株式会社 ビズライト・テクノロジー

BH3
テクニカルノート

最終更新日: 2018/03/28
バージョン: 1.0

（株）ビズライト・テクノロジー

1. 本製品について

この度は BH シリーズ、BH3 をご購入いただき、誠にありがとうございます。
本製品は下記を組み込んだセットとなっております。
・金属ケース（放熱用パッドつき）
・Raspberry Pi3 Model B
・（株）ビズライト・テクノロジー、オリジナル基盤
・AC アダプター（7.5V3A）

※microSD カードは別売となっております。

2. BH3 の概要

BH3 は Raspberry Pi3 Model B を使用した汎用エンベデッド STB です。
ご好評いただきました当社 BH2 のオリジナル基盤パターンを変更し汎用スイッチに割り当てられている GPIO を開放できるようにするなど、機能性もアップしております。
Raspberry Pi は特に IoT 分野などに多く使われはじめております。しかし、プロトタイプを終えて実際にフィールドで利用するためには、電源断時に正常終了させたい、電源のブレーカー連動、システム時刻保持、電源の安定化、耐ノイズ、放熱、ウオッチドックタイマーなどクリアしなければならない問題がたくさん出てきます。BH3 は特に堅牢性に重点を置き、フィールドでも使える STB を目指して開発されました。

3. 電気二重層コンデンサ簡易 UPS 機能

Linux ベースであるが故に正常なシャットダウンの確保は必須となります。BH3 は供給電源断を検出し、CPU に割込みを発生させます。電気二重層コンデンサによる簡易 UPS 機能を持ち、1 回のシャットダウン発生時に約 30 秒程度（環境に依る）の無電源動作を行うことができるため、プログラムはこの間に安全なシャットダウンプロセスを実行することが可能となり、OS のクラッシュや Micro SD カードのクラッシュの可能性を大幅に低減させることが可能です。
これにより、Raspberry Pi 環境で OS レスの機器組込シングルボードと同じような扱いをすることが可能となりました。

4. RTC(リアルタイムクロック)搭載

データロギングなど大抵のアプリケーションには、正しいシステム時刻が必要です。BH3 はバッテリでバックアップされた RTC を搭載しています。

5. 電源供給の安定化

Raspberry Pi は通常ミニ USB コネクタから電源を供給しますが、消費電流が増加したときに、このコネクタの内部抵抗が問題になり、電源が不安定になることがありました。BH3 は I/O バスポートから電源を供給していますので、この問題が発生しません。また、内部にレギュレーターを搭載し、安定した動作電圧を確保しています。

6. CPU の放熱対策

BH3 はケースに熱伝導パッドを装着し、これを CPU に密着させることにより、放熱対策をしています。50℃の外部環境での動作テストをクリアしていますので、フィールドで安心してご利用になれます。

7. ノイズ、静電対策

金属ケースの採用や、グランド処理などにより、静電気、雷サージ、GPIO ラインノイズなどの対策を施しています。

8. 電源断サポート

電源切断時に電源断検出(GPIO23)の信号を受けて高速シャットダウンが可能です。高速シャットダウン時に任意のコマンド(独自に作成したプログラムなど)を実行することが可能です。

9. ウォッチドッグタイマーの有効化

Raspberry Pi に搭載されたウォッチドッグタイマーを有効にし、万が一 CPU が暴走した場合でも、ハードウェア的に CPU をリブートさせます。システムがハングアップしたままになる可能性を減らすことが可能です。

10. ハード仕様

OS	Raspbian BH セットアップツール 別途提供
CPU	Raspberry Pi3 Model B 1.2GHz / 64-bit quad-core ARMv8 CPU
GPU	250 MHz / Broadcom VideoCore IV
メモリ	LPDDR2 SDRAM 1GB
RTC	MAXIM DS3231 I2C 接続（10KΩでプルアップ済み） GPIO2、3 を使用
UPS 機構	電気二重層コンデンサ
ビデオ出力	HDMI(1.3 / 1.4)
音声出力	3.5mm ジャック / HDMI
USB ポート	USB2.0×4
I/O コネクタ	40 ピン **重要：シリアル番号が BH3010000 から BH3019999 までの製品と、BH3020000 からの製品は、I/O コネクタのピンアサインが上下左右に反転しております。詳細は「I/O コネクタへのアクセスと GPIO ポート」をご覧ください。**
ネットワーク	10 / 100Mbps イーサネット 802.11n Wireless LAN
Bluetooth	Bluetooth 4.1 Bluetooth Low Energy (BLE)
カードスロット	Micro SD カードスロット
ストレージ	MicroSD カードに依存
電源	AC100-240V、50/60Hz（専用 AC アダプタ使用時） DC 7.5V 3A
外寸	高さ 32mm×幅 115mm×奥行き 115mm
重量	420g（AC アダプタ含まず）
動作保障温度	0 度～40 度（非結露）
動作保障湿度	10%～80%
電源断検出	GPIO23 L レベルで検出
リセットコントロール	GPIO18 を使用
汎用スイッチ	センター、上下左右モメンタリスイッチを搭載

