

SoC Linux 道場【其ノ六】

カスタム・ドライバの作成とコンパイル(その 2)

Ver.13.1

2017 年 8 月 Rev.3

カスタム・ドライバの作成とコンパイル(その 2)

目次

1. はじめに.....	3
2. カスタム・ドライバの作成とコンパイル(その 2)	4
2-1. GPIO(LED)ドライバの作成.....	4
2-2. Helio のリファレンス・デザインの入手と確認	11
2-3. カスタム・ドライバのコンパイルとロード	16
2-4. Quartus II Programmer による .sof ファイルのプログラム	20
2-5. デバイス・ファイルの作成とドライバ・アクセス.....	24
2-6. LED の CGI 制御.....	26
改版履歴	33

1. はじめに

マクニカ Helio ボードには、LED(LED_PIO)やディップ・スイッチ(DIP_PIO)、プッシュ・ボタン(BTN_PIO)などを含む基本的なリファレンス・デザインが用意されています。

今回は、このリファレンス・デザイン内の LED を制御するカスタム・ドライバを作成してコンパイルします。

Helio ボードに NFS 経由で転送してから実行し、作成したドライバによる LED アクセス状態を確認してみます。

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 6.5 x86_64 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【其ノ貳】 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【其ノ貳】 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、Linaro から提供されている 32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーンを使用しています。 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。 「2-6. LED の CGI 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. カスタム・ドライバの作成とコンパイル(その 2)

2-1. GPIO(LED)ドライバの作成

ここでは Helio 上にある LED をアクセスするドライバを作成します。

既に Helio には LED やディップ・スイッチ、プッシュ・ボタンを制御する、いわゆる GPIO のドライバがインストールされています。

しかし、ここは自分でデバイス・ドライバを作成できるようにならないと、カスタムのハードを追加した時に OS から制御することができません。

Linux® のドライバの種類としては、主に以下の 3 つのタイプが挙げられます。

- キャラクタ型
- ブロック型
- ネットワーク型

組み込み Linux の I/O 制御は、ほとんどがキャラクタ型デバイス・ドライバで行うことができます。

そこで、以下のような仕様で Helio の LED を制御するレガシーなキャラクタ型デバイス・ドライバを作成します。

- ① 1 ビットずつのアクセスにする。
- ② シェル・コマンドの echo や cat のコマンドで制御できるようにする。
- ③ 出力デバイス(LED)に対して '1' で点灯、'0' で消灯とする(これ以外の英数字はエラーとする)。
- ④ 入力デバイス(ディップ・スイッチやプッシュ・スイッチ)は ON で '1'、OFF で '0' が返ってくるようにする。
- ⑤ マイナー番号は以下の通りとする。
4~7 : LED 0~3
8~11 : ディップ・スイッチ 0~3
12~14 : プッシュ・スイッチ 0~2
- ⑥ 今回は割り込みをサポートしない。

このような仕様で作成したキャラクタ型の GPIO デバイス・ドライバのソース・コード(ファイル名: helio_gpio.c)を【リスト 2-1.1】に示します。

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 helio_gpio.c ソース・コードを用意しました。

「2-3. カスタム・ドライバのコンパイルとロード」でコンパイルする際は、記述済み helio_gpio.c ソース・コードをダウンロードして、SoC Linux 道場【其ノ貳】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。

```
// キャラクタ・デバイスの GPIO デバイス・ドライバ(cdev)
```

```
#define SUCCESS 0
#define NODE_NAME "char_drv"
#define MY_BUFF_SIZE 4096
```

```
#define MAP_SIZE          (0x20000)
#define MAP_BASE_ADDR     (0xFF200000) //lwAxiMaster Base Addr
#define LED_PIO_BASE      (0x10040)
#define DIP_PIO_BASE      (0x10080)
#define BTN_PIO_BASE      (0x100C0)
#define LED_PIO_DATA_OFFSET (0x0)
#define DIP_PIO_DATA_OFFSET (0x0)
#define BTN_PIO_DATA_OFFSET (0x0)
```

アドレス定義

```
#include <linux/delay.h>
#include <linux/irq.h>
#include <linux/uaccess.h>
#include <linux/io.h>
#include <linux/gpio.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/gpio.h>
```

```
MODULE_LICENSE("Dual BSD/GPL");
```

```
static int major_num = 0;
static int minor_num = 16;
static struct cdev char_cdev;
```

```
#define MY_BUF_SIZE 1024
static unsigned char k_buf[MY_BUF_SIZE];
static int pos;
static int read_count = 0;
```

(次ページへ続く)

(前ページからの続き)

```
static unsigned long gpio_table [] = {
    0,
    MAP_BASE_ADDR + LED_PIO_BASE + LED_PIO_DATA_OFFSET,
    MAP_BASE_ADDR + DIP_PIO_BASE + DIP_PIO_DATA_OFFSET,
    MAP_BASE_ADDR + BTN_PIO_BASE + BTN_PIO_DATA_OFFSET
};

typedef struct my_buff {
    int bit;
    int mode;
    void *cookie;
} GPIO_BUFF;

static int char_open (struct inode *inode, struct file *filp)
{
    int minor = MINOR(inode->i_rdev);
    int bit = minor & 0x0F;
    int mode = (minor & 0x08)? 0 : 1;
    void *cookie = ioremap(gpio_table[bit>>2], MAP_SIZE);
    GPIO_BUFF *GpioBuff;

    if(bit < 4 || bit > 15) {
        printk("Illegal minor number! MINOR Number = %d\n", minor);
        return -EINVAL;
    }

    GpioBuff = (GPIO_BUFF *)kmalloc (sizeof(GPIO_BUFF), GFP_KERNEL);
    GpioBuff->bit = bit;
    GpioBuff->mode = mode;
    GpioBuff->cookie = cookie;
    filp->private_data = GpioBuff;

    printk ("----- This is the Open Function -----<pre>\n");
    printk ("Node NAME: %s:\n", NODE_NAME);
    printk ("MAJOR Number = %d, MINOR Number = %d\n",
        MAJOR (inode->i_rdev), minor);

    printk ("Open device successfully: %s:\n\n", NODE_NAME);
    printk ("\n");

    return SUCCESS;
}
```

open 関数

(次ページへ続く)

(前ページからの続き)

close 関数

```
static int char_release (struct inode *inode, struct file *filp)
{
    GPIO_BUFF *GpioBuff = (GPIO_BUFF *)filp->private_data;
    kfree(GpioBuff);
    printk ("----- This is the Close Function -----¥n");
    printk ("Closing device: %s:¥n¥n", NODE_NAME);
    printk ("¥n");
    read_count = 0;

    return SUCCESS;
}
```

read 関数

```
ssize_t char_read (struct file *filp, char __user *buf, size_t count, loff_t *f_pos)
{
    volatile unsigned int mode_port;
    unsigned int port_data = 0;
    void *cookie;
    char read_buff[2] = {'0', '¥n'};

    GPIO_BUFF *GpioBuff = (GPIO_BUFF *)filp->private_data;
    int ret, bit;

    if (read_count) {
        return 0;
    }

    mode_port = GpioBuff->mode;
    if (mode_port == 1) {
        printk ("This GPIO is Output!! No use read func.¥n");
        return -EFAULT;
    }
    printk ("----- This is the READ function -----¥n");
    GpioBuff = filp->private_data;
    bit = GpioBuff->bit;
    printk ("bit=%d¥n", bit);
    cookie = GpioBuff->cookie;
    port_data = readl(cookie);
    if ((~port_data & (0x00000001 << (bit & 0x03))))
        read_buff[0] = '1';
    else
        read_buff[0] = '0';
}
```

(次ページへ続く)

(前ページからの続き)

```

    if (copy_to_user(buf, read_buff, count)) {
        printk("Copy_to_user Error!!\n");
        return -EFAULT;
    }

    printk ("¥n");

    ret = 2;
    read_count++;

    return ret;
}

    ← write 関数
ssize_t char_write (struct file *filp, const char __user *buf, size_t count, loff_t *f_pos)
{
    volatile unsigned int mode_port;
    unsigned int value;
    void *cookie;

    GPIO_BUFF *GpioBuff = (GPIO_BUFF *)filp->private_data;
    int bit;

    mode_port = GpioBuff->mode;

    if (mode_port == 0) {
        printk("This GPIO is Input!! No use write func.¥n");
        return -EFAULT;
    }

    printk ("---- This is the WRITE function ----¥n");
    if (copy_from_user(k_buf, buf, count)) {
        printk("copy_from_user failed¥n");
        return -EFAULT;
    }
    printk ("¥n");

    if (*k_buf != '0' && *k_buf != '1') {
        printk("k_buf is bad charctor!! k_buf is %s¥n", k_buf);
        return -EFAULT;
    }
}

