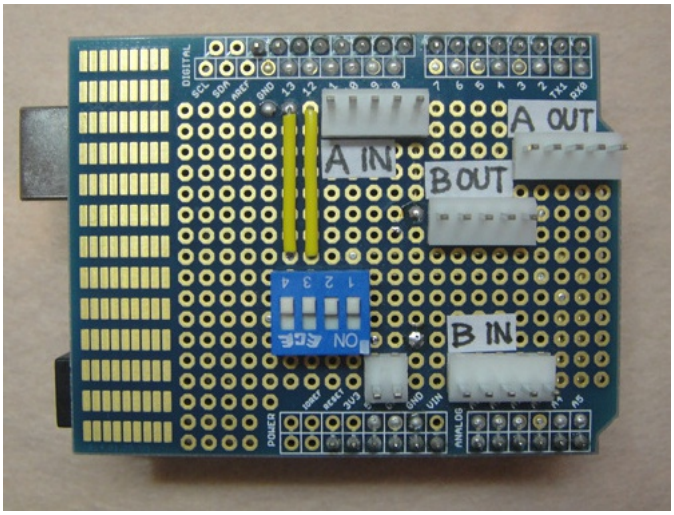
レバーディレクションモデファイヤ
技術資料

8 方向レバーの入力を 4 方向レバーの入力へ変換します。斜め入力を上下左右に変換することで、ドラゴンバスターやアサルトのような 4 方向レバーで遊ぶゲームを、8 方向レバーで遊ぶことができます。

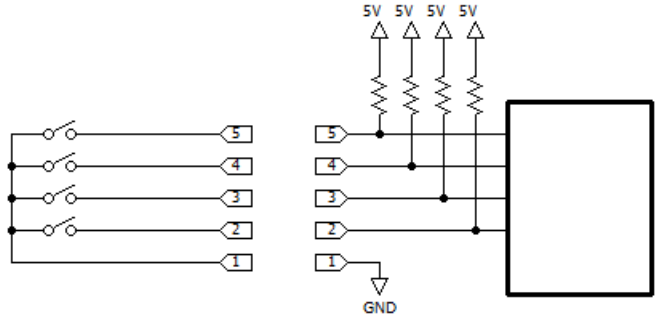


本資料は、オープンソースハードウェア、オープンソースソフトウェア等の、無償公開されている資料を元に作成させていただきました。感謝いたします。

2018 年 10 月 29 日 初版
Ise, Kazumasa
Twitter: @kaz_ise
Website: http://magicpuppet.org
Mail: kzms.ise@gmail.com

レバーの基本的な仕組み

8 方向レバーと 4 方向レバーの基本的な仕組みは同じで、上下左右の入力に対応するスイッチが存在します。スイッチの一端は共通（コモン）端子に接続されており、スイッチが ON のときコモン端子と短絡します。通常は接続される側で信号線をプルアップし、コモン端子は GND に接続します。そうすることでスイッチが OFF のとき HIGH (5V)となり、ON のとき LOW (GND)となります。



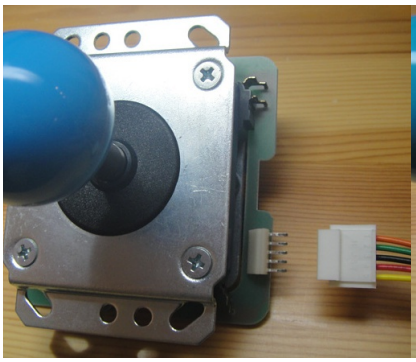
8 方向レバーの斜め入力では、上下のどちらかと、左右のどちらかの 2 つのスイッチが ON になります。例えば、右上の入力では、上と右のスイッチが ON になります。

```
右上入力     上と右 ON
□□■          ■
□○□   ＝   □○■
□□□          □
```

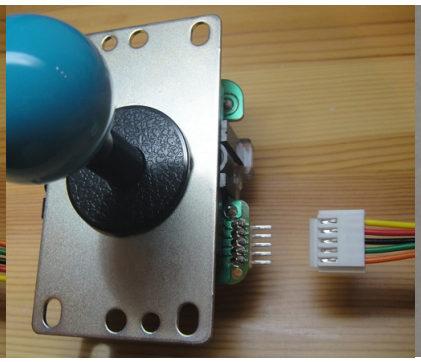
4 方向レバーでは、ガイドによって斜め方向にレバーが入らないようになっています。