11. 試験結果

試験場所:　　地方独立行政法人　北海道立総合研究機構　工業試験場
北海道札幌市北区北 19 条西 11 丁目

雷サージ許容度試験（試験装置: ノイズ研究所㈱LSS-15AX-C3）

試験内容	AC ラインより 1KV の雷サージを、正負 1 回ずつ、20 秒間隔で 5 回、合計 10 回与える。
試験結果	正常

試験内容	AC ラインより 2KV の雷サージを、正負 1 回ずつ、20 秒間隔で 5 回、合計 10 回与える。
試験結果	正常

静電気許容度試験（試験装置: ノイズ研究所㈱ESS-2000 & 放電ガン TC=815R）

試験内容	間接放電（本装置から 10 センチメートル離れたところで放電する）
試験結果	8KV * 20 回: 正常 15KV * 20 回: 正常

試験内容	直接放電（本装置に放電ガンを接触させて放電する）
試験結果	4KV * 20 回: 正常 8KV * 20 回: 正常

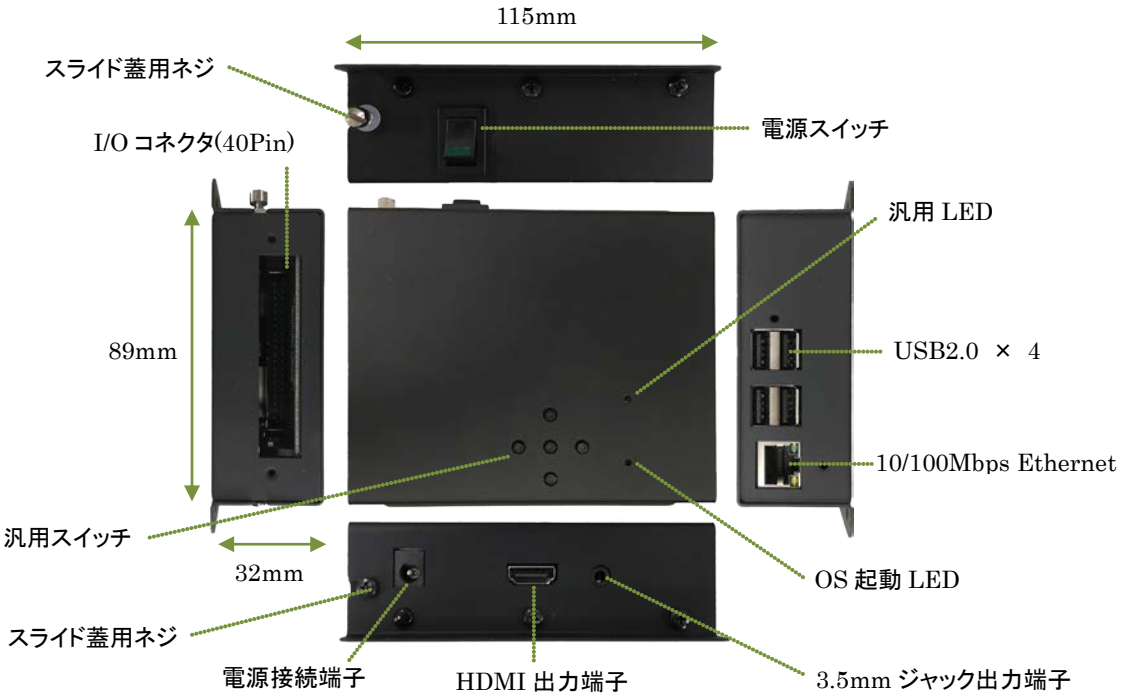
ただし間接、直接双方とも放電時にモニタ画面が一瞬乱れて 1 秒以内に正常に戻るという現象が見られた。本体リセットがかかったのではなく、高圧パルスによるノイズで HDMI の同期が一瞬だけ外れることが原因と推測する。