```

(次ページへ続く)

(前ページからの続き)

```

        bit = GpioBuff -> bit;
        cookie = GpioBuff -> cookie;
        value = readl(cookie);
        if (!(*k_buf - 0x30))
            value = value | (0x00000001 << (bit - 4));
        else
            value = value & ~(0x00000001 << (bit - 4));
        writel(value, cookie);

        pos = count;
        printk("k_buf is %s pos = %d\n", k_buf, pos);
        return count;
    }

    struct file_operations char_fops = {
        .read = char_read,
        .write = char_write,
        .open = char_open,
        .release = char_release
    };

    static int char_init_module(void)
    {
        dev_t dev = MKDEV(major_num, 0);
        int major_num_ret;
        int cdev_ret = 0;
        //major_num_ret = register_chrdev (major_num, NODE_NAME, &char_fops);
        major_num_ret = alloc_chrdev_region(&dev, 0, minor_num, NODE_NAME);
        if (major_num_ret < 0) {
            printk ("Failed chrdev region %d\n", major_num_ret);
            unregister_chrdev_region(dev, minor_num);
            return major_num_ret;
        }
        //if (major_num_ret) {
        major_num = major_num_ret = MAJOR(dev);
        //}
        cdev_init(&char_cdev, &char_fops);
        char_cdev.owner = THIS_MODULE;

        cdev_ret = cdev_add(&char_cdev, MKDEV(major_num, 0), minor_num);
        if (cdev_ret < 0) {
            printk ("Failed cdev add %d\n", cdev_ret);
            cdev_del (&char_cdev);
            return cdev_ret;
        }
    }

```

(次ページへ続く)

(前ページからの続き)

```

    printk ("Device registered successfully, Major No. = %d¥n", major_num);
    return 0;
}

static void char_exit_module (void)
{
    //unregister_chrdev (major_num, NODE_NAME);
    dev_t dev = MKDEV(major_num, 0);

    cdev_del(&char_cdev);
    unregister_chrdev_region(dev, minor_num);

    printk ("Device unregistered successfully, Major No. = %d¥n", major_num);
}

module_init(char_init_module);
module_exit(char_exit_module);

```

【リスト 2-1.1】 キャラクタ型 GPIO ドライバ・ソース・コード(ファイル名:helio_gpio.c)の内容

2-2. Helio のリファレンス・デザインの入手と確認

今回の Helio 用デバイス・ドライバのポイントとしては、LED(LED_PIO_BASE)やディップ・スイッチ(DIP_PIO_BASE)、プッシュ・ボタン(BTN_PIO_BASE)のベース・アドレスです。

このアドレスは、Helio のハードウェア・リファレンス・デザインを Quarus II から Qsys を立ち上げて確認することができます。

ここでは、Helio 向けのハードウェア・リファレンス・デザインをダウンロードして、その内容を確認します。

- (1) 下記の Helio のページに行きます。

<http://www.rocketboards.org/foswiki/Documentation/MacnicaHelioSoCEvaluationKit>

- (2) ご使用の Helio のボード・リビジョンを確認します。ボード・リビジョンは下図の赤枠の位置に印刷されています(この例では、Rev.1.3 を使用)。



【図 2-2.1】 Helio のボード・リビジョン表記の確認

- (3) 確認したボード・リビジョンに合ったリンクを選択します(この例では、Rev.1.3 を選択)。

Board Specific Resources:

Schematic ,BOM, Reference Manual, Reference Designs, etc

Board Rev.	Download Link	Note
Helio Board Rev.1.2	Helio Resources for Rev.1.2	5CSXF
Helio Board Rev.1.3	Helio Resources for Rev.1.3	5CSXFC6C6U23C8NES
Helio Board Rev.1.4	Helio Resources for Rev.1.4	5CSXFC5C6U23C7N

Helio のボード・リビジョンに合ったリンクをクリック
(この例では、Rev.1.3 を選択)

【図 2-2.2】 Helio のボード・リビジョン別リンク

- (4) 移動したページからハードウェア・リファレンス・デザインのダウンロード・リンクをクリックして、ダウンロード保存します(この例では、Quartus II 13.1 向けの helio_ghrd_v13.1.zip を選択)。

Helio Release Contents for Rev.1.3:

Category	Item	Rev.	Download	Note
Documentation	Getting Started	1.3	Helio_Getting_Started_1.3.pdf	
	Reference Manual	1.2	Helio_reference_manual_v1.2.pdf	
Board References	Schematic	1.31	helio_board_SCH_v1.31.pdf	
	Bill of Materials	1.31	helio_board_BOM_v1.31.xls	
	PWB data	1.3	PWB_data_v1.3.zip	
Reference Materials	HW Reference Designs	v14.0	helio_ghrd_5csxc6es_v14.0.zip	Quartus II v14.0
	HW Reference Designs	v13.1	helio_ghrd_v13.1.zip	Quartus II v13.1

ハードウェア・リファレンス・デザインのダウンロード・リンクをクリックして、ダウンロード保存します
(この例では、Quartus II 13.1 向けの helio_ghrd_v13.1.zip を選択)

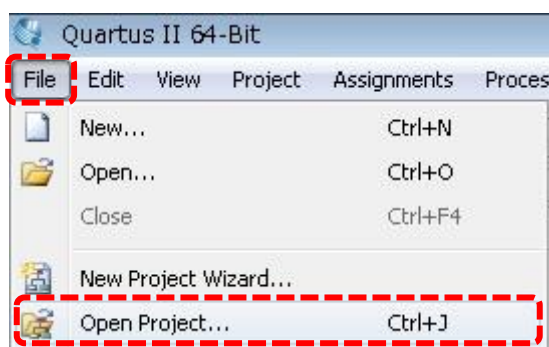
【図 2-2.3】リファレンス・デザインの入手

- (5) ダウンロードが完了したら、Windows 上で任意のフォルダ(この例では C:\Temp\Helio の下)に解凍します。
- (6) 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。

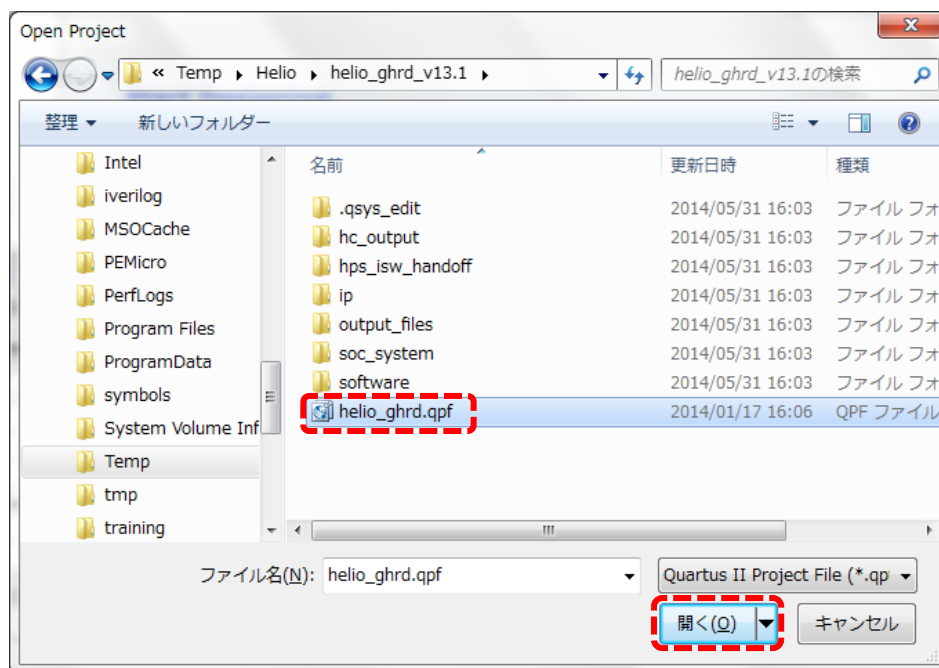


【図 2-2.4】Quartus II の起動

- (7) Quartus II の File メニュー ⇒ Open Project... を選択して、解凍しておいたリファレンス・デザイン内の helio_ghrd.qpf ファイルを選択し、[開く]をクリックしてプロジェクトを Open します。

