX(1) <= '1';       --SX(1) を 光らせる
 SX(2) <= '1';        --SX(2) を 光らせる
```

```
when 278 =>                -- 278 ならば
 SX(1) <= '0';         --SX(1)を 光らせない
 SX(2) <= '0';         --SX(2)を 光らせない
when others =>                -- それ以外は
null;                  -- なしよ。
end case;
end process;

process begin
wait until CLK'event and CLK = '1';   -- CLK が 変化すると
case HORIZ_CNT is                     -- HORIZ CNT が
when 302 =>               -- 302 ならば
 TX(4) <= '1';         --TX(4)を 光らせる
 TX(5) <= '1';         --TX(5)を 光らせる
when 318 =>                -- 318
 TX(4) <= '0';         --TX(4)を 光らせない
 TX(5) <= '0';         --TX(5)を 光らせない
when 322 =>                -- 322
 TX(0) <= '1';         --TX(0)を 光らせる
 TX(3) <= '1';         --TX(3)を 光らせる
 TX(6) <= '1';         --TX(6)を 光らせる
when 358 =>                 -- 358
 TX(0) <= '0';         --TX(0)を 光らせない
 TX(3) <= '0';         --TX(3)を 光らせない
 TX(6) <= '0';         --TX(6)を 光らせない
when 362 =>                -- 362
   TX(1) <= '1';        --TX(1)を 光らせる
 TX(2) <= '1';         --TX(2)を 光らせる
when 378 =>                -- 378
 TX(1) <= '0';         --TX(1)を 光らせない
 TX(2) <= '0';         --TX(2)を 光らせない
when others =>                -- それ以外は
null;                  -- ないの。
end case;
end process;

-- 垂直方向 の制御
process begin
wait until HORIZ_SYNC'event and HORIZ_SYNC = '0';  --HORIZ SYNC が '1' に変化すると,

if ( VERT_CNT = 523 ) then                    -- もし VERT CNT が 523 ならば
VERT_CNT <= 0;                        -- VERT CNT に'0' を代入
else                                  -- それ以外は
VERT_CNT <= VERT_CNT + 1;             --VERT CNT に 1 を足す
end if;
end process;

-- 垂直方向の制御
process begin
wait until HORIZ_SYNC'event and HORIZ_SYNC = '1';  -- HORIZ SYNC が変化したら


case VERT_CNT is                               -- VERT CNT が
when 0 =>                              -- 0 ならば
VERT_SYNC <= '0';              -- VERT SYNC に 0を入れる
when 10 =>                             -- 10 ならば

VERT_SYNC <= '1';              -- VERT SYNC に 1 を入れる
when others =>
null;

end case;

end process;

-- 垂直方向  OUTPUT
process begin
wait until HORIZ_SYNC'event and HORIZ_SYNC = '1'; -- HORIZ SYNC が変化したら
```

```
case VERT_CNT is                                -- VERT CNT が
when 102 =>                             -- 102 の時
 SY(0) <= '1';                  -- SY(0) に 1 を入れる
 SY(1) <= '1';                  -- SY(1) に 1 を入れる
 SY(5) <= '1';                  -- SY(5) に 1 を入れる
when 110 =>                             -- 110 の時
 SY(0) <= '0';                  -- SY(0) に 0 を入れる
when 178 =>                             -- 178 の時
 SY(1) <= '0';                  -- SY(1) に 0 を入れる
 SY(5) <= '0';                  -- SY(5) に 0 を入れる
              SY(6) <= '1';                 -- SY(6) に 1 を入れる
when 182 =>                             -- 182 の時
 SY(2) <= '1';                  -- SY(2) に 1 を入れる
                  SY(4) <= '1';                 -- SY(4) に 1 を入れる
         when 192 =>                            -- 192 の時
 SY(6) <= '0';                  -- SY(6) に 0 を入れる
when 250 =>                             -- 250 の時
 SY(3) <= '1';                  -- SY(3) に 1 を入れる
when 258 =>                             --250 の時
 SY(2) <= '0';                  -- SY(2) に 0 を入れる
 SY(3) <= '0';                  -- SY(3) に 0 を入れる
                  SY(4) <= '0';                 -- SY(4) に 0 を入れる
  when others =>                          -- それ以外は
null;                           -- ない。

end case;
end process;

RED <= '0';                             -- RED に 0 を入れる
SS(0) <= SX(0) and SY(0);
SS(1) <= SX(1) and SY(1);
SS(2) <= SX(2) and SY(2);
SS(3) <= SX(3) and SY(3);
SS(4) <= SX(4) and SY(4);
SS(5) <= SX(5) and SY(5);
SS(6) <= SX(6) and SY(6);
SS(7) <= SX(7) and SY(7);

TT(0) <= TX(0) and SY(0);
TT(1) <= TX(1) and SY(1);
TT(2) <= TX(2) and SY(2);
TT(3) <= TX(3) and SY(3);
TT(4) <= TX(4) and SY(4);
TT(5) <= TX(5) and SY(5);
TT(6) <= TX(6) and SY(6);
TT(7) <= TX(7) and SY(7);
BLUE <= (SS(0) and HYOUJI1(7)) or
        (SS(1) and HYOUJI1(6)) or
        (SS(2) and HYOUJI1(5)) or
        (SS(3) and HYOUJI1(4)) or
        (SS(4) and HYOUJI1(3)) or
        (SS(5) and HYOUJI1(2)) or
        (SS(6) and HYOUJI1(1)) or
        (SS(7) and HYOUJI1(0)) or
        (TT(0) and HYOUJI(7)) or
        (TT(1) and HYOUJI(6)) or
        (TT(2) and HYOUJI(5)) or
        (TT(3) and HYOUJI(4)) or
        (TT(4) and HYOUJI(3)) or
        (TT(5) and HYOUJI(2)) or
        (TT(6) and HYOUJI(1)) or
        (TT(7) and HYOUJI(0));


HORIZ <= HORIZ_SYNC;                    -- HORIZ に HORIZ SYNC を入れる
VERT <=  VERT_SYNC;                     -- VERT に VERT SYNC を入れる

process begin
wait until CLK'event and CLK ='1';      -- CLK が 変化すると
   CLK_2 <= CLK_2 + 1 ;                 -- CLK2 に 1 を足す
end process;
```

```
DCLK <= CLK_2(22) ;                          -- これで分周を行う


process begin
wait until DCLK'event and DCLK ='1';         -- DCLK が変化すると

if LED = "1001" then                         -- LED が 9 なら
 LED <= "0000" ;                             -- LED を 0 にする
 if LED1 = "1001" then                       -- LED 1 が 9 なら
  LED1 <= "0000" ;                           -- LED 1 を 0 にする
 else                                        --それ以外
  LED1 <= LED1 + "0001" ;                    -- LED 1 に 1 を足す
 end if;

else LED <= LED + "0001" ;                   -- LED に 1 を足す


end if;
end process;

   process (LED) begin

       case LED is
         when "1111" =>
              DIGIT_2 <= "11111111";
              HYOUJI  <= "11111111";
         when "0000" =>
              DIGIT_2 <= "00000011";
              HYOUJI  <= "11111100";
         when "0001" =>                      --1
              DIGIT_2 <= "10011111";
              HYOUJI  <= "01100000";
         when "0010" =>                      --2
              DIGIT_2 <= "00100101";
              HYOUJI  <= "11011010";
         when "0011" =>                      --3
              DIGIT_2 <= "00001101";
              HYOUJI  <= "11110010";
         when "0100" =>                      --4
              DIGIT_2 <= "10011001";
              HYOUJI  <= "01100110";
         when "0101" =>                      --5
              DIGIT_2 <= "01001001";
              HYOUJI  <= "10110110";
         when "0110" =>                      --6
              DIGIT_2 <= "01000001";
              HYOUJI  <= "10111110";
         when "0111" =>                      --7
              DIGIT_2 <= "00011011";
              HYOUJI  <= "11100100";
         when "1000" =>                      --8
              DIGIT_2 <= "00000001";
              HYOUJI  <= "11111110";
         when "1001" =>                      --9
              DIGIT_2 <= "00001001";
              HYOUJI  <= "11110110";
         when others =>
              DIGIT_2 <= "XXXXXXXX";
              HYOUJI  <= "11111111";
       end case;

       case LED1 is
         when "1111" =>
              DIGIT_1 <= "11111111";
              HYOUJI1 <= "11111111";
         when "0000" =>
              DIGIT_1 <= "00000011";
              HYOUJI1 <= "11111100";
         when "0001" =>                    --1
              DIGIT_1 <= "10011111";
              HYOUJI1 <= "01100000";
         when "0010" =>                    --2
              DIGIT_1 <= "00100101";
              HYOUJI1 <= "11011010";
```

