SoC Linux 道場【其ノ八】

ネットワーク・アプリでの遠隔 PWM 制御、モーター制御、起動スクリプトの作成方法

ver.13.1

2015 年 3 月 Rev.1

目次

1. はじめに

今回（最終回）の 2 章では、ソケット・サーバとクライアント・プログラムを自身で作成して、ソケット通信を使用した遠隔 PWM 制御を行ってみます。

3 章では、Helio ボードの HSMC コネクタに、モーター・ドライブ回路を接続した独自の拡張ボードを接続して、実際のモーターを使って PWM 制御を行うための概念について説明します。

4 章では、追加したサービスやあらかじめ起動しておきたいプログラムを Helio ボードの起動時に実行させるシェル・スクリプト・ファイルの作成方法を紹介します。

【注記】

※1. **2-2 項** は、Android クライアント・アプリがないため、実際の動作確認は行わず、概念の説明だけになります。

※2. **3 章** は、現在、拡張ボードの回路仕様が公開されていないため、実際のモーター制御の動作確認は行わず、概念の説明だけになります。

鳥海氏による「SoC FPGA 組み込み Linux 2 日間集中セミナー」では、実際に拡張ボードを使用したモーター制御を体験することができます。

尚、この資料の説明で使用している主な開発環境は以下の通りです。

【表 1.1】この資料の説明で使用している主な開発環境

項番	項目	内容
1	ホスト OS	Microsoft® Windows® 7 Professional sp1 日本語版（64 bit）
2	ゲスト OS	Vine Linux 6.2.1 i686（32bit） この資料では、Linux® 開発環境として、Vine Linux ディストリビューションを使用します。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
3	仮想 OS 実行環境	OS を仮想的に実行するための環境です。 この説明では、「VMware Player for Windows」と呼ばれるフリーウェア・ソフトを使用しています。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
4	クロス・コンパイラ	ターゲット（Helio）向けの実行イメージを生成するためのコンパイラです。 この説明では、Mentor Graphics® 社から提供されている Sourcery CodeBench Lite Edition for ARM GNU/Linux を使用しています。 詳細については、SoC Linux 道場【其ノ参】を参照ください。
5	thttpd	組み込み機器では比較的使用されている Web サーバ・ソフトです。 「2-3. CGI による PWM 制御」を実行するために必要となります。 詳細については、SoC Linux 道場【其ノ四】を参照ください。

6	アルテラ Quartus® II	アルテラ FPGA のハードウェアを開発するためのツールです。 本説明書では、Quartus II バージョン v13.1 を使用しています。 ■ Quartus II 開発ソフトウェア https://www.altera.co.jp/products/design-software/fpga-design/quartus-ii/overview.html
7	Helio ボード	動作確認でターゲット・ボードとして使用する、 アルテラ Cyclone® V SoC を搭載したマクニカ Helio ボードです。 Helio には複数のリビジョンが存在しますが、この資料では、Rev1.2 または Rev1.3 を使用して動作確認を行っています。 ■ Helio ボード Rev1.2 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev12 ■ Helio ボード Rev1.3 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev13

2. ネットワーク・アプリで遠隔 PWM 制御

2-1. ソケット通信プログラムによる遠隔 PWM 制御

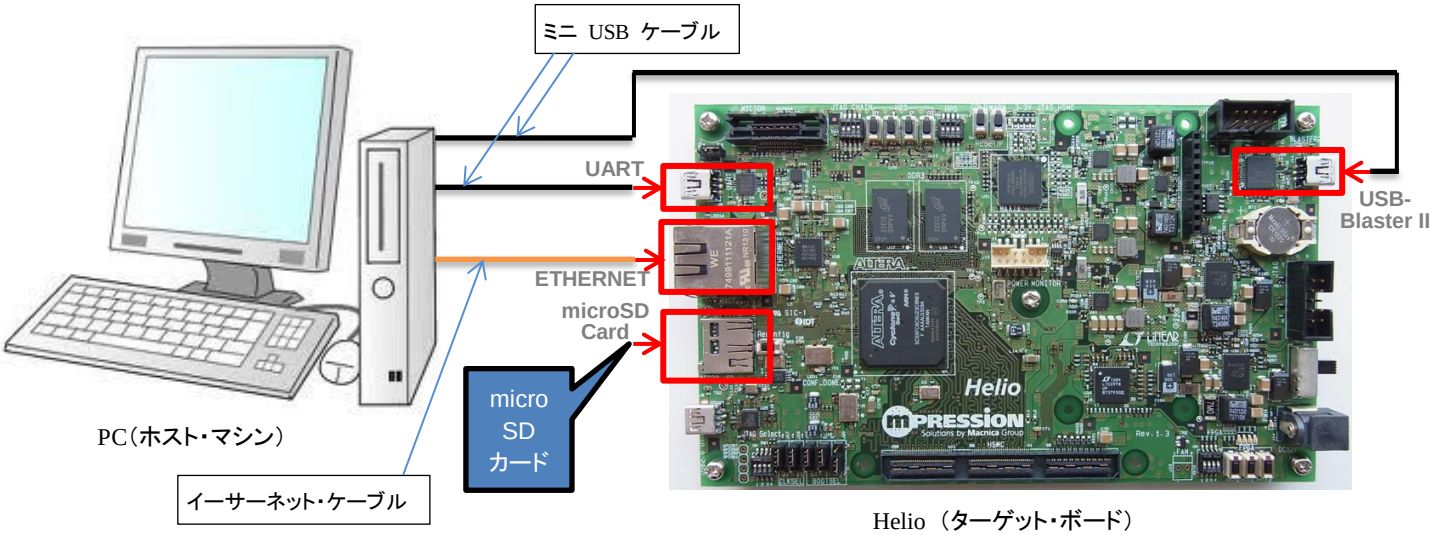
Helio で Linux が動作しているので、ここまでくるとネットワークのプログラミングが使えます。