電子機器用低温恒温恒湿試験（試験装置: エスペック㈱PL-4KP）

試験内容	0℃において本装置を 24 時間連続運転する。
試験結果	正常

試験内容	50℃において本装置を 24 時間連続運転する。
試験結果	正常

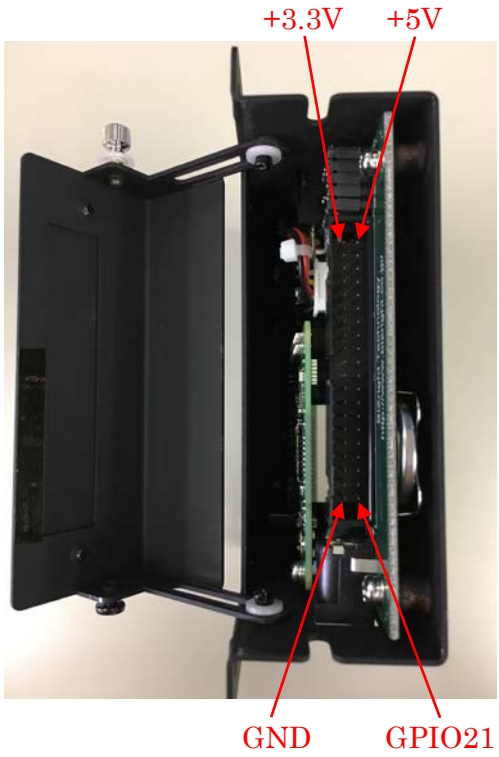
12. 外寸と各部名称

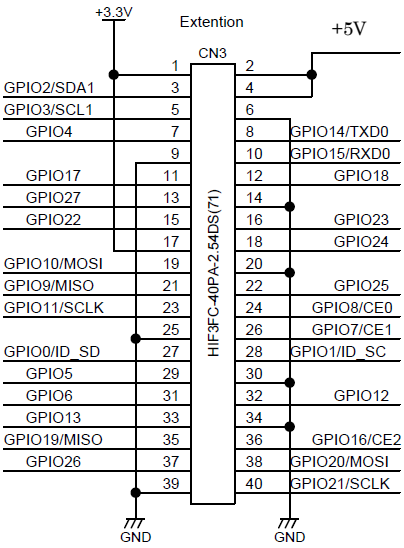


Micro SDカードは、スライド蓋を開けて、Raspberry Pi に直接挿します。

13. I/O コネクタへのアクセスと GPIO ポート

スライド蓋や、I/O コネクタの蓋をはずすと、I/O コネクタにアクセスできます。
GPIO ポート番号は下図のようになっております。
重要：本製品（シリアル番号が BH302000～）はピンアサインを BH3010000～BH3019999 までの製品から変更しております。





OS 起動時、I/O コネクタの 5V は ON になります。GPIO16 の値によって 5V を ON/OFF する事が可能です。

例(Wiring Pi の gpio コマンドを利用して I/O コネクタの 5V を ON/OFF する場合):

```
gpio -g mode 16 out
gpio -g write 16 1    （I/O コネクタの 5V を ON）
gpio -g write 16 0    （I/O コネクタの 5V を OFF）
```

14. 汎用ボタンの GPIO 割り当て

① UP ボタン: GPIO 17
② DOWN ボタン: GPIO 27
③ LEFT ボタン: GPIO 24
④ RIGHT ボタン: GPIO 22
⑤ ENTER ボタン: GPIO 25

- PULLUP モード(Raspberry Pi 内蔵のプルアップ抵抗を有効)にします
- 汎用ボタンは GPIO のピンと GND が接続されています
  (外付けのプルアップ抵抗は接続されていません)
- 押されていない場合は H レベル(1)、押された場合は L レベル(0)の値になります
  コマンドラインから値を取得する場合の例
  (Wiring Pi の gpio コマンドを利用して up ボタンの値を取得する場合):

```
gpio -g mode 17 input
gpio -g mode 17 up
gpio -g read 17
```

15. 汎用 LED の GPIO 割り当て

① 汎用 LED:　　GPIO 12

- 汎用 LED を利用する場合は、スライド蓋を開けて LED ジャンパピンを差し有効化します(詳細は「汎用スイッチや汎用 LED の無効化、リセットコントロール端子切替」をご覧ください)
- 汎用 LED を点灯する場合は L レベル(0)、消灯する場合は H レベル(1)の値にします