レバーの信号線

セイミツ工業 LS-32-01 や、三和電子 JLF-TP-8YT などのレバーは、レバーのスイッチが基板に取り付けられており、専用のハーネスで接続するようになっています。
接続に使用されているコネクターは、いずれも NH コネクターであり、ピンアサインも互換性があります。



セイミツ工業 LS-32-01



三和電子 JLF-TP-8YT

セイミツ工業 H5P

以下はセイミツ工業 H5P ハーネスの色です。

```
１ ２ ３ ４ ５
橙 緑 黒 赤 黄
```

1 がコモンで、2~5 が上下左右の信号線です。信号線は、レバーの取り付け方によって、方向が変わります。レバーを上から見たときのコネクターの位置によって、レバーの方向は以下のようになります。

```
コネクター：右          コネクター：左          コネクター：上          コネクター：下
                                                    ■
   □□□                     ■□□□                □□□                    □□□
   □○□                       □○□                □○□                    □○□
   □□□■                      □□□                □□□                    □□□
                                                                           ■
１ ２ ３ ４ ５          １ ２ ３ ４ ５          １ ２ ３ ４ ５          １ ２ ３ ４ ５
Ｃ 左 右 下 上          Ｃ 右 左 上 下          Ｃ 下 上 右 左          Ｃ 上 下 左 右
```

以降は、コネクターの位置を右として説明します。

レバー入力信号の変換

レバーのスイッチは OFF のとき HIGH (5V)となり、ON のとき LOW (GND)となります。反転信号を取得することで、OFF: 0, ON: 1 となります。レバーの上下左右の入力を 4 ビットの値に割り当てると、8 方向の値は以下のようになります。

```
b3 b2 b1 b0       1000: 上,    0100: 下,    0010: 右,    0001: 左
上 下 右 左       1010: 右上,  0110: 右下,  1001: 左上,  0101: 左下
```

8 方向の値を 4 方向の値とするには、ビットをマスクします。上下ビットを 1 としたビットマスクを用いて 8 方向の値をマスクすることで、斜め入力の値を上下入力の値とすることができます。同様に、左右ビットを 1 としたビットマスクを用いることで、斜め入力の値を左右入力の値とすることができます。
例えば、右上の値を上下ビットマスクでマスクすると上の値となり、左右ビットマスクでマスクすると右の値となります。

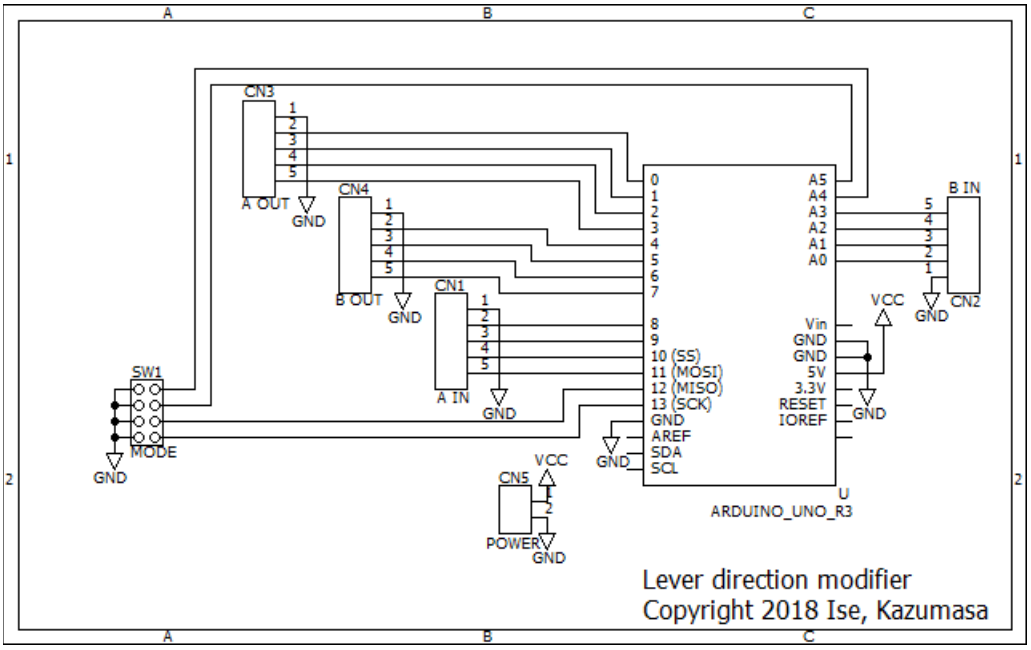
```
    1010: 右上                    1010: 右上
AND 1100: 上下ビットマスク    AND 0011: 左右ビットマスク
  = 1000: 上                    = 0010: 右
```