```
          when "0011" =>                --3
               DIGIT_1 <= "00001101";
               HYOUJI1 <= "11110010";
          when "0100" =>                --4
               DIGIT_1 <= "10011001";
               HYOUJI1 <= "01100110";
          when "0101" =>                --5
               DIGIT_1 <= "01001001";
               HYOUJI1 <= "10110110";
          when "0110" =>                --6
               DIGIT_1 <= "01000001";
               HYOUJI1 <= "10111110";
          when "0111" =>                --7
               DIGIT_1 <= "00011011";
               HYOUJI1 <= "11100100";
          when "1000" =>                --8
               DIGIT_1 <= "00000001";
               HYOUJI1 <= "11111110";
          when "1001" =>                --9
               DIGIT_1 <= "00001001";
               HYOUJI1 <= "11110110";
          when others =>
               DIGIT_1 <= "XXXXXXXX";
               HYOUJI1 <= "11111111";
        end case;


  end process;
end RTL;
```

A.2 VGA move24*48

```
%
% ここから vga move24*48 のプログラム
%
-----------------------------------
-- VGA Driver degitr   --
-- 01/6/9  25*48bit no hyouji
-- c/m-abe/vga/move24*48/move24*48.vhd
-- m-abe       --
-----------------------------------

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

library metamor;                    -- ライブラリーの呼び出し
use metamor.attributes.all;

entity newVGA5 is
port ( CLK: in std_logic;
       RED   : out std_logic;                  -- color signal red
       GREEN : out std_logic;                  -- color signal green
       BLUE  : out std_logic;                  -- color signal blue
       HORIZ : out std_logic;                  -- synchronize signal horizontal
       VERT  : out std_logic;                  -- synchronize signal vertical
       DIGIT_2 : out std_logic_vector(7 downto 0); -- synchronize signal vertical
       DIGIT_1: out std_logic_vector(7 downto 0); -- synchronize signal vertical
       SW  : in std_logic); -- synchronize signal vertical
attribute pinnum of CLK : signal is "91";
attribute pinnum of RED : signal is  "236";
attribute pinnum of GREEN : signal is  "237";
attribute pinnum of BLUE : signal is  "238";
attribute pinnum of HORIZ : signal is  "240";
attribute pinnum of VERT : signal is  "239";
attribute pinnum of DIGIT_1 : signal is "6,7,8,9,11,12,13,14";
```

```
attribute pinnum of DIGIT_2 : signal is  "17,18,19,20,21,23,24,25";
attribute pinnum of SW : signal is  "28"; -- sw_1

end newVGA5; --franch

architecture RTL of newVGA5 is

signal HORIZ_SYNC : std_logic;
signal HORIZ_CNT : integer range 0 to 800; -- std_logic_vector(9 downto 0);
signal VERT_SYNC : std_logic;
signal VERT_CNT : integer range 0 to 523; --std_logic_vector(10 downto 0);
signal RED1 : std_logic;
signal GREEN1 : std_logic;
signal BLUE1 : std_logic;
signal RED2 : std_logic;
signal RED3 : std_logic;
signal CLK_2 : std_logic_vector(22 downto 0) ;
signal DCLK : std_logic;
signal LED  : std_logic_vector(3 downto 0):= "0000";
signal LED1  : std_logic_vector(3 downto 0):= "0000";
signal GREEN2 : std_logic;
signal BLUE2 : std_logic;
signal HYOUJI : std_logic_vector(7 downto 0);-- kore ha 1 nara hyouji 0 nara hyoujisinai
signal HYOUJI1 : std_logic_vector(7 downto 0);
signal SX : std_logic_vector(7 downto 0);
signal SY : std_logic_vector(7 downto 0);
signal SS : std_logic_vector(7 downto 0);
signal TX : std_logic_vector(47 downto 0);
-- 以下は 24*48 ビットの表示　ここで出力するものを描く
signal TXDAT0 : std_logic_vector(47 downto 0)  :="111111111111111111111111111111111111111111111111";
signal TXDAT1 : std_logic_vector(47 downto 0)  :="110000000000000000000000000000000000000000000011";
signal TXDAT2 : std_logic_vector(47 downto 0)  :="110000000000000000000011000000000000000000000011";
signal TXDAT3 : std_logic_vector(47 downto 0)  :="110000000000000000000011000000000000000000000011";
signal TXDAT4 : std_logic_vector(47 downto 0)  :="110000000011111111111111111111111111110000000011";
signal TXDAT5 : std_logic_vector(47 downto 0)  :="110000000000000000011000000110000000000000000011";
signal TXDAT6 : std_logic_vector(47 downto 0)  :="110000000000000000001100001100000000000000000011";
signal TXDAT7 : std_logic_vector(47 downto 0)  :="110000000000111111001100110011111111000000000011";
signal TXDAT8 : std_logic_vector(47 downto 0)  :="110000000000001000100001100001000100000000000011";
signal TXDAT9 : std_logic_vector(47 downto 0)  :="110000000000001000100001100001000100000000000011";
signal TXDAT10 : std_logic_vector(47 downto 0) :="110000000000110000100001100001010010000000000011";
signal TXDAT11 : std_logic_vector(47 downto 0) :="110000000000000001100001100001100001000000000011";
signal TXDAT12 : std_logic_vector(47 downto 0) :="110000000000000000000000000000000000000000000011";
signal TXDAT13 : std_logic_vector(47 downto 0) :="110000000000010000000000000000000100000000000011";
signal TXDAT14 : std_logic_vector(47 downto 0) :="110000000000010011111111111111110010000000000011";
signal TXDAT15 : std_logic_vector(47 downto 0) :="110000000000010000000000000000000100000000000011";
signal TXDAT16 : std_logic_vector(47 downto 0) :="110000000000010111111111111111110100000000000011";
signal TXDAT17 : std_logic_vector(47 downto 0) :="110000000000010000000001100000000100000000000011";
signal TXDAT18 : std_logic_vector(47 downto 0) :="110000000000010000011001100110000100000000000011";
signal TXDAT19 : std_logic_vector(47 downto 0) :="110000000000010000110001100011000100000000000011";
signal TXDAT20 : std_logic_vector(47 downto 0) :="110000000000010000011000110001100010000000000011";
signal TXDAT21 : std_logic_vector(47 downto 0) :="110000000001000001100001100001100100000000000011";
signal TXDAT22 : std_logic_vector(47 downto 0) :="110000000010000000000011000000001000000000000011";
signal TXDAT23 : std_logic_vector(47 downto 0) :="110000000000000000000000000000000000000000000011";
signal TXDAT24 : std_logic_vector(47 downto 0) :="111111111111111111111111111111111111111111111111";
                                                 -- 123456789012345678901234567890123456789012345678

signal TT : std_logic_vector(47 downto 0);
--signal BLUES : std_logic_vector(7 downto 0);
begin

-- 水平方向の制御
process begin
wait until CLK'event and CLK = '1';
if ( HORIZ_CNT = 799 ) then          -- HORIZ CNT が 799 なら
ば
HORIZ_CNT <= 0;              -- HORIZ CNT を 0 にする
else
HORIZ_CNT <= HORIZ_CNT + 1;  -- それ以外は HORIZ CNT
```

```
に 1 を足す
end if;

end process;