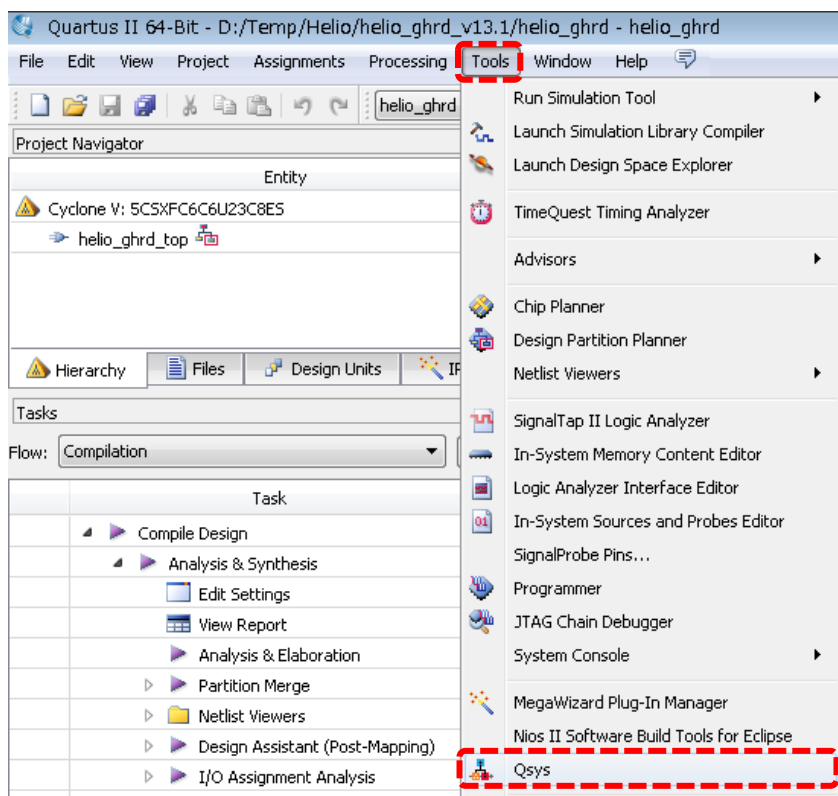


【図 2-2.5】File メニュー ⇒ Open Project... を選択

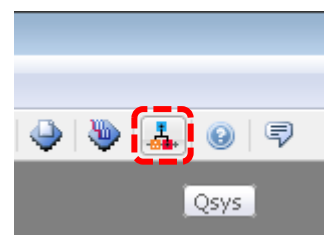


【図 2-2.6】 リファレンス・デザインのプロジェクトのオープン

- (8) プロジェクトが開いたら、Quartus II の Tools メニュー ⇒ Qsys を選択、または、Qsys アイコンをクリックして、Qsys を立ち上げます。

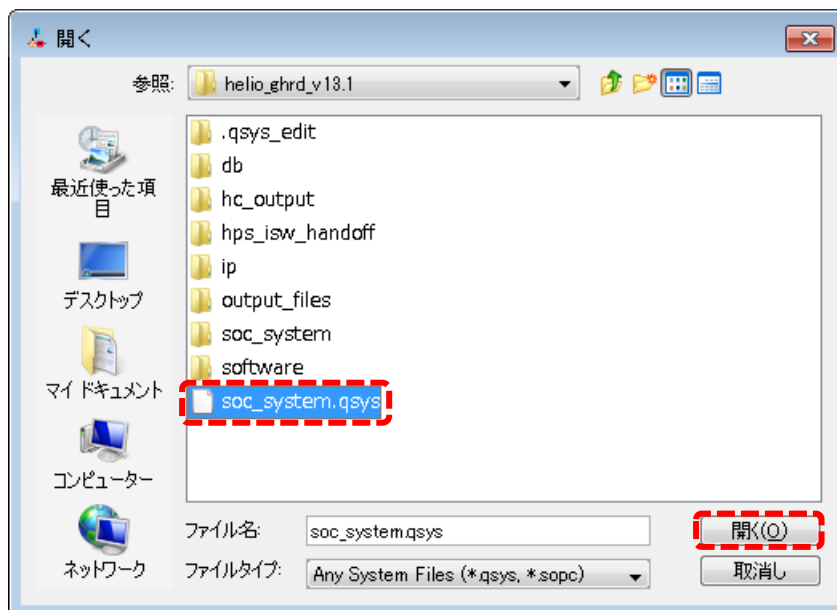


または



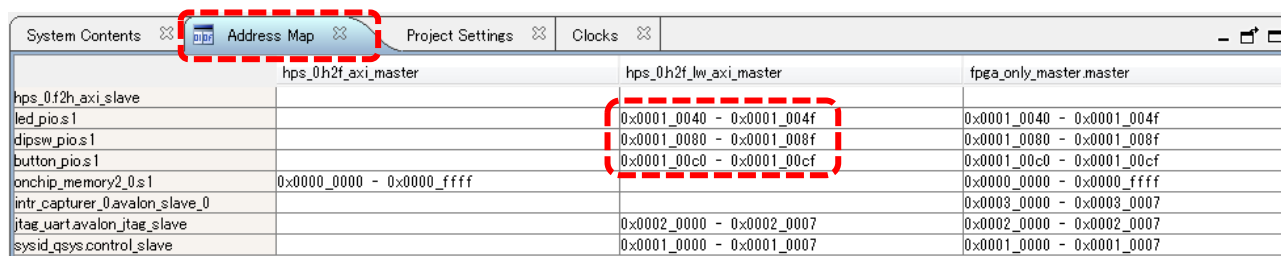
【図 2-2.7】 Qsys の起動

(9) 次に Qsys のプロジェクト・ファイル(ここでは soc_system.qsys)を開きます。



【図 2-2.8】 Qsys のプロジェクト・ファイルのオープン

(10) Qsys が起動したら Address Map タブをクリックします。



【図 2-2.9】 Address Map タブのクリック

- (11) 現れた画面の LED と ディップ・スイッチ、ボタン・スイッチ のアドレスを見てみると以下のようになっています。

led_pios1		0x0001_0040 - 0x0001_004f
dipsw_pios1		0x0001_0080 - 0x0001_008f
button_pios1		0x0001_00c0 - 0x0001_00cf

【図 2-2.10】LED、ディップ・スイッチ、ボタン・スイッチのアドレス

この先頭アドレスを【リスト 2-1.1】の中で指定しています。

従って、今後カスタムの PIO などを追加して GPIO ドライバを作成する際には、単純な動作であればこのアドレスを追加すれば基本的に OK です。

また、ベース・アドレスの下位 8 ビットがそれぞれ、

LED	:	0x40
ディップ・スイッチ	:	0x80
ボタン・スイッチ	:	0xc0

で、それぞれ 4 ビット、4 ビット、3 ビットの長さなのでマイナー番号の割り当てを、

LED	:	4~7
ディップ・スイッチ	:	8~11
ボタン・スイッチ	:	12~14

としています。

また、AXI のマスタのベース・アドレスは 0xFF200000 になっていますので、それもドライバの中で定義しています。

Helio に関わる場所は以上です。

【参考】

上記の“マイナー番号”は、後述「2-5. デバイス・ファイルの作成とドライバ・アクセス」において、mknod コマンドでデバイス・ノード(デバイス・ファイル)を作成する際に使用しています。

mknod コマンドでは、引数の一部として“メジャー番号”と“マイナー番号”を与えます。

“メジャー番号”は、どのデバイス・ドライバ・エントリを使用すべきかをカーネルに示します。

insmod コマンドでドライバのカーネル・モジュール(helio_gpio.ko)をロードする際に“メジャー番号”が返されます。

“マイナー番号”は、デバイスのどのサブ・ユニットがデバイス・ノード(デバイス・ファイル)に対応するのかをカーネルに示します。また、ドライバ内でも識別されます。

例えば、Helio の LED として led_pio がデバイスとすれば、その中の各 LED に対応した 4 つのビットがサブ・ユニット(マイナー番号)に対応し、ドライバ内で使用されます。

2-3. カスタム・ドライバのコンパイルとロード

【リスト 2-1.1】のキャラクタ型 GPIO ドライバ・ソース・コード `helio_gpio.c` が完成したら、コンパイルして Helio 上で動かしてみます。