出力信号の生成

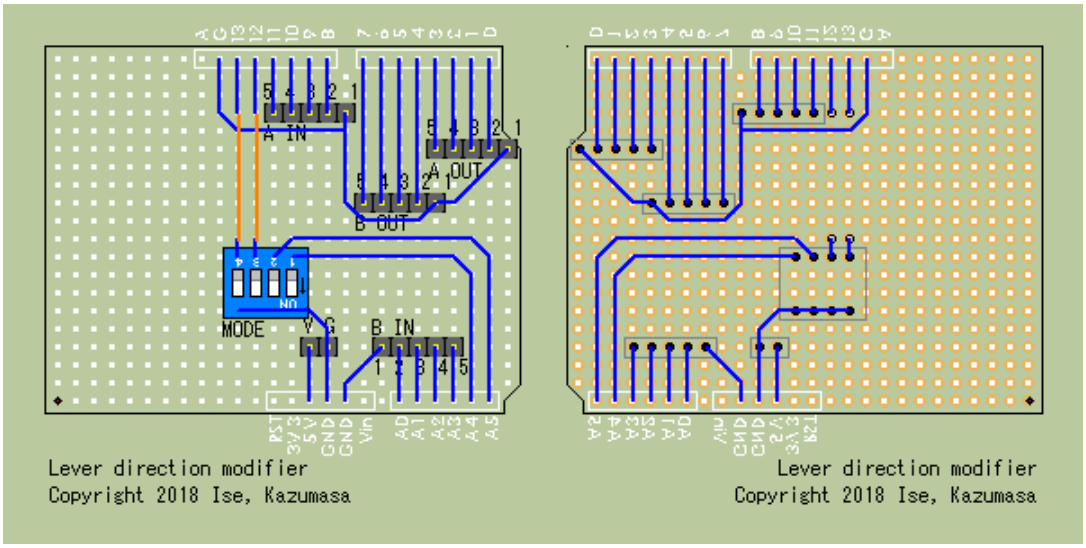
レバーのスイッチの信号を Arduino で生成することを考えます。
スイッチが OFF の場合の信号は、どこにも接続されていない状態（ハイ・インピーダンス）であるため、端子を INPUT に指定します。
スイッチが ON の場合の信号は、信号線がプルアップされているとすると、GND であるため、端子を OUTPUT LOW に指定します。
レバー入力の値は、反転信号を取得することで、OFF: 0, ON: 1 となります。この値をポートの方向レジスタに指定することで、端子を OFF: INPUT, ON: OUTPUT LOW に指定できます。

回路図

Arduino を用いて製作しました。レバーを 2 本接続できます。レバーの入出力信号のピンアサインは、レバーハーネスと同じです。DIP スイッチで動作を変更できます。



シールド基板部品配置図



DIP スイッチ設定

□OFF, ■ON

4 3 2 1
□□□□ 入力と同じ出力
□□□■ 斜め入力 OFF モード
□□■□ 上下優先モード
□□■■ 左右優先モード
□■□□ 4 方向保持モード
□■□■ 4 方向更新モード
■□■□ 上下優先ディレイインプットモード (Ver.2) ※
■□■■ 左右優先ディレイインプットモード (Ver.2) ※