既に CGI を使ってネットワーク制御できていますが、サーバ側は HTTP（thttp）のサーバ・プログラムを使用しています。

ここでは、ソケットのサーバとクライアント・プログラムを自前で作成して、遠隔 PWM 制御を行ってみます。

2-1-1. Helio 側の準備

(1) Helio とホスト PC が下図のように接続されていることを確認します。



【図 2-1-1.1】PC と Helio の接続図（全体）

① Helio の UART コネクタとホスト PC の USB ポートをミニ USB ケーブルで接続します。

② Helio の USB-Blaster™ II コネクタとホスト PC の USB ポートをミニ USB ケーブルで接続します。

③ Helio の ETHERNET コネクタとホスト PC のイーサーネット・ポートをイーサーネット・ケーブルで接続します。

④ Helio の microSD カード・スロットに、SoC Linux 道場【其ノ七】で使用した Helio 用 Linux の microSD カードを取り付けます。

(2) Helio の電源を入れて Linux が起動したら、**ターゲット側（Helio）**で以下のように root でログインして、udhcpc コマンドを実行して、DHCP サーバから Helio に IP アドレスが付与されるようにします（下記の例では、192.168.1.232 が付与されています）。

【注記】

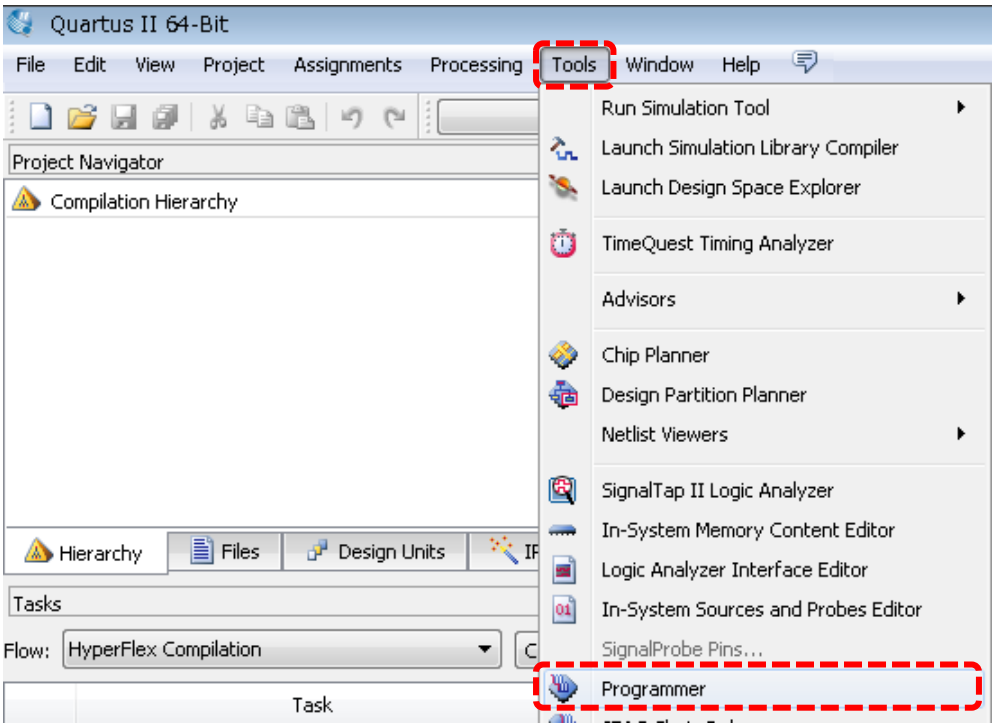
SoC Linux 道場【其ノ弐】で説明した DHCP サーバが設定されていて、使用できることを前提としています。

```
socfpga login: root
root@socfpga:~# udhcpc
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.232...