- (1) SoC Linux 道場【其ノ五】の「3. カスタム・ドライバの作成とコンパイル(その 1)」で作成した、Hello ドライバ用の Makefile を今回のキャラクタ型 GPIO ドライバ用に書き直してコンパイルします。

Linux 上の一般ユーザ(この例では `tori`)のホーム・ディレクトリ(`/home/tori`)から以下のコマンドを実行して、Leafpad で Makefile を開いて、下記の【リスト 2-3.1】のように修正します。

SoC Linux 道場【其ノ五】で Makefile を作成していない場合は、下記【リスト 2-3.1】の内容で Makefile を新規に作成してください。

```
[tori@Vine65 ~]$ leafpad Makefile
```

注: コマンド行の先頭は、スペースではなく Tab を入れること!

```
obj-m := helio_gpio.o
all:
    make -C /home/tori/linux-socfpga M=$(PWD) modules
clean:
    make -C /home/tori/linux-socfpga M=$(PWD) clean
```

この部分を、`helio_gpio.o` に修正する

カーネルを置いたディレクトリを指定する

【リスト 2-3.1】キャラクタ型 GPIO ドライバ用 Makefile の内容

作成が終了したら、Leafpad で Makefile ファイルを保存して閉じます。

【注記】

- ① Makefile の書式として、コマンド行の先頭はスペースではなく **Tab** を入力してください。スペースではコンパイルがエラーとなるので注意してください。
- ② 「leafpad: ディスプレイをオープンできません:」というエラーが出る場合は「端末」を一旦終了し、再度「端末」を起動してから、上記の `leafpad` コマンドを再度実行してください。

- (2) 以下の make コマンドでキャラクタ型 GPIO ドライバをコンパイルします。helio_gpio.c と Makefile を作成したディレクトリで実行します(この例では、/home/tori)。

```
[tori@Vine65 ~]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
make -C /home/tori/linux-socfpga M=/home/tori/modules  
make[1]: ディレクトリ `/home/tori/linux-socfpga' に入ります  
CC [M] /home/tori/helio_gpio.o  
Building modules, stage 2.  
MODPOST 1 modules  
CC /home/tori/helio_gpio.mod.o  
LD [M] /home/tori/helio_gpio.ko  
make[1]: ディレクトリ `/home/tori/linux-socfpga' から出ます
```

【注記】

SoC Linux 道場【其ノ参】 で説明した、クロス・コンパイラがインストールされて、パスが設定されていて使用できることを前提としています。

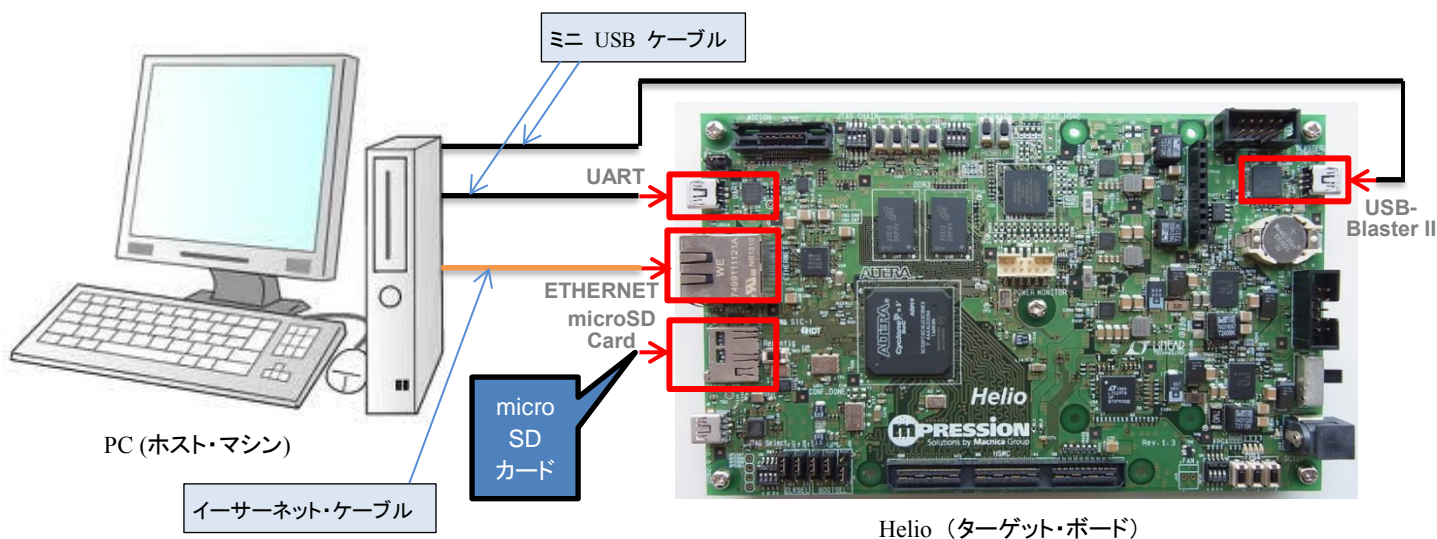
上記の make コマンドを実行してエラーが出るようであれば、
which arm-linux-gnueabihf-gcc コマンドを実行して、クロス・コンパイラのパスが通っているか確認してください。

クロス・コンパイラのパスが正しく通っていない場合は、再度 SoC Linux 道場【其ノ参】 を参照して必要な設定を行ってください。

または、次のコマンドを実行してクロス・コンパイラのパスを設定してください。

```
[tori@Vine65 ~]$ export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
```

(3) Helio とホスト PC を下図のように接続します。



【図 2-3.1】 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 のイーサネット・ポートをイーサネット・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【其ノ壱】で作成した Helio 用 Linux の microSD カードを取り付けます。
- (4) 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、udhcp 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、192.168.1.238 が付与されています)。

【注記】

SoC Linux 道場【其ノ貳】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:~# udhcp
udhcp (v1.20.2) started
Sending discover...
Sending select for 192.168.1.238...
Lease of 192.168.1.238 obtained, lease time 21600
/etc/udhcp.d/50default: Adding DNS 192.168.1.1
```

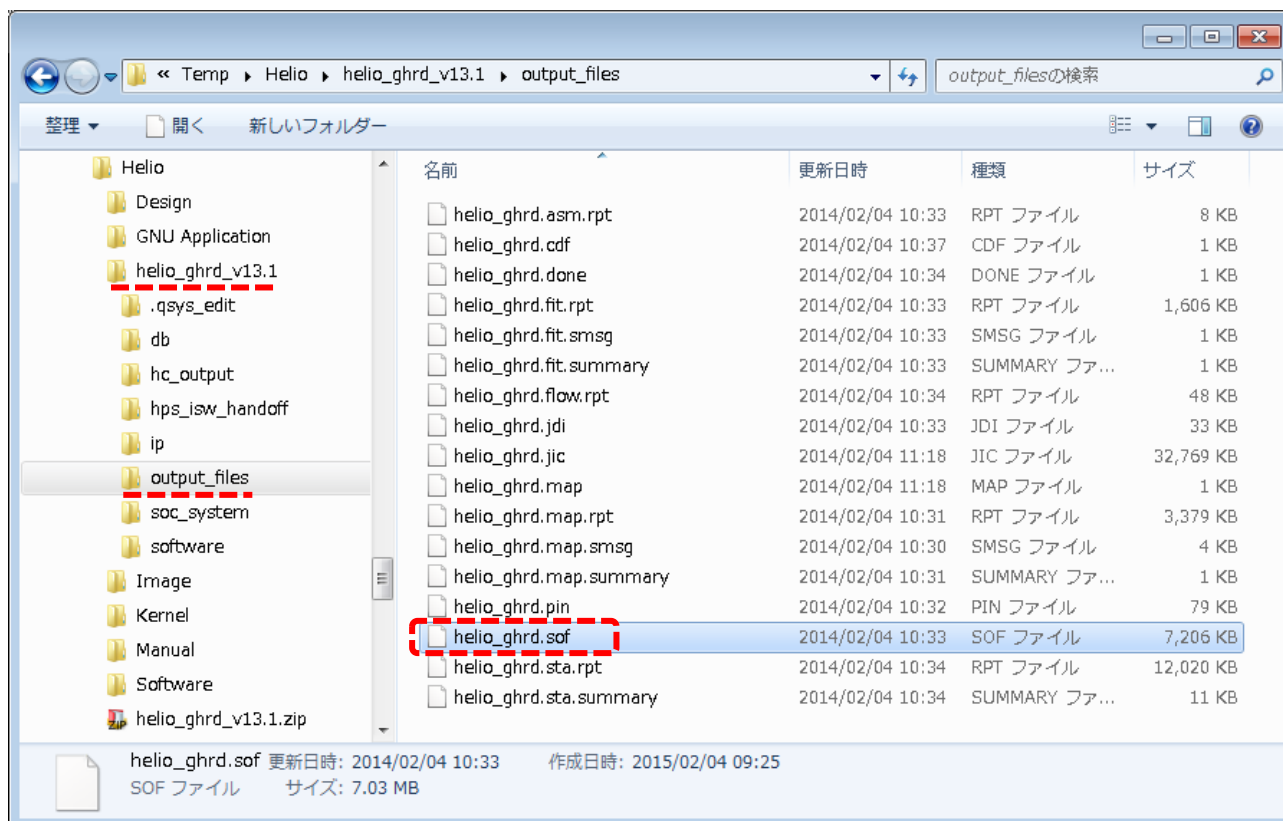
- (5) ドライバのカーネル・モジュール(helio_gpio.ko)が生成されているはずなので、それを NFS 経由で Helio 上に運ぶための準備として、**ターゲット側(Helio)**から**ホスト側(Vine Linux)**のマウント先 /opt/Helio に helio_gpio.ko をコピーしておきます。

ホスト側(Vine Linux) から以下のコマンドを実行します。