※ドラゴンバスターのためのスペシャルモード

斜め入力 OFF モード

斜め入力を OFF にするモードです。斜め入力で上下左右のスイッチが OFF となり、レバーがニュートラルと同じ状態になります。

右入力		右 ON		右上入力		OFF		上入力		上 ON
□□□		□		□□■		□		□■□		■
□○■	＝	□○■	⇒	□○□	＝	□○□	⇒	□○□	＝	□○□
□□□		□		□□□		□		□□□		□

上下優先モード

上下を優先するモードです。斜め入力で上下のスイッチだけを ON にします。

右入力		右 ON		右上入力		上 ON		上入力		上 ON
□□□		□		□□■		■		□■□		■
□○■	＝	□○■	⇒	□○□	＝	□○□	⇒	□○□	＝	□○□
□□□		□		□□□		□		□□□		□

左右優先モード

左右を優先するモードです。斜め入力で左右のスイッチだけを ON にします。

右入力		右 ON		右上入力		右 ON		上入力		上 ON
□□□		□		□□■		□		□■□		■
□○■	＝	□○■	⇒	□○□	＝	□○■	⇒	□○□	＝	□○□
□□□		□		□□□		□		□□□		□

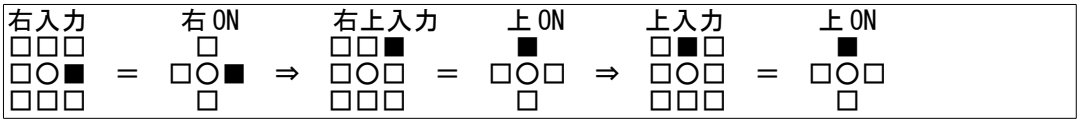
4 方向保持モード

4 方向を保持するモードです。斜め入力で直前の 4 方向入力を保持し、上下あるいは左右のスイッチだけを ON にします。

右入力		右 ON		右上入力		右 ON		上入力		上 ON
□□□		□		□□■		□		□■□		■
□○■	＝	□○■	⇒	□○□	＝	□○■	⇒	□○□	＝	□○□
□□□		□		□□□		□		□□□		□

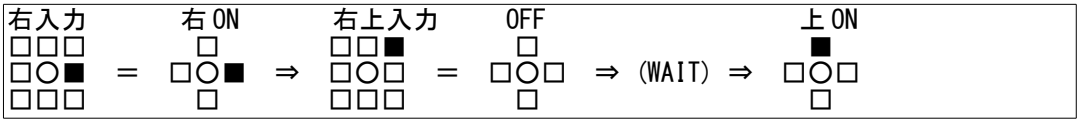
4 方向更新モード

4 方向を更新するモードです。斜め入力で直前の 4 方向入力を更新し、上下あるいは左右のスイッチだけを ON にします。



上下優先ディレイインプットモード (Ver.2)

ドラゴンバスターのためのスペシャルモードです。斜め入力で斜めジャンプを行うことができます。斜め入力を OFF にした一定時間後、上下のスイッチを ON にします。



左右優先ディレイインプットモード (Ver.2)