Lease of 192.168.1.238 obtained, lease time 21600
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

(3) PC のデスクトップにある Quartus II のアイコンをダブル・クリックして、Quartus II を起動します。

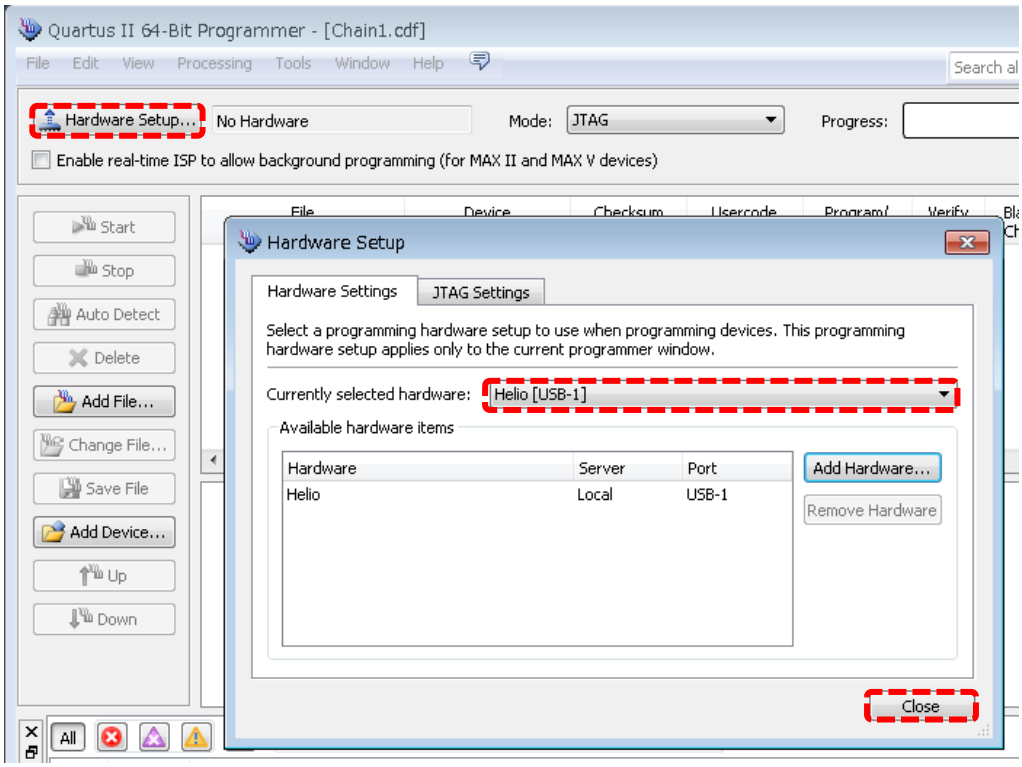
【図 2-1-1.2】Quartus II の起動

(4) Quartus II の Tools メニュー ⇒ Programmer を選択、または Programmer アイコン をクリックして、Programmer を起動します。



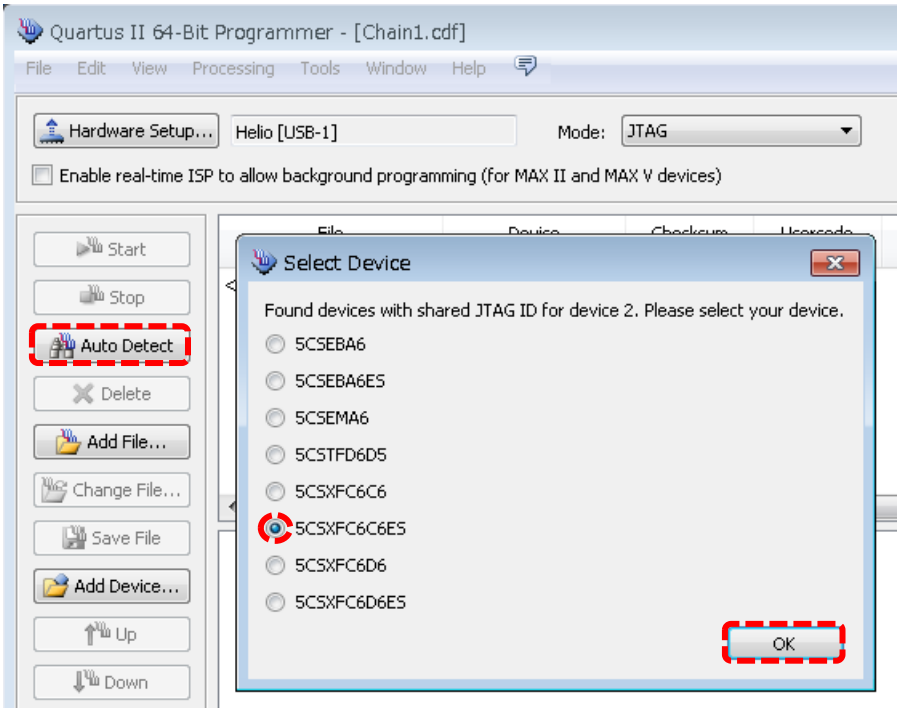
【図 2-1-1.3】Programmer の起動

(5) Programmer 内にある [Hardware Setup…] ボタンをクリックし、Currently selected hardware のプルダウンリストから Helio を選択します。 選択したら [Close] をクリックします。



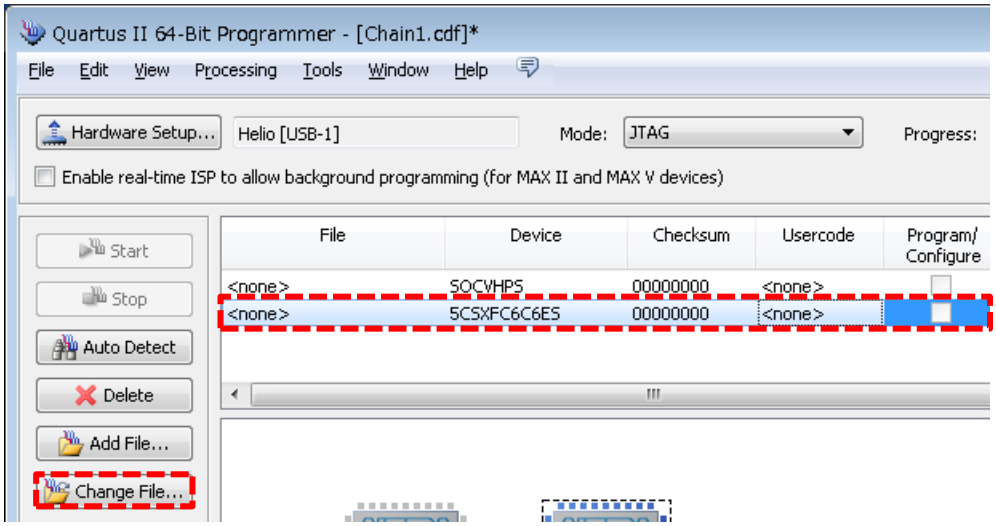
【図 2-1-1.4】Hardware Setup

(6) [Auto Detect] ボタンをクリックし、ボード上の JTAG チェインに接続されている FPGA を検出します。Select Device ウィンドウが現れたら “5CSXFC6C6ES” を選択し、[OK] をクリックします。



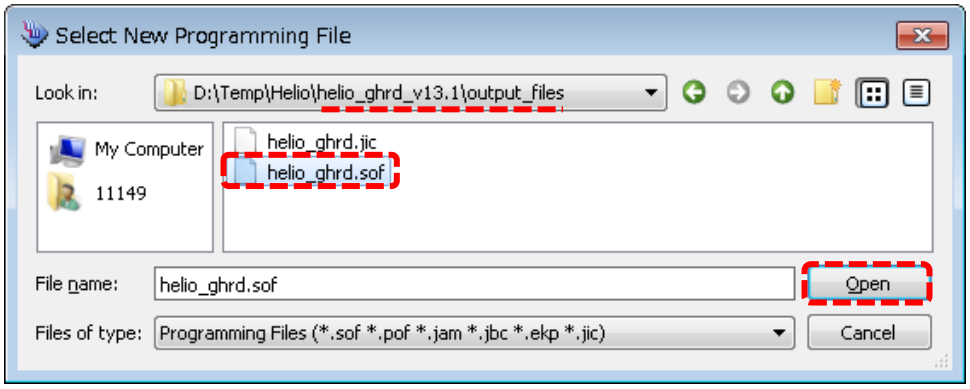
【図 2-1-1.5】デバイスの選択

(7) 5CSXFC6C6ES 上をクリックしてハイライトして、[Change File] をクリックします。