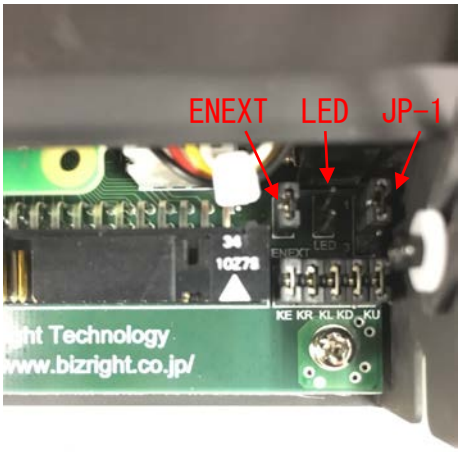
例(Wiring Pi の gpio コマンドを利用して汎用 LED を点灯、消灯する場合):

```
gpio -g mode 12 out
gpio -g write 12 0　（汎用 LED を点灯）
gpio -g write 12 1　（汎用 LED を消灯）
```

16. 汎用スイッチや汎用 LED の無効化、リセットコントロール端子切替

本製品では、ジャンパピンによって汎用スイッチ、汎用 LED、I/O コネクタの 5V の ON/OFF 切替を無効化し、割り当てられている GPIO ポートを I/O コネクタから利用することが可能です。リセットコントロール端子は、GPIO ポート番号を変更することが可能となっております。

ジャンパピンへのアクセスはスライド蓋を開けることによりアクセス可能です。



JP-1	出荷時はリセットコントロールと OS 起動 LED 制御を GPIO18 に設定(1 と 2 のピンを連結)しております。ジャンパピンの切り替えで GPIO5 に変更(2 と 3 のピンを連結)できます。
LED	ジャンパピンを差すと、汎用 LED を有効化します。 ジャンパピンを抜くと、汎用 LED を無効化し GPIO12 が利用できるようになります。
ENEXT	ジャンパピンを抜くと、I/O コネクタの 5V の ON/OFF 切替を無効化し GPIO16 が利用できるようになります。ジャンパピンを抜いた場合は I/O コネクタの 5V ピン(2,4)は常に ON になります。
KU	ジャンパピンを抜くと、UP ボタンを無効化し GPIO17 が利用できるようになります。
KD	ジャンパピンを抜くと、DOWN ボタンを無効化し GPIO27 が利用できるようになります。
KL	ジャンパピンを抜くと、LEFT ボタンを無効化し GPIO24 が利用できるようになります。
KR	ジャンパピンを抜くと、RIGHT ボタンを無効化し GPIO22 が利用できるようになります。
KE	ジャンパピンを抜くと、ENTER ボタンを無効化し GPIO25 が利用できるようになります。

JP-1 GPIO5 に変更した場合は、設定ファイル shutdownSensor.conf を /usr/local/sbin/以下に下記の内容で作成し、保存します。設定ファイルを保存後、OS の再起動が必要です。

/usr/local/sbin/shutdownSensor.conf

```
# Shutdown sensor config file

enrun pin = 5
```

17. OS について

Raspbian に対してセットアップツールをインストールします。

セットアップツールを利用すると、次の機能を追加します。

- 電源切断による高速シャットダウン
- OS 起動時に RTC 時刻をシステム時刻に設定
- ntpdate を利用して現在時刻を取得し RTC 時刻に設定
  (設定タイミング: ネットワーク接続時、毎時 30 分(1 時間毎))
- ウォッチドッグタイマーの有効化

セットアップツールは、以下からダウンロードできます。
http://dl.bizright.jp/bh/bh-tools-latest.tar.gz
（対応 OS: Raspbian 3.18 以上）

Raspbian 3.18 以上をインストール後、次のコマンドを実行します。

セットアップツールのインストールコマンド:

```
curl -O http://dl.bizright.jp/bh/bh-tools-latest.tar.gz
tar xfvz bh-tools-latest.tar.gz
cd bh-tools
./build　（※Installation of BH tools completed.が表示されれば成功）
sudo reboot
```

注意: build ファイル内で git や apt-get を呼び出しているためインターネットに接続できる状態で実行します。

既にセットアップツールを導入済みの方でバージョンアップする場合もこちらのコマンドを実行します。

一緒にインストールされるライブラリやパッケージ一覧:

- Wiring Pi
- ntpdate
- watchdog
- upstart(systemd がインストールされていない場合にインストールされる)
- git(git がインストールされていない場合にインストールされる)

セットアップツールをアンインストールする場合は次のコマンドを実行します。

セットアップツールのアンインストールコマンド:

cd bh-tools (※インストール時に利用したディレクトリに移動)
./build uninstall
(※Uninstall of BH tools completed.が表示されれば成功)
cd wiringPi
./build uninstall (※今後 Wiring Pi を利用しない場合)
sudo apt-get -y remove ntpdate (※今後 ntpdate を利用しない場合)
sudo apt-get -y remove watchdog (※今後 watchdog を利用しない場合)
sudo reboot -f

※セットアップツールでエラーが出てインストールできない場合

Error: WDT module is unknown.
のようなエラーが出てインストールが正常に行えない場合は、
apt-get update
apt-get upgrade
を行った後に、BH を再起動していない可能性があります。
sudo reboot
を行った後、再度セットアップツールのインストールを行ってください。

18. 高速シャットダウンの処理シーケンス

BH3 上での高速シャットダウンする際の処理シーケンスです。

1. BH3 の電源切断時に GPIO の 23 番（INPUT モード）が H レベル(1)から L レベル(0)になります
   ※AC アダプタ経由の電源を使わず、単三アルカリ電池 4 本での稼働で最大 30 秒間利用可能
2. 常駐プログラム「/usr/local/sbin/shutdownSensor」が 1.の割り込みを受けて次のスクリプトを実行します
   - /usr/local/sbin/fastshutdown
   - /usr/local/sbin/__fastdown-function(fastshutdown から呼ばれます)
3. 開始時間をログに出力します(高速シャットダウンの処理時間を計測するため)
   出力先: /var/log/fastshutdown.log
   Start time: (/proc/uptime の内容)
4. ディスプレイの信号を OFF にします
5. /usr/local/sbin/system-stop-use-usb スクリプトを実行します
   (10 秒経過するとタイムアウトとなり、強制終了します)
6. USB コントローラの電源を OFF にします(シャットダウン時の消費電力を減らすため、有線 LAN と USB に接続された機器を利用不可にします)
7. /usr/local/sbin/system-stop スクリプトを実行します
   (10 秒経過するとタイムアウトとなり、強制終了します)
8. 現在の時刻を保存します
   (RTC が接続されていない場合でも動作可能にするため)
9. 全てのプロセスを正常終了させます
10. メモリ上のデータの内容を強制的にディスクに書き込みます
11. 全てのプロセスを強制終了します
12. 終了時間をログに出力します(高速シャットダウンの処理時間を計測するため)
    出力先: /var/log/fastshutdown.log
    End time: (/proc/uptime の内容)
13. メモリ上のデータの内容を強制的にディスクに書き込みます
14. マウントされているすべてのファイルシステムを読み込み専用モードで再マウントします
15. GPIO の 23 番（INPUT モード）が H レベル(1)の場合はシステムを再起動、L レベル(0)の場合はシステムを停止します
    (ユーザが誤って AC 電源を切ってしまい、シャットダウンプロセスが走り始めたの

に AC 電源を復旧させたようなケースを想定)

高速シャットダウン時に任意のコマンド(独自に作成したプログラムなど)を実行したい場合は、次のスクリプトに任意のコマンドを追加します。

/usr/local/sbin/system-stop

任意のコマンド内でネットワークと通信する処理や USB を利用した処理(有線 LAN や無線 LAN の利用、USB メモリへのファイル保存など)を行う場合は、「/usr/local/sbin/system-stop」に記述せずに次のスクリプトに任意のコマンドを追加します。

/usr/local/sbin/system-stop-use-usb

system-stop スクリプトと system-stop-use-usb スクリプトは 10 秒以内に処理を完了してください。10 秒経過するとタイムアウトとなり、強制終了します。

タイムアウトの秒数は次のスクリプトで変更可能です。

/usr/local/sbin/__fastdown-function

```
timeout -s 9 タイムアウトの秒数 /usr/local/sbin/system-stop-use-usb
timeout -s 9 タイムアウトの秒数 /usr/local/sbin/system-stop
```