ドラゴンバスターのためのスペシャルモードです。レバーの取り付け方によっては、上下と左右が入れ替わるため、上下優先ディレイインプットモードの代わりに使用します。

ソースコード

```
// Lever direction modifier
//
// Copyright 2018 Ise, Kazumasa
// Released under the MIT license
// https://opensource.org/licenses/MIT
//
// 2018-07-04 1.0 Created.

#define LOW_2BITS(x)      ((x) & B00000011)
#define LOW_4BITS(x)      ((x) & B00001111)
#define MASK_LOW_2BITS(x) ((x) & B11111100)
#define MASK_LOW_4BITS(x) ((x) & B11110000)
#define MERGE_2BITS(hi, lo) LOW_4BITS(MASK_LOW_2BITS(hi) | LOW_2BITS(lo))
#define MERGE_4BITS(hi, lo)          (MASK_LOW_4BITS(hi) | LOW_4BITS(lo))
#define SET_LOW_4BITS(dst, src)  ((dst) = MERGE_4BITS(dst, src))
#define SET_HIGH_4BITS(dst, src) ((dst) = MERGE_4BITS(src, dst))

#define UP B00001000
#define DN B00000100
#define LT B00000001
#define RT B00000010
#define UD (UP | DN)
#define LR (LT | RT)

#define MODE MERGE_2BITS(~PINB >> 2, ~PINC >> 4)
#define DIRECTION_BITS_A LOW_4BITS(~PINB)
#define DIRECTION_BITS_B LOW_4BITS(~PINC)
#define SET_DIRECTION_BITS_A(x) SET_LOW_4BITS(DDRD, x)
#define SET_DIRECTION_BITS_B(x) SET_HIGH_4BITS(DDRD, (x) << 4)

class DirMask {
public:
```

```
 uint8_t ModifyDirectionBits(uint8_t mask, uint8_t bits) {
  switch (bits) {
  case UP:
  case DN:
  case LT:
  case RT:
   return bits;
  default:
   return bits & mask;
  }
 }
};

class DirHold {
private:
 uint8_t mask;
public:
 DirHold() : mask(UD) {}
 uint8_t ModifyDirectionBits(bool hold, uint8_t bits) {
  switch (bits) {
  case UP:
  case DN:
   mask = hold ? UD : LR;
   return bits;
  case LT:
  case RT:
   mask = hold ? LR : UD;
   return bits;
  default:
   return bits & mask;
  }
 }
};

class DirMod {
private:
 DirMask dirMask;
 DirHold dirHold;
public:
 uint8_t ModifyDirectionBits(uint8_t mode, uint8_t bits) {
  switch (mode) {
  case B00000001: return dirMask.ModifyDirectionBits(0, bits);
  case B00000010: return dirMask.ModifyDirectionBits(UD, bits);
  case B00000011: return dirMask.ModifyDirectionBits(LR, bits);
  case B00000100: return dirHold.ModifyDirectionBits(true, bits);
  case B00000101: return dirHold.ModifyDirectionBits(false, bits);
  default: return bits;
```

```
  }
 }
};

DirMod dirModA;
DirMod dirModB;

ISR(PCINT0_vect) {
 uint8_t bits = dirModA.ModifyDirectionBits(MODE, DIRECTION_BITS_A);
 SET_DIRECTION_BITS_A(bits);
}

ISR(PCINT1_vect) {
 uint8_t bits = dirModB.ModifyDirectionBits(MODE, DIRECTION_BITS_B);
 SET_DIRECTION_BITS_B(bits);
}

void setup() {
 PORTB  |= B00111111; // PB0~5 Pullup
 PORTC  |= B00111111; // PC0~5 Pullup
 PCMSK0 |= B00001111; // Enable PCINT0~3  (PB0~3)
 PCMSK1 |= B00001111; // Enable PCINT8~11 (PC0~3)
 PCICR  |= B00000011; // Enable PCI0~1
}

void loop() {
}
```

LOW_2BITS(x)
x の下位 2 ビットの値を取得します。
LOW_4BITS(x)
x の下位 4 ビットの値を取得します。
MASK_LOW_2BITS(x)
x の下位 2 ビットをマスクした値を取得します。
MASK_LOW_4BITS(x)
x の下位 4 ビットをマスクした値を取得します。
MERGE_2BITS(hi, lo)
lo の下位 2 ビットと、hi の下位 4 ビットの上位 2 ビットを合成した値を取得します。
MERGE_4BITS(hi, lo)
lo の下位 4 ビットと、hi の上位 4 ビットを合成した値を取得します。
SET_LOW_4BITS(dst, src)
dst に対し、src の下位 4 ビットを設定します。
SET_HIGH_4BITS(dst, src)
dst に対し、src の上位 4 ビットを設定します。
UP, DN, LT, RT
レバーの上下左右の入力に対応する各ビットの値です。
UD, LR

上下ビットマスクと、左右ビットマスクの値です。

MODE

DIP スイッチ設定の値を、ポート操作により一括で取得します。
反転信号を取得することで、OFF: 0, ON: 1 とします。

DIRECTION_BITS_A

レバー A の入力信号を、ポート操作により一括で取得します。
反転信号を取得することで、OFF: 0, ON: 1 とします。

DIRECTION_BITS_B

レバー B の入力信号を、ポート操作により一括で取得します。
反転信号を取得することで、OFF: 0, ON: 1 とします。

SET_DIRECTION_BITS_A(x)