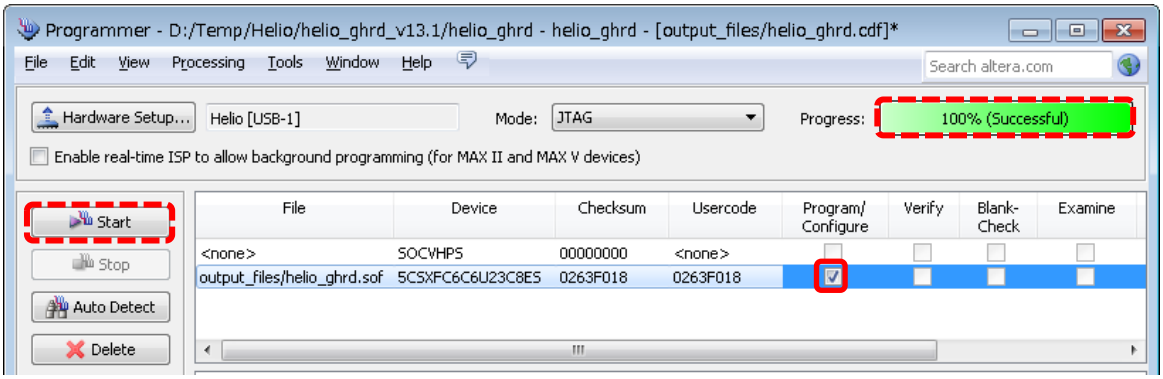
【図 2-1-1.6】[Change File] をクリック

(8) [Select New Programming File] ダイアログ・ボックスで、**SoC Linux 道場【其ノ七】**の「2-1. カスタム・ハードの作成」で PWM 制御回路を追加して作成した helio_ghrd_v13.1 フォルダの下の output_files フォルダをブラウズし、helio_ghrd.sof を選択して [Open] します。



【図 2-1-1.7】プログラミング・ファイルの選択

(9) Program/Configure にチェックを入れた後、[Start] ボタンをクリックしてコンフィギュレーションを行います。Progress バーに「100% (Successful)」と表示されれば、プログラミングは完了です。



【図 2-1-1.8】リファレンス・デザインによるコンフィギュレーションの実行

2-1-2. ソケット通信プログラムの作成と LED による PWM 制御の確認

(1) **【リスト 2-1-2.1】**と**【リスト 2-1-2.2】**にそれぞれサーバとクライアントのプログラム・ソースコードを示します。ここではポート番号に 3000 を使用しています。

なお、ソース・コードは Leafpad エディタ等を使用して一から書くこともできますが、大変なので別途記述済の Server_socket.c と Client_socket.c のソース・コードを用意しました。

この記述済 Server_socket.c と Client_socket.c ソース・コードをダウンロードして、SoC Linux 道場【其ノ弐】で説明した Samba サーバ経由で、Windows から Vine Linux のホーム・ディレクトリ（/home/tori）にコピーしてご利用ください。