-- suihei houkou
process begin
wait until CLK'event and CLK = '1';   -- クロックが'1' に変化したら
case HORIZ_CNT is
when 0 =>
HORIZ_SYNC <= '0';    -- HORIZ SYNC に '0' を入れる
when 150 =>
HORIZ_SYNC <= '1';    -- HORIZ SYNC に '1' を入れる
when others =>
null;
end case;
end process;

process begin
wait until CLK'event and CLK = '1';  -- クロックが'1' に変化したら

for L in 0 to 47 loop
if HORIZ_CNT = 447-L then
case VERT_CNT is
when 100 =>
TX(L) <= TXDAT0(L);
when 101 =>
TX(L) <= TXDAT1(L);
when 102 =>
TX(L) <= TXDAT2(L);
when 103 =>
TX(L) <= TXDAT3(L);
when 104 =>
TX(L) <= TXDAT4(L);
when 105 =>
TX(L) <= TXDAT5(L);
when 106 =>
TX(L) <= TXDAT6(L);
when 107 =>
TX(L) <= TXDAT7(L);
when 108 =>
TX(L) <= TXDAT8(L);
when 109 =>
TX(L) <= TXDAT9(L);
when 110 =>
TX(L) <= TXDAT10(L);
when 111 =>
TX(L) <= TXDAT11(L);
when 112 =>
TX(L) <= TXDAT12(L);
when 113 =>
TX(L) <= TXDAT13(L);
when 114 =>
TX(L) <= TXDAT14(L);
when 115 =>
TX(L) <= TXDAT15(L);
when 116 =>
TX(L) <= TXDAT16(L);
when 117 =>
TX(L) <= TXDAT17(L);
when 118 =>
TX(L) <= TXDAT18(L);
when 119 =>
TX(L) <= TXDAT19(L);
when 120 =>
TX(L) <= TXDAT20(L);
when 121 =>
TX(L) <= TXDAT21(L);
when 122 =>
TX(L) <= TXDAT22(L);
when 123 =>
TX(L) <= TXDAT23(L);
when 124 =>
TX(L) <= TXDAT24(L);
when others =>
```

```
TX(L) <= '0';
end case;
else
TX(L) <='0';
end if;
end loop;
--end case;
end process;

---- 垂直方向の制御
process begin
wait until HORIZ_SYNC'event and HORIZ_SYNC = '0';
--if ( VSC = 256 ) then
if ( VERT_CNT = 523 ) then
VERT_CNT <= 0;
else
VERT_CNT <= VERT_CNT + 1;
end if;
end process;

-- suityoku houkou
process begin
wait until HORIZ_SYNC'event and HORIZ_SYNC = '1';


case VERT_CNT is                        -- VERT CNT が
when 0 =>                       -- 0 の時
VERT_SYNC <= '0';     -- VERT SYNC に 0 を入れる
when 10 =>                      -- 10 の時
          VERT_SYNC <= '1';     -- VERT SYNC に 1 を入れる
when others =>                  -- それ以外
null;                  -- 無いです

end case;

end process;


TT(47 downto 0) <= TX(47 downto 0);

process begin
wait until CLK'event and CLK = '1';
if --((SS(7 downto 0 ) and HYOUJI1(7 downto 0)) or
    TT(47 downto 0) =  "000000000000000000000000000000000000000000000000" then
BLUE <= '0';          --012345678901234567890123456789012345678901234567
else
BLUE <= '1';
end if;
end process;
HORIZ <= HORIZ_SYNC;
--VERT  <= VERT_SYNC;

end RTL;
```

A.3 PS/2 マウスコントロール

```
%
% ここから マウスコントロールのプログラム
%
-----------------
--mouse clk
--01/12/05 m-abe
--m-abe/mouse/mouse4
-----------------


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 mouse4 is
port (
       CLK : in std_logic;
       MOUSE_CLK : inout std_logic ;
       MOUSE_DATA : in std_logic ;
       SW1 : in std_logic;
       SW2 : in std_logic


      );
attribute pinnum of CLK : signal is "91";
attribute pinnum of MOUSE_CLK : signal is "30";
attribute pinnum of MOUSE_DATA : signal is "31";
attribute pinnum of SW1 : signal is "28";
attribute pinnum of SW2 : signal is "29";

end mouse4 ;

architecture RTL of mouse4 is
signal CLK_2   : std_logic_vector(20 downto 0);
signal DCLK    : std_logic;
signal DCLK2    : std_logic;
signal ENSW1 : std_logic :='1';
signal MOUSEc : std_logic;
signal RS : std_logic :='1';
signal CNT1 : std_logic_vector(6 downto 0);

begin

MOUSE_CLK <= MOUSEc when RS = '0' else 'Z';

process begin
  wait until CLK'event and CLK = '1';
  CLK_2 <= CLK_2+ 1;
end process;

DCLK <= CLK_2(12);
DCLK2 <= CLK_2(20);

process(DCLK, ENSW1, SW2) begin
  if (DCLK'event and DCLK = '1' ) then
--
  if ENSW1 ='0' then
    CNT1 <= CNT1 + "0000001" ;
    else
   CNT1 <= "0000000";
  end if;
--
  end if;
end process;
process (CLK, SW1, ENSW1, CNT1) begin
 if (CLK'event and CLK = '1') then
 if SW1 ='0' and ENSW1 = '1'    then
   RS <= '0';
   ENSW1 <= '0';
   MOUSEc <= DCLK;
 end if ;
 if CNT1 = "0000001" then
  RS <= '1';
 end if;
 if CNT1 = "1000000" then
  ENSW1 <= '1';

 end if;
end if;