```
[tori@Vine65 ~]$ cp helio_gpio.ko /opt/Helio
```

2-4. Quartus II Programmer による .sof ファイルのプログラム

すでにダウンロードして解凍した Helio のファレンス・デザイン内の output_files フォルダの下に、helio_ghrd.sof ファイルがあります。



【図 2-4.1】 output_files フォルダの下にある helio_ghrd.sof ファイル

この .sof ファイルを Helio 上の SoC FPGA にプログラムすることで、FPGA がコンフィギュレーションされて、Helio の LED やスイッチ類が使用できるようになります。


以降に、Quartus II Programmer による .sof ファイルのプログラム手順を説明します。

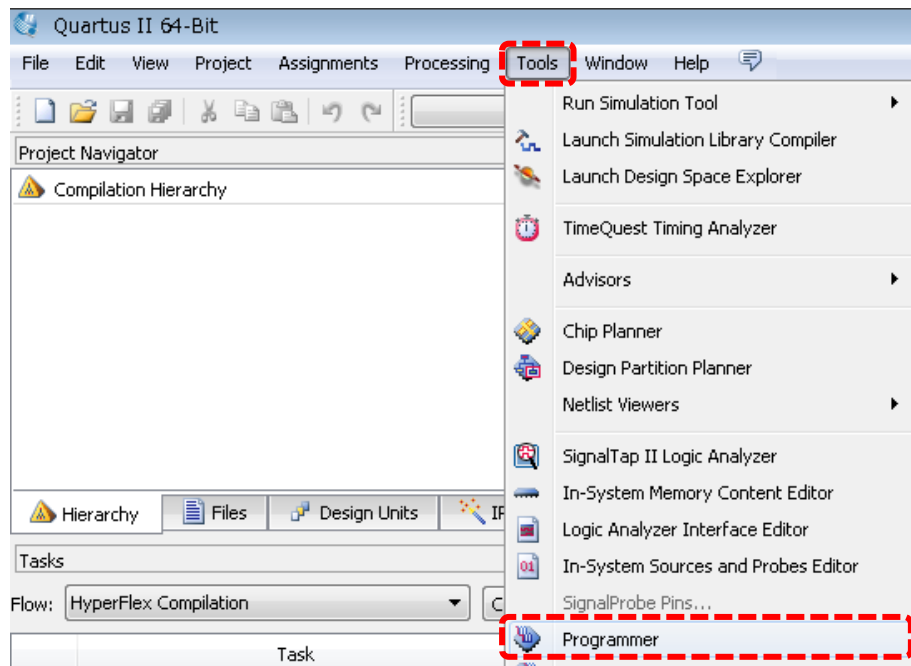
- (1) PC のデスクトップにある Quartus II のアイコンをダブルクリックして、Quartus II を起動します。

※ 2-2. 項で 既に Quartus II が起動済みである場合は、この手順はスキップしてください。



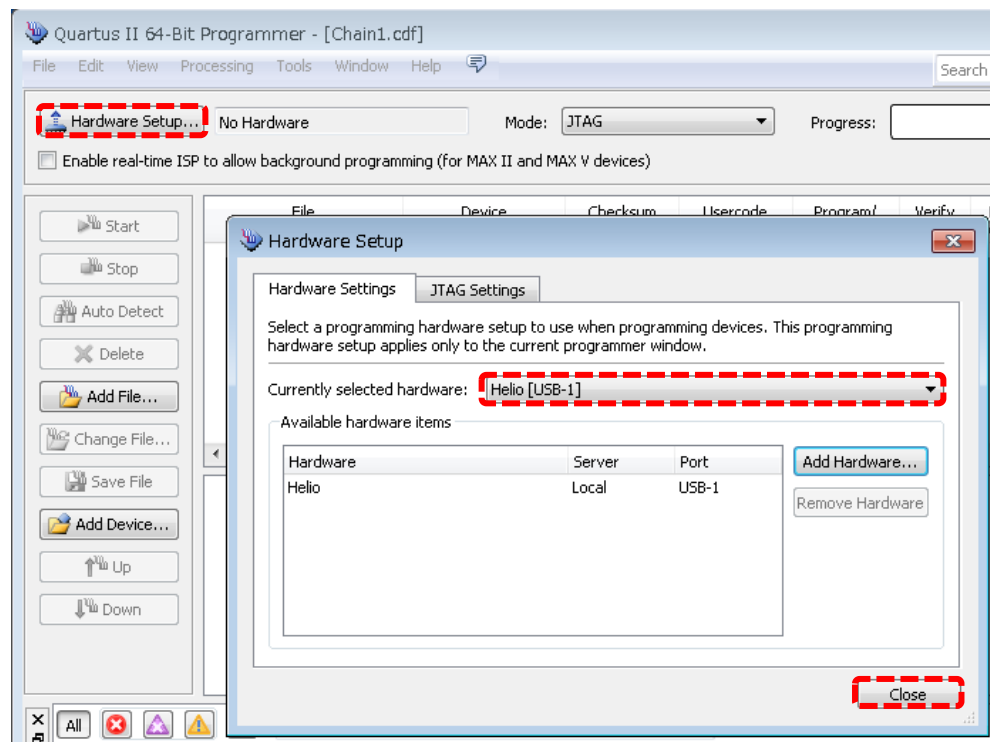
【図 2-4.2】 Quartus II の起動

- (2) 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、Programmer を起動します。



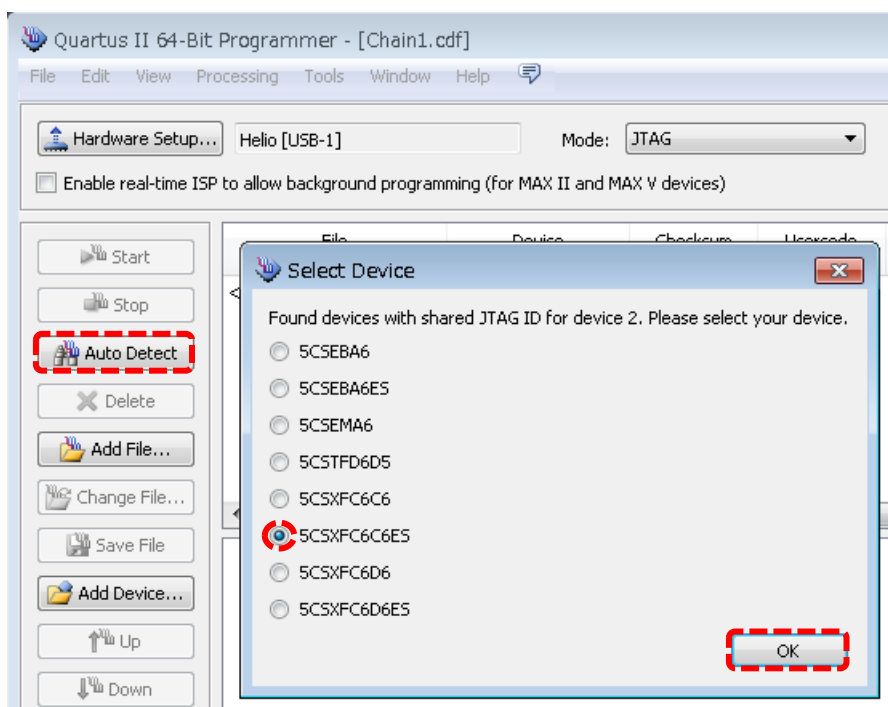
【図 2-4.3】 Programmer の起動

- (3) 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。



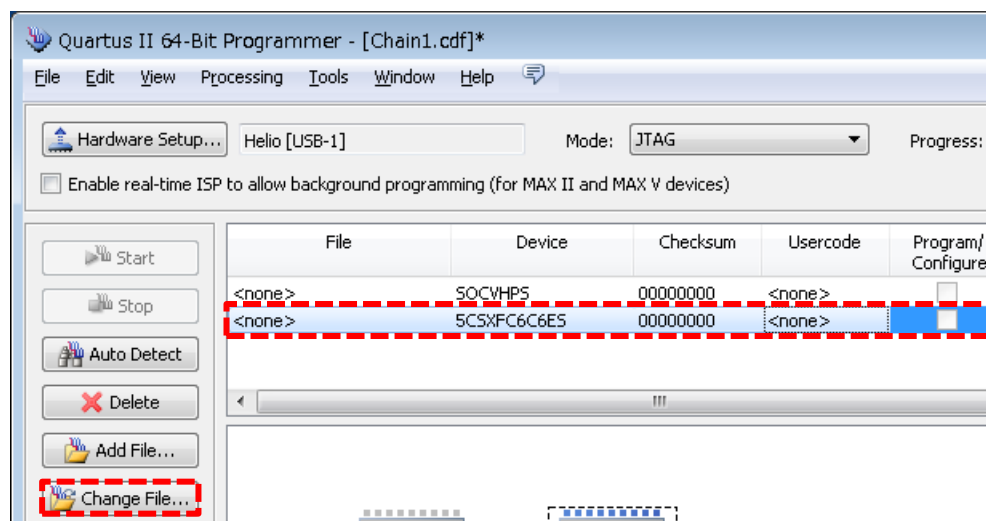
【図 2-4.4】 Hardware Setup

- (4) [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。



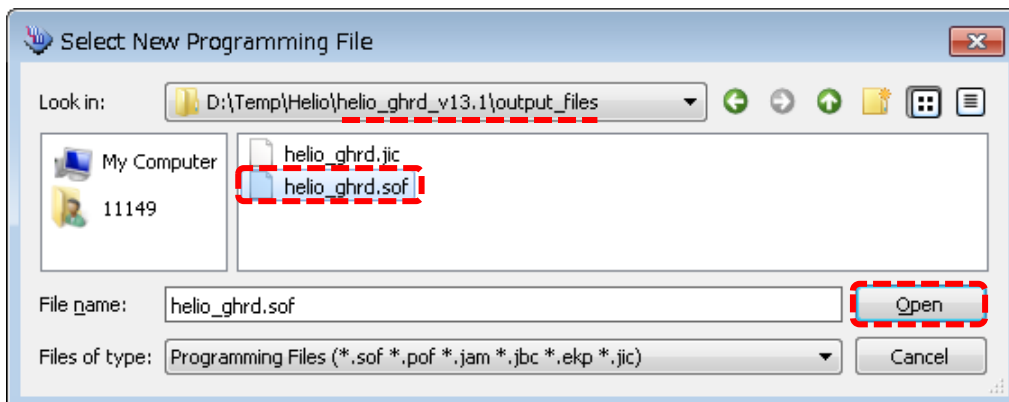
【図 2-4.5】 デバイスの選択

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



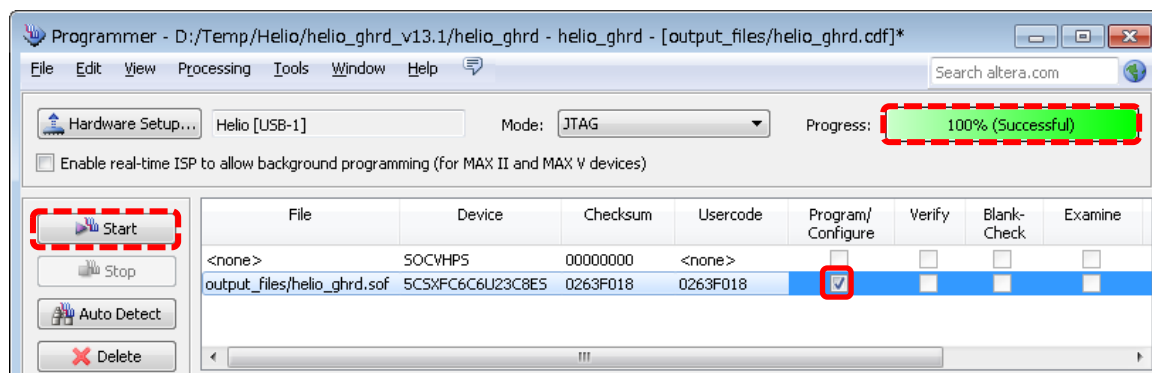
【図 2-4.6】 [Change File] をクリック

- (6) [Select New Programming File] ダイアログ・ボックスで、ダウンロード保存して解凍しておいた 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] をクリックします。



【図 2-4.7】プログラミング・ファイルの選択

- (7) 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 2-4.8】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

2-5. デバイス・ファイルの作成とドライバ・アクセス

- (1) 前述の「2-3. カスタム・ドライバのコンパイルとロード」の (5) で、コンパイルしたドライバのカーネル・モジュール(helio_gpio.ko)を /opt/Helio にコピーしました。

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。

【注記】

SoC Linux 道場【其ノ貳】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:~# mkdir /mount_dir
root@socfpga:~# busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
```