```
// リピート・サーバ・プログラム(ポート 3000 を使用)
// ソケット・オプション使用
// クライアントから送られてくる番号によってモーターの PWM 制御を行う

#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define PORT 3000

static char *LED[] = {
 "/home/root/led0",
 "/home/root/led1",
 "/home/root/led2",
 "/home/root/led3"
};

static char *MES1[] = {
 "Pos. ",
 "Neg. "
};

static char *MES2[] = {
 "Speed0",
 "Speed1",
 "Speed2",
 "Speed3",
 "Speed4",
 "Speed5",
 "Speed6",
 "Speed7"
};
```

（次ページへ続く）

（前ページからの続き）

```
int main (int argc, char *argv[])
{
 int fd_socket, fd_accept, index, i;
 int led_fd[4];
 size_t len;
 struct sockaddr_in addr;
 char buff[BUFSIZ];

 if ((fd_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
   perror("Server : socket :");
   return 1;
 }
 index = 1;
 setsockopt(fd_socket, SOL_SOCKET, SO_REUSEADDR, &index, sizeof(index));

 addr.sin_family = AF_INET;
 addr.sin_addr.s_addr = INADDR_ANY;
 addr.sin_port = htons(PORT);

 if ((bind(fd_socket, (struct sockaddr *) &addr,
   sizeof(addr))) == -1) {
   perror("S-bind:");
   return 1;
 }

 if ((listen(fd_socket, 5)) == -1) {
   perror("S-listen:");
   return 1;
 }

 while(1) {
   if ((fd_accept = accept(fd_socket,
     (struct sockaddr *) &addr, &len)) == -1) {
     perror("S-accept:");
     return 1;
   }

   if ((read(fd_accept, buff, BUFSIZ)) == -1) {
     perror("S-read:");
     return 1;
   }

   index = atoi(buff);
```

socket ソケット・システム・コール ： ソケットの作成

bind ソケット・システム・コール ： ソケットとポートの結合

listen ソケット・システム・コール ： 接続キューの作成（サーバ側）[TCP]

accept ソケット・システム・コール ： 接続受入れ（サーバ側）[TCP]

read ソケット・システム・コール ： パケット受信 [TCP]

（次ページへ続く）

（前ページからの続き）

```
   if (index == 16) {
     strcpy(buff, "Server finshed.¥n");
     goto loop_last;
   }
   if (index > 16) {
     strcpy(buff, "Invalid Index.¥n");
     goto loop_last;
   }
   strcpy(buff, MES1[(index & 0x08) >> 3]);
   strcpy(buff + strlen(MES1[(index & 0x08) >> 3]), MES2[(index & 0x07)]);
   for (i = 0; i <= 3; i++) {
     if ((led_fd[i] = open(LED[i], O_WRONLY)) == -1) {
       perror("open:");
       return 1;
     }
     if ((index & (0x01 << i)) == (0x01 << i)){
       write(led_fd[i], "1", 1);
     }
     else{
       write(led_fd[i], "0", 1);
     }
     close(led_fd[i]);
   }
loop_last : if ((write(fd_accept, buff, strlen(buff) + 1)) == -1) {
     perror("S-write");
     return 1;
   }

   close(fd_accept);
   if (index == 16)
     break;
 }
 close(fd_socket);

 return 0;
}
```

write ソケット・システム・コール ： パケット送信 [TCP]

close ソケット・システム・コール ： ソケットの終了

【リスト 2-1-2.1】ソケット・サーバ・プログラム（ファイル名： Server_socket.c）

```
// クライアント・プログラム (Intel 版)
// 引数で番号を指定する

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>

#define PORT 3000

int main (int argc, char *argv[])
{
 int fd_socket;
 struct sockaddr_in addr;
 char buff[BUFSIZ];

 if (argc < 2) {
   fprintf(stderr, "usage: %s index¥n",
     basename(argv[0]));
   return 1;
 }
 if ((fd_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
   perror("C-socket:");
   return 1;
 }

 addr.sin_family = AF_INET;
 addr.sin_addr.s_addr = inet_addr("192.168.1.232");
 addr.sin_port = htons(PORT);

 if ((connect(fd_socket, (struct sockaddr *) &addr, sizeof(addr))) == -1) {
   perror("C-connect:");
   return 1;
 }

 if ((write(fd_socket, argv[1],strlen(argv[1]) + 1)) == -1) {
   perror("C-write:");
   return 1;
 }

 if ((read(fd_socket, buff, BUFSIZ)) == -1) {
   perror("C-read:");
   return 1;
 }
 printf("----- LED status from server -----¥n%s¥n", buff);
```

socket ソケット・システム・コール ： ソケットの作成

DHCP サーバで Helio に付与された
アドレスを記述する
(この例では、192.168.1.232)

connect ソケット・システム・コール
： サーバへの接続(クライアント側) [TCP]

write ソケット・システム・コール ： パケット送信 [TCP]

read ソケット・システム・コール ： パケット受信 [TCP]

(次ページへ続く)

（前ページからの続き）

