実験「計算機の設計」における FPGA 使用法

1　はじめに

実験「計算機の設計」において、設計した CPU を FPGA 上に実装し、動作検証を行うための方法を解説します。

今回、動作検証を行うための方法として、プログラム終了後、メモリの特定のアドレスの内容を FPGA ボード上の 7 セグメント LED に表示させるという方法を考えました。

2　手順

CPU の設計を行なった後に、FPGA 上に実装するための手順は以下のようになります。

1. ダウンロードケーブルのドライバインストール
2. 動作検証のための CPU の改変、モジュールの追加
3. ピンアサインの設定
4. FPGA にダウンロード

3　手順解説

以下、それぞれの手順について解説します。

3.1　ダウンロードケーブルのドライバインストール

以下の URL を参照して、ドライバをインストールしてください。
`http://altimanet.com/pdf/altera/tool/driver_bb_v20.pdf`

3.2　動作検証のための CPU の改変、モジュールの追加

動作検証の際にいくつかの CPU の改変、およびモジュールの追加が必要となります。
以下のような改変が必要となります。

- run 信号を自動的に与える
- wait 状態でデータバスにメモリの内容を出力する
- 出力データを 7 セグメント LED で表示するためのデコーダおよび LED 表示の切り換えを行うモジュールの作成

これより、それぞれの改変、追加方法を示していきます。

run 信号を自動的に与える

まず、run 信号を自動的に与えるように改変します。

当初、ボード上のスイッチから run 信号を与えようとしたのですが、run 信号が長く入りすぎ、思うように動作しませんでした。そのため、このように改変を行いました。

改変部分

入力の run をコメントアウトし、run を NODE として定義する。

```
run:NODE;
```

そして、run が 0 の時にプログラムが開始するように stage_d の入力を run の反転とする。
そして、スイッチから reset を与えるため、各モジュールの reset も reset の反転とする。（スイッチを押した際に入力が 0 となるため）

```
stage.(clk,reset,run) = (clk,!reset,!run);
pc.(clk,reset) = (clk,!reset);
reg.(clk,reset) = (clk,!reset);
alu.(clk,reset) = (clk,!reset);
```

run 信号を読み取ってプログラムが開始したところで run が 1 となり、以降ずっと 1 になっているように以下の記述を加える。

```
IF run == B"0" & fetcha THEN
run = B"1";
ELSIF run == B"1" THEN
run = B"1";
END IF;
```

wait 状態でデータバスにメモリの内容を出力する

プログラムが終了し、wait 状態になった際にメモリの値が読み出せるように改変を行います。

改変部分

wait 状態時にアドレスを切り換えるために、新たな制御信号を NODE として定義する。
ここでは、新たな制御信号を led2ad とする。

```
led2ad : NODE;

led2ad = GND;
```

そして、wait 状態でメモリからデータを読み出すことができるように以下の記述を加える。

```
IF wait THEN
ram2data = B"1";
led2ad = B"1";
END IF;
```

モジュールの追加

今回使用した FPGA ボードには 6 個の 7 セグメント LED があり、それぞれの LED に対応する信号が 0 のとき、セグメントデータに対応する数字が表示されます。

そのため、一定のクロックごとに LED の選択信号とセグメントパターンを切り換えることでメモリの内容を 7 セグメント LED に表示させるモジュールを作成しました。

追加モジュールは入力としてクロック、リセット、データバスのデータが与えられ、セグメントパターン、7 セグメント LED の選択信号、読み出したいメモリのアドレスを出力します。

追加モジュール 1/2

```
--8 ビットカウンタをクロック数計測のため再利用--
INCLUDE "counter_d";
-----------------------------------------------
SUBDESIGN decode_d
(
clk,reset : INPUT;
in[7..0] : INPUT;
out[7..0],sel[5..0],ad[7..0] : OUTPUT;--それぞれセグメントパターン、選択信号、アドレス--
)
VARIABLE
counter : counter_d;
r[5..0] : DFF;--選択信号保持用のレジスタとして使用--
BEGIN
counter.(clk,reset) = (clk,reset);
r[].(clk,clrn) = (clk,!reset);
sel[] = r[].q;

--256 クロックごとに点灯する LED を切り換える--
IF counter.out[] == B"11111111" THEN
CASE r[].q IS
WHEN B"000000" => r[].d = B"011111";
WHEN B"011111" => r[].d = B"101111";
WHEN B"101111" => r[].d = B"110111";
WHEN B"110111" => r[].d = B"111011";
WHEN B"111011" => r[].d = B"011111";
END CASE;
ELSE
r[].d = r[].q;
END IF;
-----------------------------------------------
```