end process;
```

```
end RTL;
```

付録 B

回路設計を行う上でのソフトウェアの使用方法

ここでは、本研究において回路設計をする上でのソフトウェアの使用法して、 Accolade 社の PeakFPGA と Altera 社の Max+PLUSII の使用方法と、 FPGA のダウンロード方法について説明する。

B.1 PeakFPGA

B.1.1 VHDL ファイル作成

まず、 VHDL ファイルの作成方法について説明する。

- 最初に PeakVHDL を起動する。

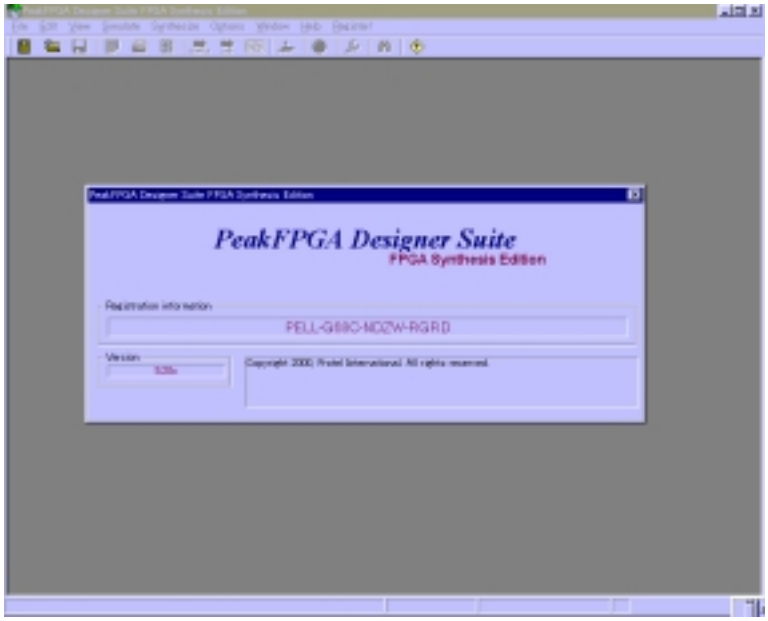


図 2.1: PeakVHDL を開く [1]

[1] ファイル名:u01mabe/zu/peak0.ps

- “File” にある “New Project” を選択

図 2.2: メニューのファイルを選択 [2]

- 次に、”File” にある”New Module” を選択し、ダイヤログボックスが表われたら、”The Project has not been saved. Save it now?” と出てくるので、”OK” を押す。すると、ダイヤログボックスが表われるので ACC(*.acc) ファイルに名前を付けて保存する。この ACC ファイルはプロジェクトのファイルなので、分かりやすい名前を付けておくのが良い。

図 2.3: New Module を選択 [3]

- 次に図 2.4が表われるので、その中の”Cleate Blank Module” を選択する。すると名前を求めてくるので、そこにモジュール名を記述する。これによって VHDL が記述できる VHD(*.vhd) ファイルが作成される。再び追加したいときも、同じ事を繰り返せば良い。

[2]ファイル名:u01mabe/zu/peak1.ps

[3]ファイル名:u01mabe/zu/peak2.ps

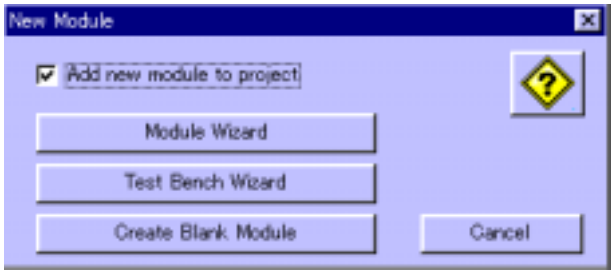


図 2.4: Cleate Blank Module を選択 [4]

- VHDL で記述した VHDL ファイルを構文解析するには、上部のメニューにある”conpile” というボタンを押す。するとコンパイルが始まり、中央にウインドウが表示される。構文に間違いがあったらここにエラーの原因とその行が示されるので、先の VHDL が書かれているウインドウから間違いを直す。

- コンパイルが正しければ、次は論理合成を行う。メニューの”Option” から”Synthesize” を選ぶ。すると、下のウインドウが表われる。ここで、右側にある”Device Family” から”Altera all Device(EDIF)” を選択する。そして、左下の、”Include Synopsys Library” にチェックを付ける。

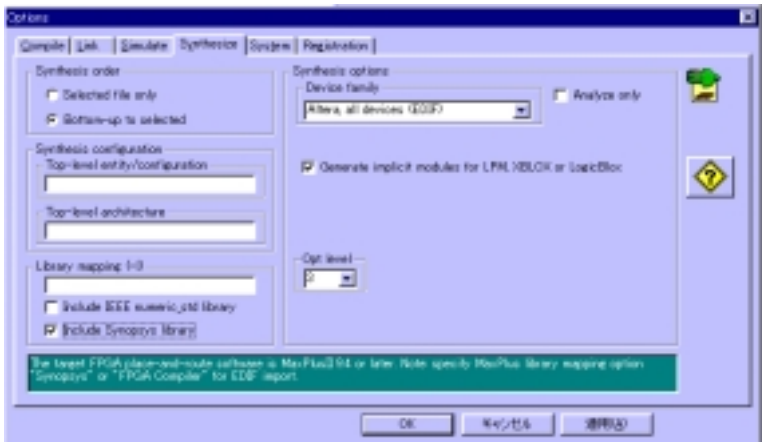


図 2.5: デバイスの選択 [5]

- そして、上部のデバイスマーク、もしくはメニューにある”Synthesize” を選択する。すると、別のウインドウで論理合成が行われる。論理合成が正しく行われたなら、モジュール名のついた EDF ファイルが作成される。また、構文にエラーがあるとコンパイル時と同じようにエラーが表示されるので、 VHD ファイルに戻り間違いを戻す。

[4]ファイル名:u01mabe/zu/peak3.ps

[5]ファイル名:u01mabe/zu/peak4.ps

図 2.6: 論理合成 [6]

- このプロジェクトファイルの中で、 VHDL ファイルを 2 つ以上合成して一つのモジュールを作ることができる。この時、それぞれのファイルは conponent 文または function 文で 2 つのファイルがリンクされていなければならない。そのリンクができている事を確かめるには、”Rebuild Hierarchy” というボタンを押して、リンクし直し”Show Hierarchy” というボタンを押してファイルがリンクできているかを確かめる。 VHDL ファイルを手直ししたときは必ず”Rebuild Hierarchy” を押すことを心掛けた方が良い。

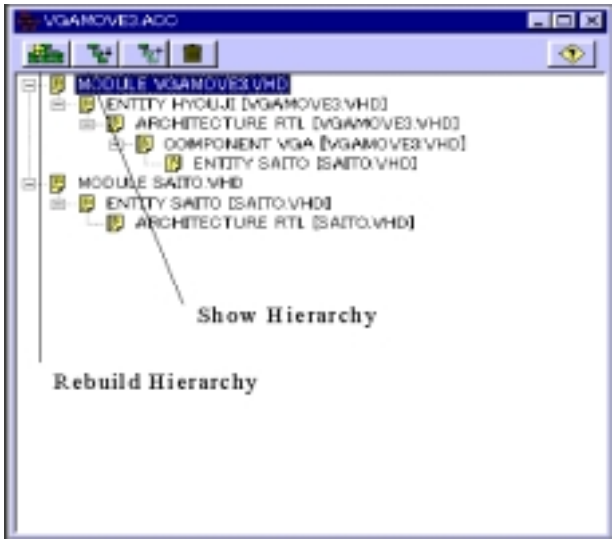


図 2.7: リンクの検査 [7]

B.1.2 VHDL ファイル作成における注意点

本研究で、 PeakVHDL を使用する場合、 VHDL ファイルの作成方法について説明する。まず最初に、

[6] ファイル名:u01mabe/zu/peak5.ps

[7] ファイル名:u01mabe/zu/peak6.ps

```
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;
```

を記述し、 VHDL のライブラリを呼び出す。また、 metamor のライブラリを呼び出すときには、 PeakFPGA で Synopsys library を呼び出すようにしなければならない。

そして、次の entity という部分で、 FPGA の入出力ポートとピンの設定をする。

```
entity mouse is
    port ( CLK : in std_logic;
           MOUSE_CLK   : inout std_logic ;
           MOUSE_DATA  : inout std_logic;
           SW1  : in std_logic
          );

attribute pinnum of CLK         : signal is ‘‘91’’;
attribute pinnum of MOUSE_CLK   : signal is ‘‘30’’;
attribute pinnum of MOUSE_DATA  : signal is ‘‘31’’;
attribute pinnum of SW1         : signal is ‘‘28’’;


end mouse;
```

まず、 entity 文でモジュール名を指定する。これは FPGA にダウンロードするときのファイル名にもなるので、長い名前はつけない方が良い。そして、 port 文で FPGA のポートを指定する。 FPGA にデータを入力する場合には in、データを出力する場合には out、両方の場合には inout を指定する。

そして attribute 文でピン番号を指定する。

```
architecture RTL of mouse is

signal CLK_CNT : std_logic_vector(7 downto 0);
signal COMMAND   : std_logic_vector(0 to 10);