レバー A の出力信号を、ポート操作により一括で出力します。

SET_DIRECTION_BITS_B(x)

レバー B の出力信号を、ポート操作により一括で出力します。

DirMask

レバー入力の値をマスクするクラスです。

uint8_t DirMask::ModifyDirectionBits(uint8_t mask, uint8_t bits)

レバー入力の値 bits の処理結果を取得します。
bits が上下左右の入力の値であった場合、そのままの値を取得します。
bits が斜め入力の値であった場合、mask でマスクした結果を取得します。

DirHold

直前の 4 方向入力の値を保持または更新するマスクの値によって、レバー入力の値をマスクするクラスです。

DirHold::mask

直前の 4 方向入力の値に応じて設定するマスクの値です。

DirHold::DirHold()

DirHold クラスを初期化します。

uint8_t DirHold::ModifyDirectionBits(bool hold, uint8_t bits)

レバー入力の値 bits の処理結果を取得します。
hold が true であれば状態保持となり、false であれば状態更新となります。
bits が上下の入力の値であった場合、状態保持であれば上下マスクを、状態更新であれば左右マスクを DirHold::mask に設定します。
bits が左右の入力の値であった場合、状態保持であれば左右マスクを、状態更新であれば上下マスクを DirHold::mask に設定します。
bits が上下左右の入力の値であった場合、そのままの値を取得します。
bits が斜め入力の値であった場合、DirHold::mask でマスクした結果を取得します。

DirMod

レバー入力の値を変換するクラスです。

DirMod::dirMask

DirMask クラスのインスタンスを保持します。

DirMod::dirHold

DirHold クラスのインスタンスを保持します。

uint8_t DirMod::ModifyDirectionBits(uint8_t mode, uint8_t bits)

レバー入力の値 bits の処理結果を取得します。
mode が 0001 であれば、斜め入力 OFF モードの処理結果を取得します。
mode が 0010 であれば、上下優先モードの処理結果を取得します。
mode が 0011 であれば、左右優先モードの処理結果を取得します。
mode が 0100 であれば、4 方向保持モードの処理結果を取得します。
mode が 0101 であれば、4 方向更新モードの処理結果を取得します。
mode が上記以外であれば、そのままの値を取得します。

ISR(PCINT0_vect)

ピンチェンジ割り込みの割り込みハンドラです。
レバー A の入出力信号を処理します。

ISR(PCINT1_vect)

ピンチェンジ割り込みの割り込みハンドラです。
レバー B の入出力信号を処理します。

void setup()