```
close(fd_socket);

return 0;
}
```

close ソケット・システム・コール ： ソケットの終了

【リスト 2-1-2.2】ソケット・クライアント・プログラム（ファイル名： Client_socket.c）

(2) ソース・ファイルの準備ができたら、**ホスト側（Vine Linux）**から以下のようにコンパイルします。

生成されたサーバ側プログラム Server_socket を NFS 経由で Helio 上に運ぶための準備として、**ターゲット側（Helio）**から**ホスト側（Vine Linux）**のマウント先 /opt/Helio に Server_socket をコピーしておきます。

サーバ側プログラムをクロスコンパイル

```
[tori@Vine621 ~]$ arm-none-linux-gnueabi-gcc Server_socket.c -o Server_socket
[tori@Vine621 ~]$ cp Server_socket /opt/Helio/
[tori@Vine621 ~]$ gcc Client_socket.c -o Client_socket
```

クライアント側プログラムを gcc でコンパイル

(3) Helio の SD カードの /bin の下にもともと入っている busybox を使って、**ホスト側（Vine Linux）**の /opt/Helio を /mount_dir にマウントします。**ターゲット側（Helio）**で以下のコマンドを実行します。これにより、上記 (2) で /opt/Helio にコピーした Server_socket を Helio の /mount_dir から直接アクセスできるようになります。

【注記】

SoC Linux 道場【其ノ弐】で説明した、NFS サーバが使用できることを前提としています。

```
root@socfpga:~# busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
```

(4) SoC Linux 道場【其ノ六】の「2-3. カスタム・ドライバのコンパイルとロード」で作成しておいたドライバのカーネル・モジュール（helio_gpio.ko）を**ターゲット側（Helio）**で以下の insmod コマンドを実行してロードします。

【注記】

SoC Linux 道場【其ノ六】で説明したドライバのカーネル・モジュール（helio_gpio.ko）が Helio で使用できることを前提としています。

```
root@socfpga:~# insmod /mount_dir/helio_gpio.ko
Device registered successfully, Major No. = 252
```

(5) **ターゲット側（Helio）**で以下のようにサーバ・プログラムを実行します。

```
root@socfpga:~# /mount_dir/Server_socket
```

(6) **ホスト側(Vine Linux)**でクライアント・プログラムを以下のように実行します。

サーバ・プログラムから以下のような「回転方向」と「回転スピード」を示すメッセージが返され、Helio の LED が**【図 2-1-2.1】**のような状態になることが確認できれば、ソケット通信による制御は正しく動作しています。

クライアント・プログラムの引数に「16」を指定すると、サーバのプログラムを終了させることができます。

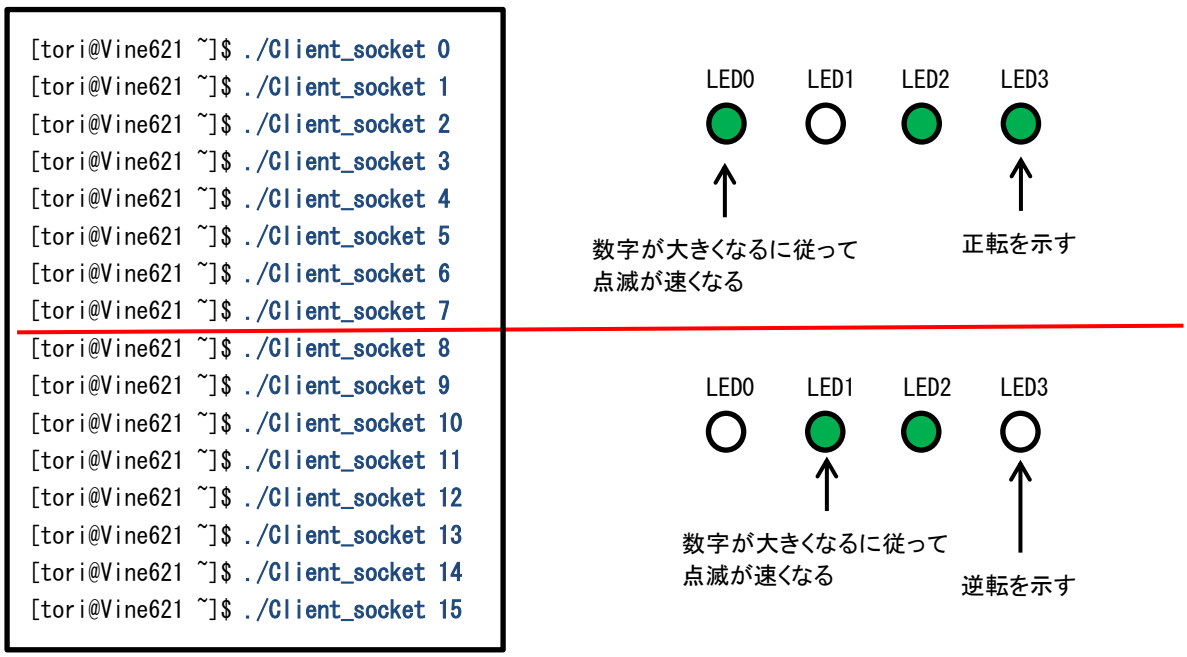
もし正しく動作しない場合は、「Helio と PC の接続は正しいか？」、「FPGA がコンフィギュレーションされているか？」、「Client_socket.c で編集した IP アドレスが正しいか？」、「NFS サーバが使用できる状態か？」などを再度確認してください。