begin

end RTL;
```

次に architecture 文で回路の設計要素を記述する。 count の部分は entity の名前と同じにする。 begin 文の前で、内部信号 signal を宣言し、 begin 文以降は回路の動作を記述する。

B.2 Max+PLUS2

PeakVHDL 上で論理合成が終わったら、次は FPGA に搭載できるファイルに置き換えなければならない。ここでは、その方法について説明する。

- まず、 Max+PLUS2 を起動する。

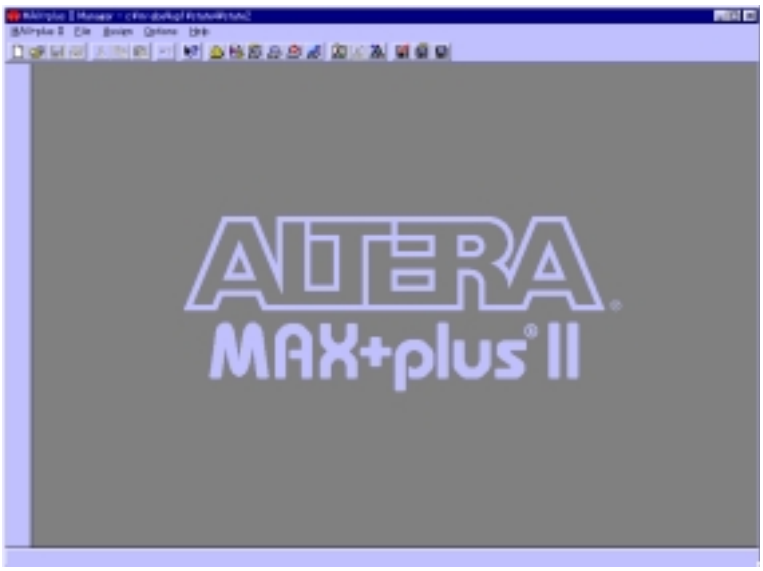


図 2.8: Max+PLUS2 の起動 [8]

- “File” の”Open” を選択し、論理合成した EDF ファイルを選ぶ。

[8]ファイル名:u01mabe/zu/max0.ps

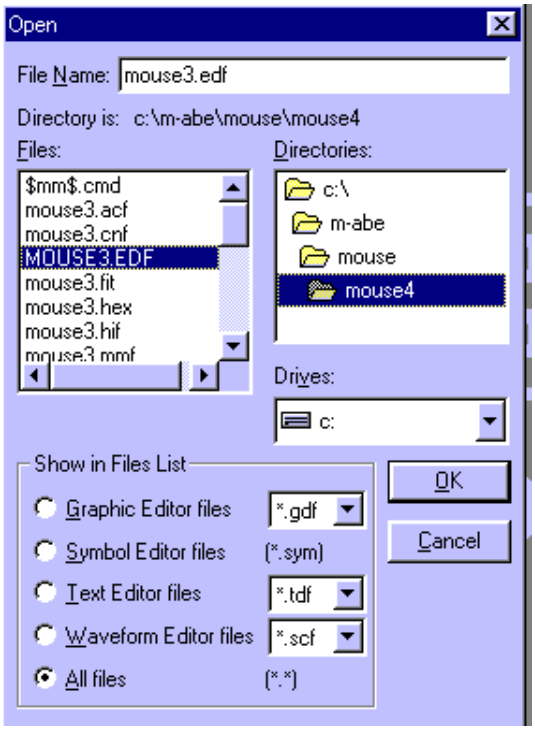


図 2.9: EDF ファイルの選択 [9]

- 次にメニューから”File” の中の”Project” を選択し、”Set Project to Current File” を選んで、 Max+PLUS2 上にセットする。セットされたならば、ウインドウの一番上に選択されたファイルの名前に変換される。

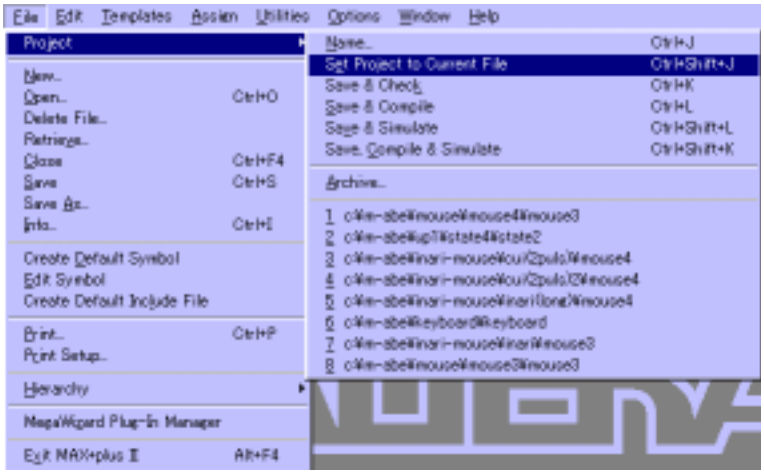


図 2.10: ファイルのセット [10]

- 次に、 “Assign” の “Device” を選択し、ダウンロードする FPGA を選択する。本研究では、EPF10K20RC240-4 を使用しているので、それを選択する。

[9] ファイル名:u01mabe/zu/max1.ps

[10] ファイル名:u01mabe/zu/max2.ps

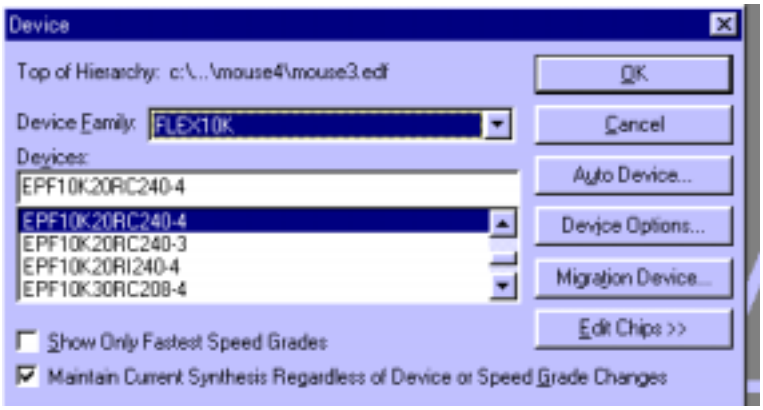


図 2.11: デバイスの選択 [11]

- また、 “Device” 中の ‘’Devaice Options” を選び、チェックをすべて外す。

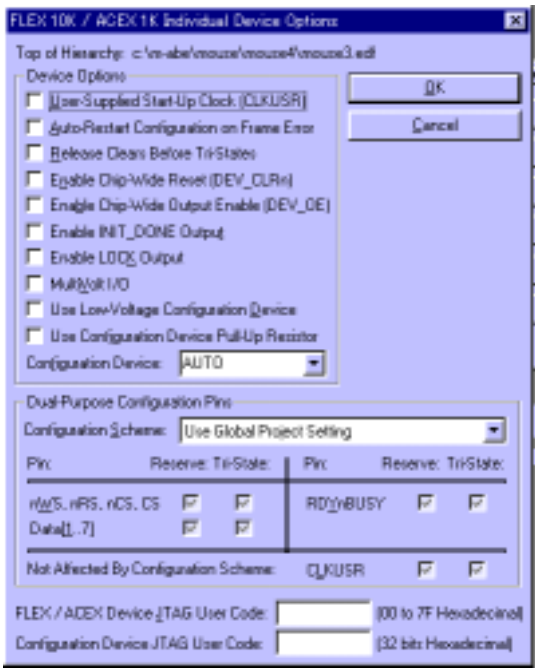
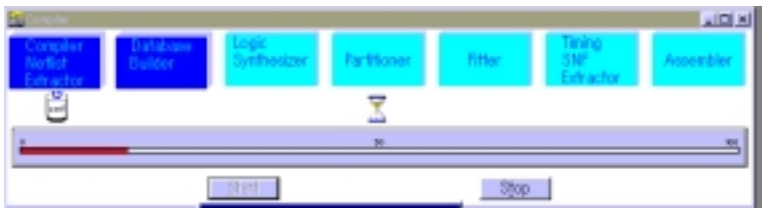


図 2.12: オプション画面 [12]

- 次に、メニューの”Max+PlusII” から”Compiler” を選択する。すると、次のような画面が出てくるので、”Start” を押す。



[11] ファイル名:u01mabe/zu/max3.ps

[12] ファイル名:u01mabe/zu/max4.ps

図 2.13: コンパイル開始 [13]

- コンパイルが終わると、 SOF(*.sof) ファイルが作成されるので、あとはこれを FPGA にダウンロードすれば良い。
  ダウンロードの仕方は、メニューの”Max+PlusII” から”Programmer” を選択すると次のような画面が表われる。

図 2.14: ”Programmer” を選択 [14]

- 次に、メニューから”JTAG” を選択する。すると、次のような画面が表われるので、”Multi-Device JTAG chain” にチェックを付ける。

図 2.15: ”Multi-Device JTAG chain” にチェック [15]

- さらに、”Multi-Device JTAG chain Setup” を選択すると次のような画面が表われるので、”Device Name” と”Programming File Name” を選ぶ。ここでは、デバイスを EPF10k20 に、ファイル名をコンパイル時に作成された SOF(*sof) ファイルを選択する。すべて選択したら右側にある”Add” を押したあと、”OK” を押す。

[13]ファイル名:u01mabe/zu/max5.ps

[14]ファイル名:u01mabe/zu/max8.ps

[15]ファイル名:u01mabe/zu/max6.ps

図 2.16: JTAG セットアップ画面 [16]

- すべての作業が終わったら、図 2.14に戻り”Comfigure” を押す。すると、 UP1 へのダウンロードが開始される。

[16] ファイル名:u01mabe/zu/max7.ps

付録 C

他のボード使用法

VHDL を学ぶにあたって練習用に使ったボードの使用方法、サンプルプログラムをここに示す。

C.1 cq ボード

C.1.1 使用方法

ここでは CQ ボードの使い方について述べる。

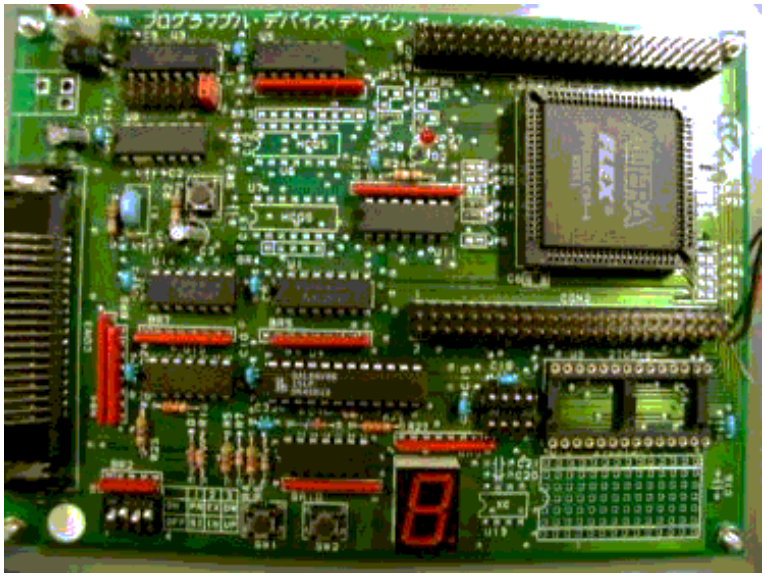


図 3.1: CQ ボード [1]

この CQ ボードは WINDOWS 95 用である。まず、図 3.1の CQ ボードと WINDOWS95 搭載の理論グループ専用の PC を CQ ボード上の FPGA にコンフィグレーションするためプリンターケーブルでつなげる。そしてこの CQ ボードは DC5V 電源を使用する為、電圧源をつなげる。そして、

[1] ファイル名:u01mabe/zu/cq.ps

あらかじめ作っておいた VHDL ソースを MAX+plus2 でコンパイルする。コンパイル法は沼の修士論文の付録に詳しく書いてある。そして DOS プロンプトを開き、目的のソースがあるところに移り、その後 C: flexcq ***.ttf と ttf ファイルを指定し、コンフィグレーションを開始させる。

C.1.2 ピン配線

CQ ボードのピン配線を 3.1に記す。

表 3.1: CQ ボードのピン配線

信号名	用途	方向	ピン数	ピン番号
CLK	クロック供給	in	1	50
SW1	SW1 の信号 負論理が on	in	1	27
SW2	SW2 の信号 負論理が on	in	1	19
LED	LED ランプの表示 正論理が点灯	out	8	a:15,b:16,c:18,d:46, e:35, f:37g:39,h:40

図 3.2: LED ランプ配置図 [2]

C.1.3 サンプルプログラム

以下にサンプルプログラムを添付させる。このプログラムはコンフィグレーションすると、 LED が 1 ずつカウントしていき、 SW を押すと、偶数の数をカウントしていくプログラムである。