高速シャットダウン時に独自に作成したプログラム内でファイルのクローズ処理などを行う場合は、kill シグナルの割込みを受けてファイルのクローズ処理を行います。system-stop スクリプトまたは system-stop-use-usb スクリプトに killall コマンドなどを追加して、高速シャットダウン時に独自に作成したプログラムに対して kill シグナルが送られるようにします。

追加例:

/usr/local/sbin/system-stop

```
killall 独自に作成したプログラム名
```

OS 起動中に予期しないリセットが発生しないようにするため、常駐プログラム「/usr/local/sbin/shutdownSensor」の起動時(OS 起動時)に GPIO の 18 番(OUT モード)を H レベル(1)にし、シャットダウン完了までそのままの状態を保持しています。

19. RTC の設定シーケンス

OS 起動時(/usr/local/sbin/set-rtc):

1. RTC の I2C 接続を初期化します
2. RTC 時刻をシステム時刻に設定します
3. 2.の実行結果が失敗の場合、fake-hwclock を利用して前回のシャットダウン時刻をシステム時刻に設定します

ネットワーク接続時と毎時 30 分(1 時間毎))に実行(/etc/network/if-up.d/ntpdate):

1. 重複起動をチェックします(既に起動中の場合は処理を停止)
2. 20 秒間待機します(ネットワーク接続直後は ntpdate が失敗しやすいため)
3. /usr/sbin/ntpdate-debian コマンドを実行し、ネットワーク経由でシステム時刻を設定します
4. 3. の実行結果が成功の場合、システム時刻を RTC 時刻に設定します
5. 3. の実行結果が成功の場合、10 分間待機します
   (ntpdate が連続して実行されるのを防ぐため)

ntpdate の実行間隔は次のファイルで変更可能です。
/etc/cron.d/ntpdate(毎時 30 分(1 時間毎)の場合)

```
30 * * * * root /etc/network/if-up.d/ntpdate > /dev/null 2>&1
```

NTP サーバーを指定する場合は次のファイルを修正します。
/etc/default/ntpdate

```
NTPSERVERS="ntp.nict.jp ntp.jst.mfeed.ad.jp ntp.ring.gr.jp"
```

手動で nptdate を実行し、システム時刻を RTC に設定する場合は次のコマンドを実行します。

```
sudo /usr/sbin/ntpdate-debian
sudo hwclock -w
```

ネットワークに接続しない場合や ntpdate を利用してシステム時刻を合わせる必要がない場合は次のファイルを削除します。
/etc/cron.d/ntpdate
/etc/network/if-up.d/ntpdate

任意の時刻を RTC に設定する場合は次のコマンドを実行します。
sudo date -s '時刻(例: 2015-06-19 00:00:00)'
sudo hwclock -w

RTC 時刻を表示する場合は次のコマンドを実行します。
sudo hwclock -r

RTC 時刻をシステム時刻に設定する場合は次のコマンドを実行します。
sudo hwclock -s

20. ウォッチドッグタイマーのセットアップ内容

セットアップツールを利用した場合、次のコマンドを実行してウォッチドッグタイマーを有効化しています。watchdog のプログラムをインストールして設定ファイルを変更し、OS 起動時に bcm2708_wdog モジュール(Raspbian 4.2 以下の場合)または bcm2835_wdt モジュール(Raspbian 4.3 以上の場合)を自動的にロードします。

```
sudo apt-get install watchdog
sudo vi /etc/default/watchdog
        変更前:
                watchdog_module="none"
        変更後(Raspbian 4.2 以下の場合):
                watchdog_module="bcm2708_wdog"
        変更後(Raspbian 4.3 以上の場合):
                watchdog_module=" bcm2835_wdt "
sudo vi /etc/watchdog.conf
        変更前:
                #max-load-1= 24
                #watchdog-device = /dev/watchdog
        変更後:
                max-load-1= 24
                watchdog-device = /dev/watchdog
                watchdog-timeout = 10
sudo vi /etc/modules
        最終行に追加(Raspbian 4.2 以下の場合):
                bcm2708_wdog
        最終行に追加(Raspbian 4.3 以上の場合):
                bcm2835_wdt
sudo vi /lib/systemd/system/watchdog.service
        最終行に追加(ファイル内の[Install]の下に追加)
                WantedBy=multi-user.target
sudo update-rc.d watchdog enable
sudo reboot
```