```
[tori@Vine621 ~]$ ./Client_socket 0
----- LED status from server -----
Pos.  Speed0
[tori@Vine621 ~]$ ./Client_socket 1
----- LED status from server -----
Pos.  Speed1
[tori@Vine621 ~]$ ./Client_socket 11
----- LED status from server -----
Neg.  Speed3
 [tori@Vine621 ~]$ ./Client_socket 16
----- LED status from server -----
Server finished.

[tori@Vine621 ~]$
```

「回転方向」と「回転スピード」を示すメッセージが返される

引数に「16」を指定すると、サーバのプログラムを終了する



【図 2-1-2.1】ソケット通信プログラムによる遠隔 PWM 制御を使用した LED の状態

このようにソケット・プログラムが使用できる点は、サーバ側で Linux が動作している最大の強みです。

2-2. Android アプリによるアクセス

【注記】

この 2-2 項 は、Android クライアント・アプリがないため、実際の動作確認は行わず、概念の説明だけになります。

サーバ・プログラムができると、クライアント側に Android 端末も利用できます。

サーバをアクセスする Android のクライアント・アプリを作成すれば、下図の例のようにサーバをアクセスして、I/O を制御することができます。

【図 2-2.1】仮想 Android 端末からのソケットによるサーバ・アクセス例

3. モーター制御

【注記】

この 3 章 は、現在、拡張ボードの回路仕様が公開されていないため、実際のモーター制御の動作確認は行わず、概念の説明だけになります。

鳥海氏による「SoC FPGA 組み込み Linux 2 日間集中セミナー」では、実際に拡張ボードを使用したモーター制御を体験することができます。

Helio に演習用に作成した拡張ボードを接続し、その拡張ボードにモーターのドライブ回路を接続して、それをコントロールすることもできます。

(1) 拡張ボードにモーター・ドライブ回路を作成し、Helio と接続します。

(2) SoC Linux 道場【其ノ七】の 2-1 項「カスタム・ハードの作成」の“**【リスト 2-1.3】**ピン・アサインの変更（helio_ghrd.qsf の抜粋）”により、LED 用にピン・アサインしたものを拡張ボード用にピン・アサインします。

具体的にはピン・アサインを以下の**【リスト 3.1】**のように変更します（.qsf ファイルを直接変更します）。

```
# set_location_assignment PIN_U9 -to MOTOR_OUT[0]
# set_location_assignment PIN_AD4 -to MOTOR_OUT[1]
# set_location_assignment PIN_V10 -to MOTOR_OUT[2]
# set_location_assignment PIN_AC4 -to MOTOR_OUT[3]

set_location_assignment PIN_T12 -to MOTOR_OUT[0]
set_location_assignment PIN_T13 -to MOTOR_OUT[1]
set_location_assignment PIN_V13 -to MOTOR_OUT[2]
set_location_assignment PIN_W14 -to MOTOR_OUT[3]
```

コメント・アウトする

左記の通り追加する

【リスト 3.1】拡張ボード用ピン・アサイン（抜粋）

(3) 変更後、Quartus II でコンパイルし、生成された .sof ファイルを Helio にダウンロードします。

(4) SoC Linux 道場【其ノ七】の「2-2. LED の PWM 制御（コマンド・ライン）」を参考にして /led 0 ～ 3 をアクセスすれば、モーターが PWM 制御で回転します。

4. 起動スクリプトの作成方法

Helio を再起動するたびに NFS サービスの起動を行うのは、コマンドも長いことから面倒です。

ここでは、追加したサービスやあらかじめ起動しておきたいプログラムを Linux の起動時に実行させる方法を紹介します。

(1) **ターゲット側(Helio)**で以下の /etc/init.d ディレクトリに移動します。

このディレクトリに起動したいプログラムのスクリプトを置きます。

```
root@socfpga:~# cd /etc/init.d/
```

(2) 例えば、NFS のサービスを起動したい場合には、以下の**【リスト 4.1】**のようなスクリプト(ファイル名: mount.sh)を作成します。

ターゲット側(Helio)で「vi エディタ」を使用して作成します。

「vi エディタ」の使用方法については、下記のページなどを参考にしてください。

http://hp.vector.co.jp/authors/VA016670/unix/vi_reference.html

```
root@socfpga:~# vi mount.sh
```

```
#!/bin/sh

echo "Starting NFS mount....."
busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
```

【リスト 4.1】mount.sh の中身

【注記】

Linux 上で使用する mount.sh スクリプト・ファイルは、**文字コードを「UTF-8」、改行コードを「LF のみ」**で作成する必要があります。

vi エディタで作成する場合は問題ないですが、ご自身でテキスト・エディタ等を使用して mount.sh スクリプト・ファイルを作成して Helio に転送して使用する場合は注意してください。

(3) 作成したら**ターゲット側(Helio)**で以下の chmod コマンドで実行権を与えます。