次項に続く

追加モジュール 2/2

```
--7 セグメント LED デコーダ部分。4 ビットのデータを一つの 7 セグメント LED で表示する。--
IF r[].q == B"101111" # r[].q == B"111011" THEN
CASE in[7..4] IS
WHEN 0 => out[] = B"00111111";
WHEN 1 => out[] = B"00000110";
WHEN 2 => out[] = B"01011011";
WHEN 3 => out[] = B"01001111";
WHEN 4 => out[] = B"01100110";
WHEN 5 => out[] = B"01101101";
WHEN 6 => out[] = B"01111101";
WHEN 7 => out[] = B"00000111";
WHEN 8 => out[] = B"01111111";
WHEN 9 => out[] = B"01101111";
WHEN 10 => out[] = B"01110111";
WHEN 11 => out[] = B"01111100";
WHEN 12 => out[] = B"01011000";
WHEN 13 => out[] = B"01011110";
WHEN 14 => out[] = B"01111001";
WHEN 15 => out[] = B"01110001";
END CASE;
END IF;

IF r[].q == B"011111" # r[].q == B"110111" THEN
CASE in[3..0] IS
上記と同内容のため省略
END CASE;
END IF;
-----------------------------------------------------------------
--読み出すアドレスの切り換え--
IF r[].q == B"011111" # r[].q == B"101111" THEN
ad[] = B"00101001";--読み出したいアドレス 1--
END IF;
IF r[].q == B"110111" # r[].q == B"111011" THEN
ad[] = B"00101000";--読み出したいアドレス 2--
END IF;
------------------------------
END;
```

次に、このモジュールを導入するための CPU の改変部分を示します。

改変部分

まず、作成したモジュール追加する。

```
INCLUDE "decode_d.inc";

decode : decode_d;
```

次に、セグメントパターンと選択信号を出力として追加し、アドレスとデータの出力は使用しないのでコメントアウトしておく。

```
--address_out[7..0] : OUTPUT;
--data_out[7..0] : BIDIR;
led_out[7..0],led_sel[5..0] : OUTPUT;
```

そして、追加モジュールのポートの接続と、不要な部分のコメントアウトを行う。

```
--address_out[] = address[];
--data_out[] = data[];
address[] = decode.ad[] & led2ad;
decode.in[] = data[];
led_out[] = decode.out[];
led_sel[] = decode.sel[];
```

追加、改変部分は以上です。

3.3　ピンアサインの設定

次に、ピンアサインの設定を行います。

作成した CPU をコンパイルすると、適当なピンにアサインが行われます。ここでは、出力を決まったピンにするための設定方法を示します。

まず、コンパイルを行なった後に、トップメニューの Max+plusII → Floorplan Editer を選択し、Floorplan Editer を起動してください。最後にコンパイルを行なった際のアサイン設定が表示されます。ここで、図 1 の位置をクリックすると、現在のアサイン設定が表示されます。まだアサイン設定を行っていないため、すべての部分が白く表示されます。