21. 汎用スイッチを利用したプログラムのサンプルソース

汎用スイッチを押すとボタン名を表示するプログラムのサンプルソースです。
このサンプルソースは、以下からダウンロードできます。
http://dl.bizright.jp/bh/bhButton.zip

C 言語で Wiring Pi を利用した場合:
bhButton.c

```
#include <stdio.h>
#include <sys/time.h>
#include <wiringPi.h>

#define GPIO_UP 17
#define GPIO_DOWN 27
#define GPIO_RIGHT 22
#define GPIO_LEFT 24
#define GPIO_ENTER 25
#define BUTTON_USLEEP 50000
#define RESET_SEC 5

int setupGpio();
void isrUp();
void isrDown();
void isrRight();
void isrLeft();
void isrEnter();
void isrButton(int pin, char *button);

// 現在押されているボタン名
static volatile char *currentButton = NULL;

// 押された時間
static volatile long currentButtonTime = 0;

// メイン関数
int main(int argc, char *argv[]) {
```

```
	struct timeval now;

	// GPIO の初期化
	if (setupGpio() == -1) return 1;

	while (1) {
		sleep(1);

		// 現在、汎用ボタンが押されている場合
		// 他の汎用ボタンを最大 RESET_SEC(5 秒)間無効
		if (currentButton != NULL) {
			gettimeofday(&now, NULL);
			if (RESET_SEC < now.tv_sec - currentButtonTime) {
				currentButton = NULL;
			}
		}
	}
	return 0;
}

// GPIO の初期化関数
int setupGpio() {
	// WiringPi の初期化
	if (wiringPiSetupGpio() == -1) return -1;

	// 入力モードに設定
	pinMode(GPIO_UP, INPUT);
	pinMode(GPIO_DOWN, INPUT);
	pinMode(GPIO_RIGHT, INPUT);
	pinMode(GPIO_LEFT, INPUT);
	pinMode(GPIO_ENTER, INPUT);

	// PULLUP モードに設定
	pullUpDnControl(GPIO_UP, PUD_UP);
	pullUpDnControl(GPIO_DOWN, PUD_UP);
	pullUpDnControl(GPIO_RIGHT, PUD_UP);
```

```
        pullUpDnControl(GPIO_LEFT, PUD_UP);
        pullUpDnControl(GPIO_ENTER, PUD_UP);

        // GPIO の値の変化を検出し、割り込み処理を行う
        wiringPiISR(GPIO_UP, INT_EDGE_BOTH, &isrUp);
        wiringPiISR(GPIO_DOWN, INT_EDGE_BOTH, &isrDown);
        wiringPiISR(GPIO_RIGHT, INT_EDGE_BOTH, &isrRight);
        wiringPiISR(GPIO_LEFT, INT_EDGE_BOTH, &isrLeft);
        wiringPiISR(GPIO_ENTER, INT_EDGE_BOTH, &isrEnter);
        return 0;
}

// up ボタンの割込み関数
void isrUp() {
        isrButton(GPIO_UP, "up");
}

// down ボタンの割込み関数
void isrDown() {
        isrButton(GPIO_DOWN, "down");
}

// right ボタンの割込み関数
void isrRight() {
        isrButton(GPIO_RIGHT, "right");
}

// left ボタンの割込み関数
void isrLeft() {
        isrButton(GPIO_LEFT, "left");
}

// enter ボタンの割込み関数
void isrEnter() {
        isrButton(GPIO_ENTER, "enter");
}
```

```
// ボタンの割込み関数
void isrButton(int pin, char *button) {
        int read1, read2;
        struct timeval now;

        // チャタリングによる誤動作防止のため
        // BUTTON_USLEEP(0.05 秒)の間隔で GPIO の値を 2 回取得し
        // 一致する場合のみボタン名を表示
        read1 = digitalRead(pin);
        usleep(BUTTON_USLEEP);
        read2 = digitalRead(pin);
        if (read1 == 0 && read2 == 0) {
                if (currentButton == NULL) {
                        gettimeofday(&now, NULL);
                        currentButtonTime = now.tv_sec;
                        currentButton = button;
                        printf("%s¥n", button);
                }
        } else if (currentButton == button && read1 == 1 && read2 == 1) {
                currentButton = NULL;
        }
}
```