- (2) **ターゲット側(Helio)** で以下の insmod コマンドを実行して、すでに作成してマウントされているドライバをロードします。

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

- (3) 上記のコマンド出力では、メジャー番号が 252 を獲得したことを示していますので、以下の mknod コマンドでデバイス・ノード(デバイス・ファイル)を作成します。

```
root@socfpga:~# mknod -m 666 led0 c 252 4
root@socfpga:~# mknod -m 666 led1 c 252 5
root@socfpga:~# mknod -m 666 led2 c 252 6
root@socfpga:~# mknod -m 666 led3 c 252 7
root@socfpga:~# mknod -m 666 dip0 c 252 8
root@socfpga:~# mknod -m 666 dip1 c 252 9
root@socfpga:~# mknod -m 666 dip2 c 252 10
root@socfpga:~# mknod -m 666 dip3 c 252 11
root@socfpga:~# mknod -m 666 btn0 c 252 12
root@socfpga:~# mknod -m 666 btn1 c 252 13
root@socfpga:~# mknod -m 666 btn2 c 252 14
```

- (4) 次にシェルから LED のアクセスを行います。以下のように実行して LED を点灯・消灯させます。

```
root@socfpga:~# echo 0 > led0
root@socfpga:~# echo 0 > led1
root@socfpga:~# echo 0 > led2
root@socfpga:~# echo 0 > led3
root@socfpga:~# echo 1 > led0
root@socfpga:~# echo 1 > led1
root@socfpga:~# echo 1 > led2
root@socfpga:~# echo 1 > led3
```