```
root@socfpga:/etc/init.d# chmod 755 mount.sh
```

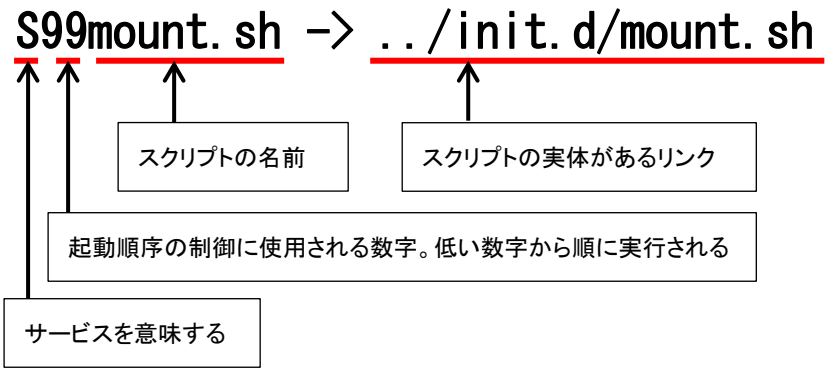
(4) このままでは、まだ mount.sh は自動で起動しません。明示的にどのラン・レベルで実行するかを指定する必要があります。

/etc ディレクトリの下に rc0.d ～ rc6.d および rcS.d のディレクトリがあります。

例えば、/etc/rcS.d の下を見てみると mountall.sh などのスクリプトのシンボリック・リンクがあります。

```
root@socfpga:/etc/init.d# ls -l /etc/rcS.d
lrwxrwxrwx    1 root     root            19 Mar   3  2014 S02banner.sh -> ../init.d/banner.sh
lrwxrwxrwx    1 root     root            18 Mar   3  2014 S02sysfs.sh -> ../init.d/sysfs.sh
lrwxrwxrwx    1 root     root            22 Mar   3  2014 S06alignment.sh -> ../init.d/alignment.sh
```

(5) シンボリック・リンクのファイル名の意味についてですが、以下のようになっています。



(6) 今回 mount.sh のシンボリック・リンクを、この /etc/rcS.d の下に作成します。

ターゲット側(Helio)で 以下の ln コマンドで作成します。

```
root@socfpga:/etc/init.d# cd /etc/rcS.d
root@socfpga:/etc/rcS.d# ln -s ../init.d/mount.sh S99mount.sh
root@socfpga:/etc/rcS.d# ls -l
  ～
lrwxrwxrwx    1 root     root            21 Mar   3  2014 S45mountnfs.sh -> ../init.d/mountnfs.sh
lrwxrwxrwx    1 root     root            21 Mar   3  2014 S55bootmisc.sh -> ../init.d/bootmisc.sh
lrwxrwxrwx    1 root     root            18 Nov   5 07:08 S99mount.sh -> ../init.d/mount.sh
root@socfpga:/etc/rcS.d#
```

(7) これで準備が整いましたので、 Helio を再起動します。

Helio が再起動すると Linux の起動メッセージの中に、mount.sh スクリプトの中に記述した次のようなメッセージがあるはずです。

Starting NFS mount.....

(8) Helio 上で Linux が起動したら、root でログインして、udhcpc コマンドを実行して DHCP サーバから Helio に IP アドレスが付与されるようにします（下記の例では、192.168.1.232 が付与されています）。

【注記】

SoC Linux 道場【其ノ弐】で説明した DHCP サーバが設定されていて、使用できることを前提としています。

```
socfpga login: root
root@socfpga:~# udhcpc
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.232...
Lease of 192.168.1.238 obtained, lease time 21600
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

(9) NFS マウントが成功していることを確認するために、/mount_dir ディレクトリに**ホスト側（Vine Linux）**の /opt/Helio がマウントされて見えることを確認します（/mount_dir ディレクトリの内容は下記の表示例と異なる場合があります）。

```
root@socfpga:~# ls /mount_dir
Server_socket    hello_driver.ko  printenv.cgi     thttpd.conf
busybox          index.html       pwm_cnt.cgi      toriumi.txt
busybox_command  led.cgi          pwm_set.sh
helio_gpio.ko    thttpd
root@socfpga:~#
```

上記のように見えれば、NFS マウントは成功です。

Thttpd などあらかじめ起動しておいた方が良いプログラムは、この方法で起動スクリプト・ファイルを作成し、シンボリック・リンクを作成しておきましょう。

本連載「SoC Linux 道場」はいかがでしたでしょうか？

アルテラ Cyclone V SoC を搭載したマクニカ Helio ボードをターゲットとして、ビルド済み SD カード・バイナリ・イメージを SD カードに書き込んで Linux の起動を確認していただきました。

その後、コンパイル環境（Linux マシン ＋ クロス・コンパイラ）を自分で用意し、Linux カーネル・ソース・コードの入手とコンパイル、カスタム・ドライバの作成、SoC FPGA へのユーザ回路の追加、PWM 制御の動作確認といった、Linux に関する基礎的な開発手法を確認していただけたかと思います。

皆様の Linux 開発の参考としてお役に立てれば幸いです。

改版履歴

Version	年月	概要
1	2015 年 3 月	新規作成