ビルドコマンド:

```
gcc -lwiringPi -o bhButton bhButton.c
```

実行コマンド:

```
sudo ./bhButton
```

Python で RPi.GPIO を利用した場合：

bhButton.py

```
# coding: UTF-8

import RPi.GPIO as GPIO
import time

GPIO_UP = 17
GPIO_DOWN = 27
GPIO_RIGHT = 22
GPIO_LEFT = 24
GPIO_ENTER = 25
BUTTON_SLEEP = 0.05

# ボタンの割込み関数
def gpio_callback(channel):

        # チャタリングによる誤動作防止のため
        # BUTTON_SLEEP(0.05 秒)の間隔で GPIO の値を 2 回取得し
        # 一致する場合のみボタン名を表示
        value1 = GPIO.input(channel)
        if value1 == 1:
                return
        time.sleep(BUTTON_SLEEP)
        value2 = GPIO.input(channel)
        if value2 == 1:
                return
        if value1 != value2:
                return

        if channel == GPIO_UP:
                print('up')
        elif channel == GPIO_DOWN:
                print('down')
        elif channel == GPIO_RIGHT:
                print('right')
```

```
        elif channel == GPIO_LEFT:
                print('left')
        elif channel == GPIO_ENTER:
                print('enter')

# 入力モード,PULLUP モードに設定
GPIO.setmode(GPIO.BCM)
GPIO.setup(GPIO_UP, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(GPIO_DOWN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(GPIO_RIGHT, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(GPIO_LEFT, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(GPIO_ENTER, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# GPIO の値の変化(1 から 0)を検出し、割り込み処理を行う
GPIO.add_event_detect(GPIO_UP, GPIO.FALLING)
GPIO.add_event_detect(GPIO_DOWN, GPIO.FALLING)
GPIO.add_event_detect(GPIO_RIGHT, GPIO.FALLING)
GPIO.add_event_detect(GPIO_LEFT, GPIO.FALLING)
GPIO.add_event_detect(GPIO_ENTER, GPIO.FALLING)
GPIO.add_event_callback(GPIO_UP, gpio_callback)
GPIO.add_event_callback(GPIO_DOWN, gpio_callback)
GPIO.add_event_callback(GPIO_RIGHT, gpio_callback)
GPIO.add_event_callback(GPIO_LEFT, gpio_callback)
GPIO.add_event_callback(GPIO_ENTER, gpio_callback)

while 1:
        time.sleep(1)
```

実行コマンド:

sudo python bhButton.py

22. 汎用 LED を利用したプログラムのサンプルソース

汎用 LED を点滅させるプログラムのサンプルソースです。
このサンプルソースは、以下からダウンロードできます。
http://dl.bizright.jp/bh/bhLED.zip

C 言語で Wiring Pi を利用した場合:
bhLED.c

```
#include <stdio.h>
#include <wiringPi.h>

#define GPIO_LED 12

int setupGpio();

// メイン関数
int main(int argc, char *argv[]) {

	// GPIO の初期化
	if (setupGpio() == -1) return 1;

	while (1) {

		// 汎用 LED を点灯
		digitalWrite(GPIO_LED, 0);

		// 0.5 秒スリープ
		delay(500);

		// 汎用 LED を消灯
		digitalWrite(GPIO_LED, 1);

		// 0.5 秒スリープ
		delay(500);

	}
```

```
        return 0;
}

// GPIO の初期化関数
int setupGpio() {
        // WiringPi の初期化
        if (wiringPiSetupGpio() == -1) return -1;

        //  出力モードに設定
        pinMode(GPIO_LED, OUTPUT);

        return 0;
}
```

ビルドコマンド:

```
gcc -lwiringPi -o bhLED bhLED.c
```

実行コマンド:

```
sudo ./bhLED
```

Python で RPi.GPIO を利用した場合：

bhLED.py

```
# coding: UTF-8

import RPi.GPIO as GPIO
import time

GPIO_LED = 12

# 出力モードに設定
GPIO.setmode(GPIO.BCM)
GPIO.setup(GPIO_LED, GPIO.OUT)

try:
        while 1:

                # 汎用 LED を点灯
                GPIO.output(GPIO_LED, 0)

                # 0.5 秒スリープ
                time.sleep(0.5)

                # 汎用 LED を消灯
                GPIO.output(GPIO_LED, 1)

                # 0.5 秒スリープ
                time.sleep(0.5)

finally:
        # GPIO の設定をリセット
        GPIO.cleanup()
```

実行コマンド：

sudo python bhLED.py

23. 改訂履歴

更新日	バージョン	内容
2018/3/28	1.0	初版