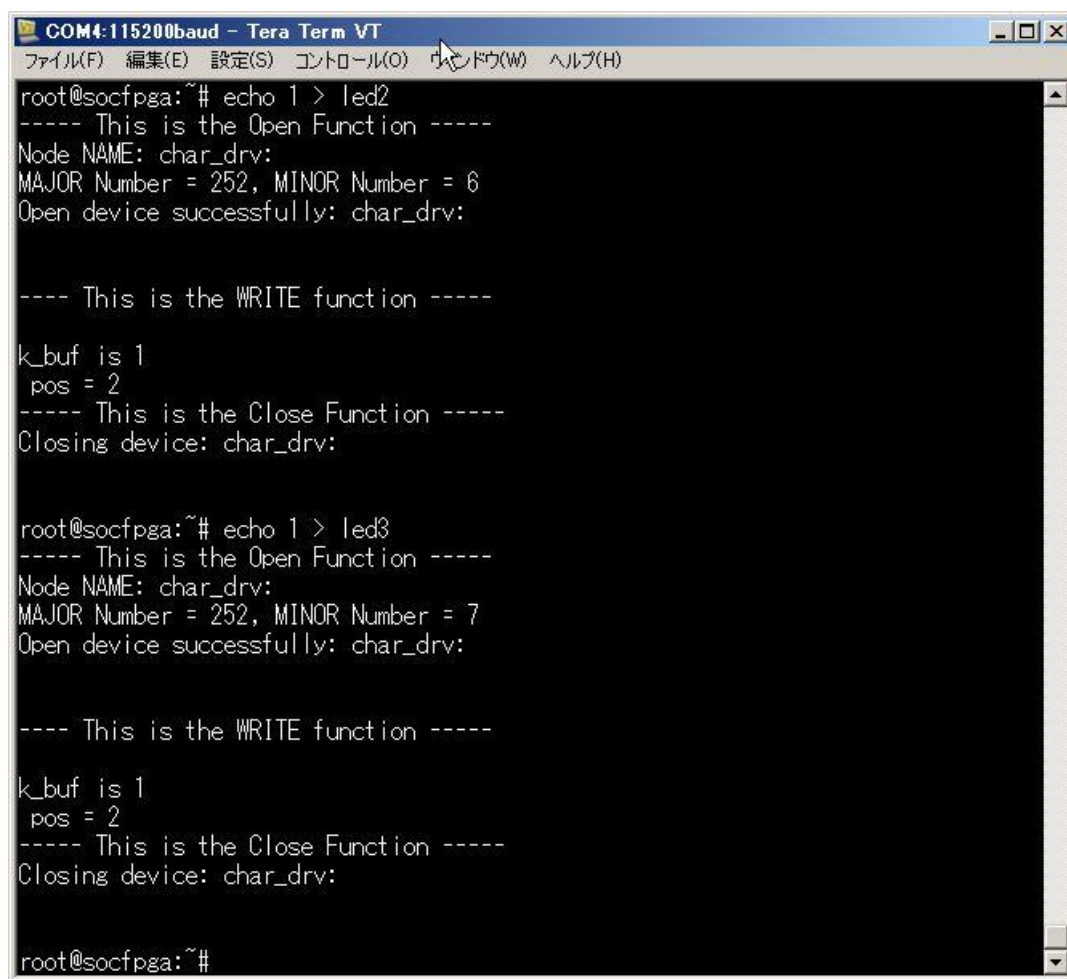
}

LED がそれぞれ消灯

}

LED がそれぞれ点灯

- (5) 前の (4) で echo コマンドを実行すると、helio_gpio.c ドライバの中で printk で出力しているメッセージが、下図のように Helio 側 Linux のコンソールに表示されます。



```
COM4:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

root@socfpga:~# echo 1 > led2
----- This is the Open Function -----
Node NAME: char_drv:
MAJOR Number = 252, MINOR Number = 6
Open device successfully: char_drv:

----- This is the WRITE function -----

k_buf is 1
pos = 2
----- This is the Close Function -----
Closing device: char_drv:

root@socfpga:~# echo 1 > led3
----- This is the Open Function -----
Node NAME: char_drv:
MAJOR Number = 252, MINOR Number = 7
Open device successfully: char_drv:

----- This is the WRITE function -----

k_buf is 1
pos = 2
----- This is the Close Function -----
Closing device: char_drv:

root@socfpga:~#
```

【図 2-5.1】 Helio 側 Linux のコンソール表示例

前の (4) のコマンド実行により LED が制御できていれば OK です。

2-6. LED の CGI 制御

次に LED 制御用の CGI プログラムを作成して、ブラウザから LED の制御を試してみます。

CGI は Common Gateway Interface(コモン・ゲートウェイ・インタフェース)の略で、ウェブ・サーバ上でユーザ・プログラムを動作させるための仕組みです。

(1) CGI プログラムのソース・コード(ファイル名: led.c)を次の【リスト 2-6.1】に示します。

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 led.c ソース・コードを用意しました。

この記述済 led.c ソース・コードをダウンロードして、[SoC Linux 道場【其ノ貳】](#)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。

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int i;
    int led_fd[4];
    char *env;
    char buff[10];

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

    printf("Content-type: text/html\n\n");
    printf("<html><body><p><big>LED control CGI sample</big></p>\n");
    printf("<form action=\"%led.cgi\" method=\"%get\">\n");
    env = (char *)getenv("QUERY_STRING");
    if (env == NULL) env = "led";
    //printf("<p>%s</p>\n", env);
    for (i = 0; i <= 3; i++) {
        //strcpy(buff, "/tmp/");
        strcpy(buff, "/home/root/");
        strcat(buff, LED[i]);
        if ((led_fd[i] = open(buff, O_WRONLY)) == -1) {
            perror("open:");
            return 1;
        }
        printf("%s<input type=\"%checkbox\" name=\"%s\" value=\"%on\"", LED[i], LED[i]);
        if (strstr(env, LED[i])) {
            write(led_fd[i], "1", 1);
            printf(" checked>\n");
        }
        else {
            write(led_fd[i], "0", 1);
            printf(">\n");
        }
    }
    printf("<input type=\"%submit\" value=\"%submit\"><br>\n");
    printf("</form></body></html>\n");
    return 0;
}
```

open システム・コールを実行し、ファイル・ディスクリプタを取得

write システム・コールを実行

write システム・コールを実行

【リスト 2-6.1】 LED 制御 CGI プログラム・ソース・コード(ファイル名:led.c)の内容

- (2) **ホスト側(Vine Linux)** から以下のコマンドで、led.c のクロス・コンパイルを実行します。

```
[tori@Vine65 ~]$ arm-linux-gnueabi-gcc led.c -o led.cgi
```

- (3) led.cgi が生成されているはずなので、それを 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 led.cgi をコピーしておきます。

ホスト側(Vine Linux) から以下のコマンドを実行します。

```
[tori@Vine65 ~]$ cp led.cgi /opt/Helio/
```

- (4) 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。

これにより、上記 (3) で /opt/Helio にコピーした led.cgi を Helio の /mount_dir に送ります。

また、CGI プログラムを実行するためには、SoC Linux 道場【其ノ四】で説明した tthttpd という Web サーバ・ソフトを使用します。

SoC Linux 道場【其ノ四】の「2-2. tthttpd のクロス・コンパイルとインストール」を一通り実施している場合は、tthttpd のファイル(tthttpd、tthttpd.conf、index.html、printenv.cgi)が /opt/Helio に既にコピーされているはずなので、これらのファイルも Helio の /mount_dir に送られます。

ターゲット側(Helio)で以下のコマンドを実行します。

【注記】

※1. SoC Linux 道場【其ノ貳】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。

※2. tthttpd のファイル(tthttpd、tthttpd.conf、index.html、printenv.cgi)が、**ホスト側(Vine Linux)**の /opt/Helio ディレクトリにコピーされていない場合は、SoC Linux 道場【其ノ四】の「2-2. tthttpd のクロス・コンパイルとインストール」の 20 ページ (8) まで実行して tthttpd のファイルを /opt/Helio ディレクトリにコピーしてから、これ以降のステップを実行してください。

```
root@socfpga:~# busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
root@socfpga:~# ls /mount_dir
busybox          hello_driver.ko  printenv.cgi     toriumi.txt
busybox_command  index.html       tthttpd
helio_gpio.ko    led.cgi          tthttpd.conf
```

- (5) **ターゲット側(Helio)** で `thttpd` の設定を行います。まず NFS 経由で見えている 4 つのファイル (`thttpd`、`thttpd.conf`、`index.html`、`printenv.cgi`) を `root` ディレクトリにコピーしておきます。

```
root@socfpga:~# cp /mount_dir/thttpd .
root@socfpga:~# cp /mount_dir/thttpd.conf .
root@socfpga:~# cp /mount_dir/index.html .
root@socfpga:~# cp /mount_dir/printenv.cgi .
root@socfpga:~# ls
README      index.html  led1        led3        printenv.cgi  thttpd.conf
altera      led0        led2        mount_dir   thttpd
```

最後の「.」(ドット)を忘れないこと

- (6) **ターゲット側(Helio)** で以下のように必要なユーザやファイルを作成します。

```
root@socfpga:~# adduser httpd
Changing password for httpd
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Bad password: too simple.