```
----------------------------------------
-- 可変速 10 進アップカウンタ (FLEX8000)
-- 2001/03/15
-- m-abe@tube.ee.uec.ac.jp
----------------------------------------
```

[2]ファイル名:u01mabe/zu/7seg.ps

```
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);
signal DCLK    : std_logic;
signal CNT     : std_logic_vector(3 downto 0);
signal CNT_1   : std_logic_vector(3 downto 0):="0000";
signal CNT_2   : std_logic_vector(3 downto 0):="0000";
signal CRY     : std_logic;
signal ST      : std_logic;

begin
process begin
  wait until CLK'event and CLK = '1';-- もしクロックが'1' に変化したら
  CLK_2 <= CLK_2+1;-- クロック 2 に'1' を加えなさい
end process;

DCLK <= CLK_2(20);-- クロックを分周ここでカウントの表示速度を
決定 CLK_2(X) の X の値を小さくすると表示速度が早くなる。

process begin
  wait until DCLK'event and DCLK = '1';-- もし分周したクロックが'1' に変化したら
  case SW_1 is--SW1 が off の時
    when '1' =>--1 の時
      if CNT_1 = "1001" then-- もし CNT1 が 9 になったら
       CNT_1 <= "0000";--CNT1 を 0 に戻してください
       CRY <= not CRY;
      else-- それ以外は
       CNT_1 <= CNT_1 + "0001";--CNT1 に 1 を足して下さい
      end if ;
     when '0' =>--SW1 が on の時
       if CNT_2 = "1000" then-- もし CNT2 が 8 になったら
        CNT_2 <= "0000";--CNT2 を 0 に戻してください
        CRY <= not CRY;
       else-- それ以外は
        CNT_2 <= CNT_2 + "0010";--CNT2 に 1 を足して下さい
       end if ;
      when others => null;
   end case;
  end process;

  process (SW_1) begin
   if ( SW_1 = '1' ) then-- もし SW1 が off なら
      ST <= '0';--ST に'0' を入れる
   elsif (SW_1 = '0' ) then-- もし SW1 が on なら
      ST <= '1';--ST に'1' を入れる
   end if;
  end process ;
    CNT <= CNT_1 when ST = '0' else--CNT を SW によって選択
           CNT_2;
```

```
process ( CNT ) begin
  case CNT is--LEDを光らせる  '0'で点灯  '1'で消灯
      when "0000" => LED <= "01111110";--0を表示
      when "0001" => LED <= "00110000";--1を表示
      when "0010" => LED <= "01101101";--2を表示
      when "0011" => LED <= "01111001";--3を表示
      when "0100" => LED <= "00110011";--4を表示
      when "0101" => LED <= "01011011";--5を表示
      when "0110" => LED <= "00011111";--6を表示
      when "0111" => LED <= "01110010";--7を表示
      when "1000" => LED <= "01111111";--8を表示
      when "1001" => LED <= "01111011";--9を表示
      when others => LED <= "XXXXXXXX";
  end case;
 end process;
end RTL;
```

C.2 UP1 ボード

C.2.1 UP1 ボードの設定方法

ここでは、 UP1 ボードの使い方について述べる。

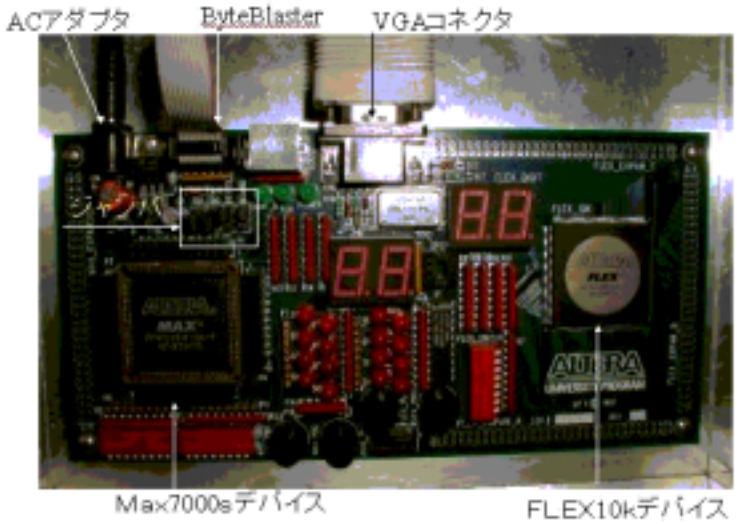


図 3.3: UP1 ボード [3]

UP1 ボードには二つの FPGA(FLEX10k,MAX7000s) が搭載されている。
このボードを動作させるため設定方法についての説明をします。

- UP1 ボードに電源を入れるために、 AC アダプタを使用する。この AC アダプタの入力は、UP1 ボードの左上にある DC-IN から行う。

[3]ファイル名:u01mabe/zu/up.ps

- 次に、ジャンパの設定です。
  このジャンパは、UP1 ボードの二つの FPGA を、使用する FPGA の種類の応じて変える必要がある。図 3.4にジャンパの設定を示す。

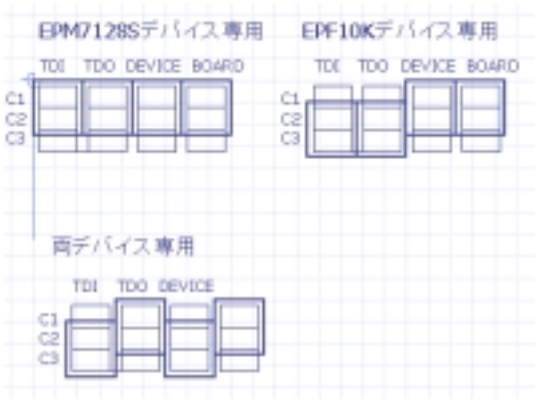


図 3.4: ジャンパの設定 [4]

C.2.2 ピン配線

UP1 ボードに割り当てられている FPGA のピン配線を次に記す。

表 3.2: UP1 ボードのピン配線

信号名	用途	方向	ピン数	ピン番号
CLK	クロック供給	in	1	91
SW1	FLEX_PB1 の信号 負論理が on	in	1	28
SW2	FLEX_PB2 の信号 負論理が on	in	1	29
F_SW	FLEX_SWITCH の信号 上に上げると on	in	8	41,40,39,38,36,35,34,33
LED_1	LED ランプの表示, 負論理が点灯	out	8	a:17,b:18,c:19,d:20,e:21, f:23, g:24,DecimalPoint:25

[4]ファイル名:u01mabe/zu/jump.ps

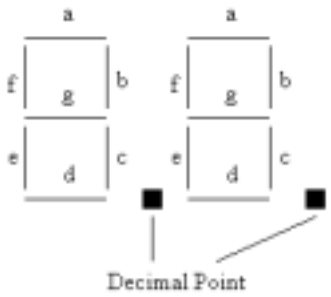


図 3.5: LED の配置図 [5]

C.2.3 サンプルプログラム

以下にサンプルプログラムを添付する。このプログラムは UP1 ボード上にある 2 つの LED を 00 から 99 まで 1 ずつ光らせるプログラムである。