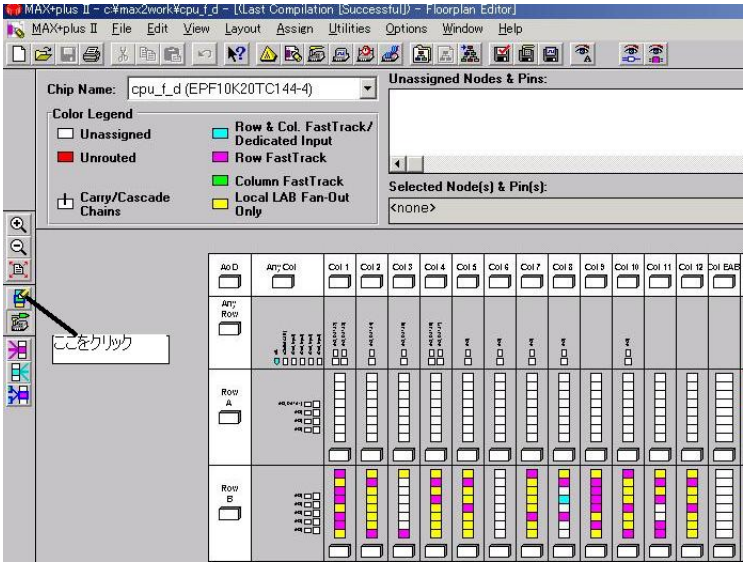

図 1: Floorplan Editer

次に、トップメニューの Assign → Pin/Location/Chip 選択します。すると、図 2 のように表示され、ピンアサインの設定ができます。

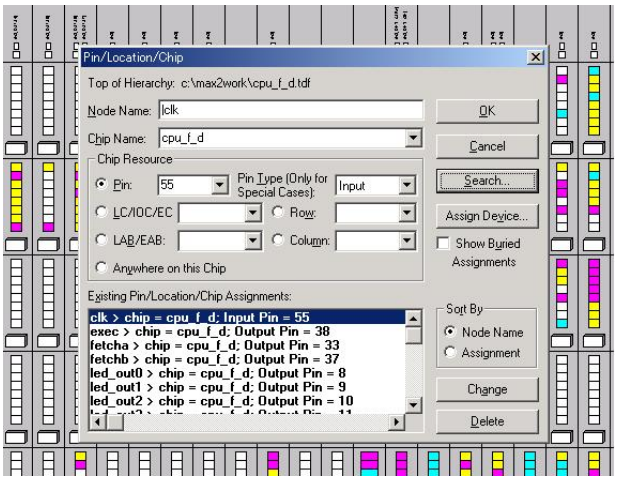

図 2: ピンアサインの設定

まず、Search でアサイン設定したいノードを探し、アサインしたいピン番号を選んで Add ボタンをクリックしてください。

今回使った主なピン番号を表 1 に示します。詳しくは、FPGA ボードの取扱説明書を見てください。

表 1: ピン番号

クロック	55
押しボタンスイッチ	39,41,42,43
7 セグメント LED 選択信号	19,20,21,22,23,26
セグメントパターン	8,9,10,11,12,13,17,18
LED	32,33,37,38

ピンアサインの設定を行なった後、再びコンパイルを行うことで設定が反映されます。

3.4 FPGA にダウンロード

ピンアサイン後、コンパイルを行ったら、トップメニューの Max+plusII → Programmer を選択し、Programmer を起動してください。Programmer を起動したら、FPGA ボードとコンピュータの接続、ボードの電源を確認してから、トップメニューの Options → Hardware Setup を選択してください。そして、図 3 のように Hardware Type に ByteBruster[MV] を選択し、OK をクリックしてください。

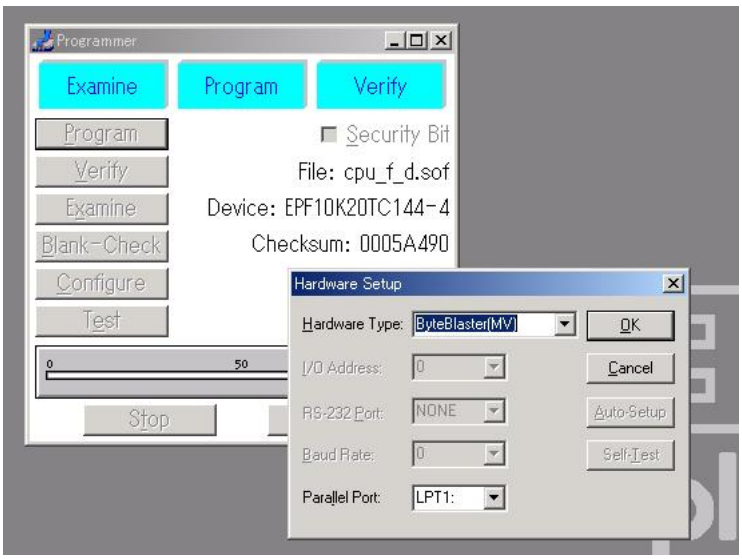

図 3: Hardware Setup

最後に Programmer の Configure ボタンをクリックすれば、FPGA 上に CPU が作成され、7 セグメント LED にメモリの内容が出力されます。