Warning: weak password (continuing).
Re-enter new password:
Password changed.
root@socfpga:~# mkdir /home/httpd/html
root@socfpga:~# mkdir /home/httpd/html/cgi-bin
root@socfpga:~# cp index.html /home/httpd/html/
root@socfpga:~# chmod 644 /home/httpd/html/index.html
root@socfpga:~# cp printenv.cgi /home/httpd/html/cgi-bin
root@socfpga:~# chmod 755 /home/httpd/html/cgi-bin/printenv.cgi
```

SoC Linux 道場【其ノ四】で実施済みであれば、エラーが出ますが無視して問題ありません

適当なパスワード(例えば、toriumi)を入力します(非表示)

SoC Linux 道場【其ノ四】で実施済みであれば、エラーが出ますが無視して問題ありません

- (7) **ターゲット側(Helio)** で `led.cgi` ファイルを `/home/httpd/html/cgi-bin` ディレクトリへコピーしてから、`chmod` コマンドで実行権限を与えます。

```
root@socfpga:~# cp /mount_dir/led.cgi /home/httpd/html/cgi-bin
root@socfpga:~# chmod 755 /home/httpd/html/cgi-bin/led.cgi
```

- (8) ターゲット側(Helio) で tthttpd を以下のコマンドで起動します。その後、ps コマンドでプロセスが起動していることを確認します。

```
root@socfpga:~# ./tthttpd -C tthttpd.conf
root@socfpga:~# ps
```

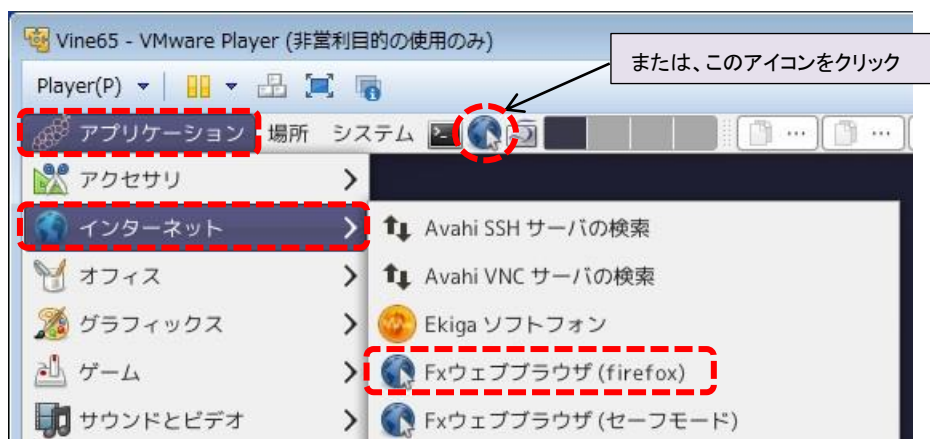
PID	USER	VSZ	STAT	COMMAND
1	root	1312	S	init [5]
2	root	0	SW	[kthreadd]
3	root	0	SW	[ksoftirqd/0]

(途中省略)

```
704 httpd      2388 S  ./tthttpd -C tthttpd.conf
705 root       1944 R  ps
```

root@socfpga:~#

- (9) ホスト側(Vine Linux) で Web ブラウザ(Firefox)を起動します。



【図 2-6.1】 Web ブラウザ(Firefox)を起動

(10) 例えば DHCP サーバから付与された Helio の IP アドレスが「192.168.1.238」の場合は、

`http://192.168.1.238:8080/cgi-bin/led.cgi`

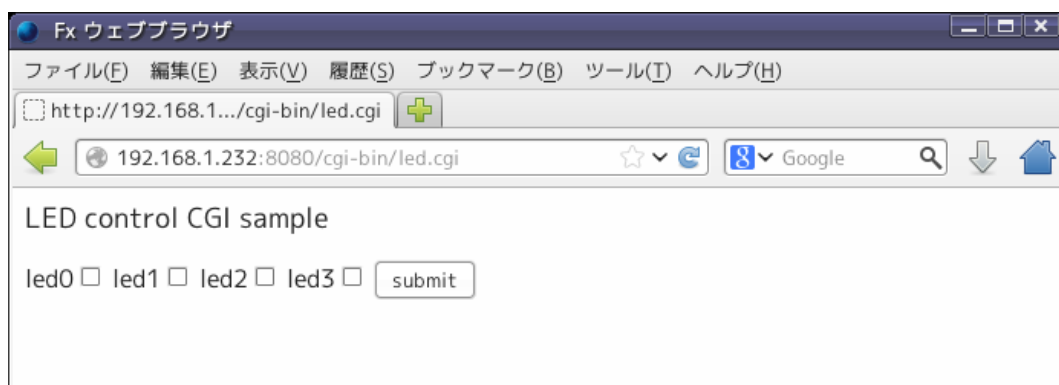
を URL に指定して、下図のようなブラウザが現れて、LED がアクセスできるようになります。

各 LED のチェック・ボックスに、チェックを入れて [submit] ボタンをクリックすると LED が点灯します。チェックを外して [submit] ボタンをクリックすると LED が消灯します。

【注記】

LED 制御 CGI を実行する前には、前述の「2-5. デバイス・ファイルの作成とドライバ・アクセス」の (2), (3) で実行した下記のコマンドによって、ドライバのロードとデバイス・ノード(デバイス・ファイル)の作成を行っておく必要があります。

```
root@socfpga:~# insmod /mount_dir/helio_gpio.ko
root@socfpga:~# mknod -m 666 led0 c 252 4
root@socfpga:~# mknod -m 666 led1 c 252 5
root@socfpga:~# mknod -m 666 led2 c 252 6
root@socfpga:~# mknod -m 666 led3 c 252 7
```



【図 2-6.2】 LED 制御 CGI

もしうまく表示されない場合は、以下の項目を確認してください。

- SoC Linux 道場【其ノ四】の「2-2. tthttpd のクロス・コンパイルとインストール」が一通り実施されていて、`printenv.cgi` が正しく表示されるようにしてください。
この時点で正しく表示されない場合は、【其ノ四】23 ページ (14) の ① ~ ④ などを確認して、`printenv.cgi` が正しく表示されることを確実にしてください。
またこれによって、`tthttpd` のファイル(`tthttpd`、`tthttpd.conf`、`index.html`、`printenv.cgi`)が、**ホスト側(Vine Linux)**の `/opt/Helio` ディレクトリにコピーされていることを確認してください。
- CGI プログラムのソース・コード `led.c` の内容が正しいか？
- `led.c` が正しくクロス・コンパイルされているか？
- クロス・コンパイルした `led.cgi` を `/opt/Helio` にコピーしたか？
- `busybox` を使って、**ホスト側(Vine Linux)**の `/opt/Helio` を `/mount_dir` にマウントしたか？

上記が正しく行われていれば、再度 28 ページの (5) から実行してください。

次回の SoC Linux 道場【其ノ七】 では、Helio ボードの FPGA 上にモーターに対して PWM 制御するユーザ独自のハードウェアを追加して動作の確認を行います (Helio ボードの LED の様子で PWM 制御できているかどうかを確かめます)。

改版履歴

Revision	年月	概要
1	2015 年 2 月	新規作成
2	2015 年 3 月	誤記訂正 P.24 誤) <code>mkdir mount_dir</code> ⇒ 正) <code>mkdir /mount_dir</code>
2.1	2015 年 3 月	アルテラ社の Web サイトのリニューアルに伴う URL 変更
3	2017 年 8 月	① Sourcery CodeBench Lite Edition for ARM GNU/Linux の配布終了に伴い、クロス・コンパイル環境として Linaro Toolchain を使用する説明に変更 ② ゲスト OS を Vine Linux 6.2.1 i686 から Vine Linux 6.5 x86_64 に変更

Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

免責およびご利用上の注意

弊社より資料を入手されましたお客様におかれましては、下記の使用上の注意を一読いただいた上でご使用ください。

- 本資料は非売品です。許可無く転売することや無断複製することを禁じます。
- 本資料は予告なく変更することがあります。
- 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がございましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。
株式会社マクニカ アルティマ カンパニー <https://www.alt.mcnica.co.jp/> 技術情報サイト アルティマ技術データベース <http://www.altima.jp/members/>
株式会社エルセナ <http://www.elsena.co.jp> 技術情報サイト ETS <https://www.elsena.co.jp/elspear/members/index.cfm>
- 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
- 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカー発行の英語版の資料もあわせてご利用ください。