```
------------------------------------------
-- 可変速 2 桁 10 進アップカウンタ (UP1 ボード)
-- 2001/03/18
-- m-abe@tube.ee.uec.ac.jp
------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library metamor ;
use metamor.attributes.all;
entity countup is
 port (
        SW_1,SW_2,CLK : in std_logic;
        LED_1,LED_2 : out std_logic_vector ( 7 downto 0 )
       ) ;

 attribute pinnum of LED_1  : signal is "6,7,8,9,11,12,13,14";
 attribute pinnum of LED_2  : signal is "17,18,19,20,21,23,24,25";
 attribute pinnum of CLK     : signal is "91";
 attribute pinnum of SW_1   : signal is "28";
 attribute pinnum of SW_2   : signal is "29";

end countup;

architecture RTL of countup is

 signal CLK_2 : std_logic_vector ( 20 downto 0 );
 signal DCLK  : std_logic;
 signal CNT_1 : std_logic_vector (3 downto 0 ) := "0000";
 signal CNT_2 : std_logic_vector (3 downto 0 ) := "0000";
begin
 process begin
   wait until CLK'event and CLK = '1';-- クロックが'1' になった瞬間
   CLK_2 <= CLK_2 + '1';-- クロック 2 に'1' を足す
 end process;

 DCLK <= CLK_2(20);-- クロックの分周　表示速度を決定

process begin

process begin
  wait until DCLK'event and DCLK = '1' ;-- 分周したクロックが'1' になったら
  if CNT_1 = "1001" and CNT_2 = "1001" then-- もし CNT_1 が 9 かつ CNT_2 が 9 つまり 99 になったら
     CNT_1 <= "0000";--0 に戻す
```

[5] ファイル名:u01mabe/zu/led.ps

```
    CNT_2 <= "0000";--0 に戻す
  elsif CNT_2 = "1001" then-- もし CNT_2 が 9 つまり 1 の位が 9 になったら
    CNT_1 <= CNT_1 + "0001";--CNT_2 に 1 追加つまり 10 の位に 1 つ繰り上げ
    CNT_2 <= "0000";--CNT_2 を 0 にするつまり 1 の位を 0 の戻す
  else-- それ以外は
    CNT_2 <= CNT_2 + "0001";--CNT_2 に 1 を足すつまり 1 ずつカウントしていく
  end if;
 end process;

 process  begin  --'0' が点灯  '1' が消灯
       case CNT_1 is-- 1 の位が
           when "0000" => LED_1 <= "00000011";--0 を表示する
           when "0001" => LED_1 <= "10011111";--1 を表示する
           when "0010" => LED_1 <= "00100101";--2 を表示する
           when "0011" => LED_1 <= "00001101";--3 を表示する
           when "0100" => LED_1 <= "10011001";--4 を表示する
           when "0101" => LED_1 <= "01001001";--5 を表示する
           when "0110" => LED_1 <= "01000001";--6 を表示する
           when "0111" => LED_1 <= "00011011";--7 を表示する
           when "1000" => LED_1 <= "00000001";--8 を表示する
           when "1001" => LED_1 <= "00001001";--9 を表示する
           when others  => LED_1 <= "XXXXXXXX";
         end case;
  case CNT_2 is-- 1 0 の位が
           when "0000" => LED_2 <= "00000011";--0 を表示する
           when "0001" => LED_2 <= "10011111";--1 を表示する
           when "0010" => LED_2 <= "00100101";--2 を表示する
           when "0011" => LED_2 <= "00001101";--3 を表示する
           when "0100" => LED_2 <= "10011001";--4 を表示する
           when "0101" => LED_2 <= "01001001";--5 を表示する
           when "0110" => LED_2 <= "01000001";--6 を表示する
           when "0111" => LED_2 <= "00011011";--7 を表示する
           when "1000" => LED_2 <= "00000001";--8 を表示する
           when "1001" => LED_2 <= "00001001";--9 を表示する
           when others  => LED_2 <= "XXXXXXXX";
         end case;
 end process;
end RTL;
```

付録 D

PS/2 キーボードのコントロールシステムの設計

PS/2 マウスのコントロールシステムの設計の他に、 PS/2 キーボードのコントロールシステムの設計も行った。ここでは、 PS/2 キーボードの交信アルゴリズムを説明し、キーボード上のボタンを押すと、 UP1 ボード上の 7 セグメントディスプレイに何らかの反応を示すという事を行った。

D.1 PS/2 キーボードの構造

本研究で使用するキーボードは、一般的に普及している PS/2 キーボードを使用した。 PS/2 キーボードは、 6 ピンの DIN コネクタを採用していて PC とキーボードの間は データ線、クロック線、 5V、 GND の 4 線インターフェイスで接続されている。余りの 2 本は予約となっている。図 4.1にコネクタとピン配置を示す。データ線とクロック線の信号線は双方向に通信できるようになっていて、キーボードから押されたキーコードを受け取るだけでなく、システムからキーボードへ制御命令を送ることで、キーボードについている LED を点灯したり、キーボードの試験をしたり、動作を変更したりすることも可能である。また、キーボードは 106 個のキー配置が設置されている。 [5]

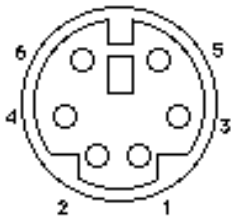


図 4.1: PS/2 キーボード・コネクタ [1]

[1] ファイル名:u01mabe/zu/key0.ps

表 3.1 コネクタのピン割り当て

ピン	I/O	信号名
1	I/O	データ
2	-	Reserved
3	-	GND
4	-	+5V
5	I/O	クロック
6	-	Reserved

本研究で使用するキーボードは、一般的に使用されている日本語版のキーボードを使用した。

D.2 PS/2 キーボードの通信アルゴリズム

キーボードと PC との通信に使用される信号は、データとクロックである。二つの信号は入出力であるが、 PC とキーボードが同時に信号を出しても壊れることはない。また、二つの信号は出力されていないときは常に High レベルに保られている。信号レベルは Low レベルが 0.7V 以下であり、 High レベルが 2.4V 以上である。 PC は DATA 線を使ってキーボードにコマンドを書き込んだり、キーボードのデータを読み込んだりする。たとえば、 PC からキーボードへの転送の時、キーボード側で DATA の出力を High レベルにする。また、キーボードから PC への転送時は、 PC 側で DATA の出力を High レベルにする。ただし、 PC 側がクロックの出力を強制的に Low にすることによって、キーボードからの出力を禁止したり、データの転送を中止する事ができる。

PC とキーボードとの通信でデータ転送時に重要な信号はクロックである。キーボードのクロックはマウスのクロックと同じ、 11 ビットを 1 クロックとしていてスタートビット、奇数パリティビット、エンドビットから成り立っている。 [4]

- PC からキーボードへの送信

  PC からキーボードへの送信は、キーボードに対するコマンドの転送である。キーボードのコマンドには、キーボードの動作モードの設定や、初期状態へのリセット、データの再送信要求などのコマンドがある。各コマンドは 16 進数で 表され、マウスのコマンドと同じ様な動作を示す。次にキーボードコマンドの一覧と説明を示す。

  - Default Disable(F5)

    電源が入るとデフォルト状態にリセットし、また、走査停止して、次のコマンドを待つ。

- Echo(EE)
  キーボードの状態を診断するコマンドである。
- Enable(F4)
  このコマンドで出力バッファとタイプマティックを停止して、データ転送を開始する。
- Resend(FE)
  PC はこのコマンドを受けると、キーボードにデータの再転送を要求する。
- Reset(FF)
  キーボードをデフォルト状態にリセットし、キーボードの状態もチェックする。
- Set All Key(F7)
  すべてのキーをタイプマティックにセットする。
- Set All Key(F8)
  すべてのキーをメイク or ブレイクにセットする。
- Set All Key(F9)
  すべてのキーをメイクにセットする。
- Set All Key(FA)
  すべてのキーをタイプマティック or メイク or ブレイクにセットする。
- Set Default(F6)
  電源が入るとキーボードをデフォルト状態にセットする。
- Set Key type(FB)
  個別のキーをタイプマティックにセットする。
- Set Key type(FC)
  個別のキーをメイクにセットにする。
- Set Reset status (FD)
  キーボード上の NumLock CapsLock ScrollLock の LED の ON,OFF を行う。
- セット・タイプマティック・レート、ディレイ (F3)
  タイプマティックのレートとディレイ設定を行う。

PC 側は、いつでもキーボード側にコマンドを転送することができる。また、キーボードから送信中であっても、キーボード CLK の 10 クロック目に入る前であれば CLK を Low レベルにして転送を中止する事ができる。ただし、 PC が送信を行うためにはキーボード側

から CLK を出力してもらわなければならないので、実際には送信要求の状態である DATA が Low CLK が High の状態にしなければならない。また、キーボードは、 10ms 以下の間隔でその状態を調べて送信要求を検出したらそれに応じなければならない。

- キーボードから PC への送信
  キーボードから PC への送信は、コマンドの場合とデータの場合がある。コマンドというのは、 PC からのコマンドに対する応答の意味を持っている。また、データというのはキーを押したときにの状態をスキャンして得られるスキャンコードである。スキャンコードとは、キーボードにある各キーに対して一意に割り当てられているものであり、キーボードのキー構成が変われば定義も変わるものである。スキャンコードには 1-3 種類のコードが用意されていてどれを使うかは、 PC からのキーボードへのコマンドで指定できる。
  また、キーボード側では、 SHIFT+A や CTRL+C というような複数のキーの組み合わせの認識処理は行わない。キーボードは、 [SHIFT キーが押された] -¿ [A キーが押された] -¿[A キーが離された] -¿[SHIFT キーが離された] などというように、個々のキーの状態の変化を PC に転送する。これらの作業をキーボード側が [SHIFT キーが押されている間に A キーが押された] というように判定して [SHIFT+A］ として認識する。
  キーボードも PC 側と同じようにいつでも PC に対してコマンドやデータを送信できる。キーボード側には転送を中止する権限はないので、 DATA および CLK が今どういう状態かを調べて、転送中でないときだけ CLK を Low レベルにして転送を開始する。

D.3 結果と考察

キーボード上のボタンを押すと、 UP1 ボード上の 7 セグメントディスプレイが光った。しかし、キーボードを押し続けていないと光っているのか分かりにくい。また、現段階では、 A のボタンを押すとディスプレイに A を表示するといった個別のデータをまだ認識できていない。
今後の課題としては、まずデータを個別に認識できるようにすることが望ましい。その後、キーボードを自由に制御し VGA モニターへの出力を行ったらどうだろうか。

D.4 PS/2 キーボードコントロール

```
%
% ここから キーボードコントロールのプログラム
%
-----------------
--keyboard
--01/12/28 m-abe
--m-abe/key/keyboard2.vhd
-----------------
library IEEE;                       -- ライブラリの呼び出し
use IEEE.std_logic_1164.all;        -- ライブラリの呼び出し
use IEEE.std_logic_arith.all;       -- ライブラリの呼び出し
use IEEE.std_logic_unsigned.all;    -- ライブラリの呼び出し

library metamor;                    -- ライブラリの呼び出し
use metamor.attributes.all;         -- ライブラリの呼び出し

entity keyboard is                     -- ここで、入出力ポート
とピンの設定を行う。
port  (keydata :inout std_logic;
       keyclock : inout std_logic ;
       CLK : in std_logic ;
       DATOUT : out std_logic_vector(7 downto 0); --simulation
       LED_1: out std_logic_vector(7 downto 0); --synchronize signal vertical
       LED_2: out std_logic_vector(7 downto 0);
       SW_1 : in std_logic);

attribute pinnum of keydata: signal is "31";
attribute pinnum of keyclock : signal is "30";
attribute pinnum of LED_1 : signal is "6,7,8,9,11,12,13,14";
attribute pinnum of LED_2 : signal is "17,18,19,20,21,23,24,25";
attribute pinnum of CLK: signal is "91";
attribute pinnum of SW_1: signal is "28";
end keyboard;

architecture RTL of keyboard is
signal start_bit :std_logic;

signal DATA : std_logic_vector(9 downto 0);
signal CNT : std_logic_vector(3 downto 0);
signal break_code: std_logic;

signal DDD :std_logic_vector(7 downto 0);
signal keyclock_filter: std_logic;
signal filter : std_logic_vector(7 downto 0);
--signal CNT_filter : std_logic_vector(1 downto 0) := "00";
begin

keyclock <= CLK;          クロックをキーボードのクロックに代
入

 process

  begin
wait until keyclock'event and keyclock = '1';
if break_code = '0' then
 if keydata = '0' and start_bit = '0' then
     start_bit <= '1';
     CNT <= "0000";
     DATA <= "0000000000";
 else if start_bit = '1' then
     CNT <= CNT + 1;

  case CNT(3 downto 0) is
      when "0001" => DATA(0) <= keydata;
      when "0010" => DATA(1) <= keydata;
      when "0011" => DATA(2) <= keydata;
```

```
      when "0100" => DATA(3) <= keydata;
      when "0101" => DATA(4) <= keydata;
      when "0110" => DATA(5) <= keydata;
      when "0111" => DATA(6) <= keydata;
      when "1000" => DATA(7) <= keydata;
      when "1001" => DATA(8) <= keydata;
      when others  => DATA(9) <= keydata;
                  start_bit <= '0';

   end case;
  end if;
 end if;
 elsif break_code = '1' then
  if keydata = '0' and start_bit = '0' then
     start_bit <= '1';
     CNT <= "0000";
     DATA <= "0000000000";
     else if start_bit = '1' then
     CNT <= CNT + 1;
       case CNT(3 downto 0) is
        when "0001" => DATA(0) <= '1';
        when "0010" => DATA(1) <= '1';
        when "0011" => DATA(2) <= '1';
        when "0100" => DATA(3) <= '1';
        when "0101" => DATA(4) <= '1';
        when "0110" => DATA(5) <= '1';
        when "0111" => DATA(6) <= '1';
        when "1000" => DATA(7) <= '1';
        when "1001" => DATA(8) <= '1';
        when others  => DATA(9) <= '1';
                  start_bit <= '0';

      end case;
    end if;
  end if;
 end if;
  DDD(7 downto 0) <= DATA(7 downto 0);
end process;

 process (DDD(7 downto 0)) begin
  DATOUT <=DDD;

        case DDD(7 downto 0) is
           when "00010101" => LED_1 <= "10101010"; --15 ->(17)q
           when "00011111" => LED_1 <= "10101010"; --1d ->(18)w
           when "00100100" => LED_1 <= "10101010"; --24 ->(19)e
           when "00101101" => LED_1 <= "10101010"; --2d ->(20)r
           when "00101100" => LED_1 <= "10101010"; --2c ->(21)t
           when "00110101" => LED_1 <= "10101010"; --35 ->(22)y
           when "00111100" => LED_1 <= "10101010"; --3c ->(23)u
           when "01000011" => LED_1 <= "10101010"; --43 ->(24)i
           when "01000100" => LED_1 <= "10101010"; --44 ->(25)o
           when "01001101" => LED_1 <= "10101010"; --4d ->(26)p
           when "00101011" => LED_1 <= "10101010"; --2b ->(34)f
           when "00110100" => LED_1 <= "10101010"; --34 ->(35)g
           when "00110011" => LED_1 <= "10101010"; --33 ->(36)h
           when "00111011" => LED_1 <= "10101010"; --3b ->(37)j
           when "01000010" => LED_1 <= "10101010"; --42 ->(38)k
           when "01001011" => LED_1 <= "10101010"; --4b ->(39)l
           when "00011010" => LED_1 <= "10101010"; --1a->(46)z
           when "00100010" => LED_1 <= "10101010"; --22 ->(47)x
           when "00100001" => LED_1 <= "10101010"; --21 ->(48)c
           when "00101010" => LED_1 <= "10101010"; --2a ->(49)v
           when "00011100" => LED_1 <= "10101010"; --1c ->(31)a
           when "00011011" => LED_1 <= "01010101";
                              LED_2 <= "11111110"; --1b ->(32)s
           when "00100011" => LED_1 <= "01010101"; --23 ->(33)d
           when "00110010" => LED_1 <= "10101010"; --32 ->(50)b
           when "00110001" => LED_1 <= "10101010"; --31 ->(51)n
           when "00111010" => LED_1 <= "10101010"; --3a ->(52)m
           when "11110000" => break_code <= '1';
           when "11111111" => break_code <= '0';
           when others =>LED_1 <= "10000001";
        end case;
```

```
 end process;
end RTL;
```