初期化を行います。
入力ピンをポート操作により一括でプルアップします。
レバーの入力信号のピンチェンジ割り込みを設定します。

ソースコード Ver.2

ドラゴンバスターのためのスペシャルモードを追加したバージョンです。

```
// Lever direction modifier
//
// Copyright 2018 Ise, Kazumasa
// Released under the MIT license
// https://opensource.org/licenses/MIT
//
// 2018-10-22 2.0 Add delay input mode.
// 2018-07-04 1.0 Created.

#define LOW_2BITS(x)      ((x) & B00000011)
#define LOW_4BITS(x)      ((x) & B00001111)
#define MASK_LOW_2BITS(x) ((x) & B11111100)
#define MASK_LOW_4BITS(x) ((x) & B11110000)
#define MERGE_2BITS(hi, lo) LOW_4BITS(MASK_LOW_2BITS(hi) | LOW_2BITS(lo))
#define MERGE_4BITS(hi, lo)          (MASK_LOW_4BITS(hi) | LOW_4BITS(lo))
#define SET_LOW_4BITS(dst, src)  ((dst) = MERGE_4BITS(dst, src))
#define SET_HIGH_4BITS(dst, src) ((dst) = MERGE_4BITS(src, dst))

#define UP B00001000
#define DN B00000100
#define LT B00000001
#define RT B00000010
#define UD (UP | DN)
#define LR (LT | RT)
```

```
#define MODE MERGE_2BITS(~PINB >> 2, ~PINC >> 4)
#define DIRECTION_BITS_A LOW_4BITS(~PINB)
#define DIRECTION_BITS_B LOW_4BITS(~PINC)
#define SET_DIRECTION_BITS_A(x) SET_LOW_4BITS(DDRD, x)
#define SET_DIRECTION_BITS_B(x) SET_HIGH_4BITS(DDRD, (x) << 4)
#define DELAY 17

class DirMask {
public:
 uint8_t ModifyDirectionBits(uint8_t mask, uint8_t bits) {
  switch (bits) {
  case UP:
  case DN:
  case LT:
  case RT:
   return bits;
  default:
   return bits & mask;
  }
 }
};

class DirHold {
private:
 uint8_t mask;
public:
 DirHold() : mask(UD) {}
 uint8_t ModifyDirectionBits(bool hold, uint8_t bits) {
  switch (bits) {
  case UP:
  case DN:
   mask = hold ? UD : LR;
   return bits;
  case LT:
  case RT:
   mask = hold ? LR : UD;
   return bits;
  default:
   return bits & mask;
  }
 }
};

class PollingTimer {
private:
 uint32_t target;
public:
 PollingTimer() : target(0xFFFFFFFF) {}
 void Set(uint32_t timeout) { target = millis() + timeout - 1; }
```

```
 void Reset() { target = 0xFFFFFFFF; }
 bool Check() { return target < millis(); }
};

class DirMod {
private:
 DirMask dirMask;
 DirHold dirHold;
public:
 PollingTimer pollingTimer;
 uint8_t ModifyDirectionBits(uint8_t mode, uint8_t bits) {
  switch (mode) {
  case B00000001: return dirMask.ModifyDirectionBits(0, bits);
  case B00000010: return dirMask.ModifyDirectionBits(UD, bits);
  case B00000011: return dirMask.ModifyDirectionBits(LR, bits);
  case B00000100: return dirHold.ModifyDirectionBits(true, bits);
  case B00000101: return dirHold.ModifyDirectionBits(false, bits);
  case B00001010:
  case B00001011:
   pollingTimer.Set(DELAY);
   return dirMask.ModifyDirectionBits(0, bits);
  default: return bits;
  }
 }
 uint8_t ModifyDirectionBitsDelay(uint8_t mode, uint8_t bits) {
  pollingTimer.Reset();
  switch (mode) {
  case B00001010: return dirMask.ModifyDirectionBits(UD, bits);
  case B00001011: return dirMask.ModifyDirectionBits(LR, bits);
  default: return bits;
  }
 }
};

DirMod dirModA;
DirMod dirModB;

ISR(PCINT0_vect) {
 uint8_t bits = dirModA.ModifyDirectionBits(MODE, DIRECTION_BITS_A);
 SET_DIRECTION_BITS_A(bits);
}

ISR(PCINT1_vect) {
 uint8_t bits = dirModB.ModifyDirectionBits(MODE, DIRECTION_BITS_B);
 SET_DIRECTION_BITS_B(bits);
}

void setup() {
```

```
 PORTB  |= B00111111; // PB0~5 Pullup
 PORTC  |= B00111111; // PC0~5 Pullup
 PCMSK0 |= B00001111; // Enable PCINT0~3  (PB0~3)
 PCMSK1 |= B00001111; // Enable PCINT8~11 (PC0~3)
 PCICR  |= B00000011; // Enable PCI0~1
}

void loop() {
 if (dirModA.pollingTimer.Check()) {
  uint8_t bits = dirModA.ModifyDirectionBitsDelay(MODE, DIRECTION_BITS_A);
  SET_DIRECTION_BITS_A(bits);
 }
 if (dirModB.pollingTimer.Check()) {
  uint8_t bits = dirModB.ModifyDirectionBitsDelay(MODE, DIRECTION_BITS_B);
  SET_DIRECTION_BITS_B(bits);
 }
}
```

変更を加えた部分について説明します。

DELAY
ディレイインプットモードのディレイ時間を指定します。
ここでは約 1 フレームの時間である 17msec を指定しています。
PollingTimer
ポーリングタイマークラスです。
PollingTimer::target
タイムアウトの目標時刻です。
PollingTimer::PollingTimer()
PollingTimer クラスを初期化します。
void PollingTimer::Set(uint32_t timeout)
タイムアウト時間を指定し、ポーリングタイマーをセットします。
void PollingTimer::Reset()
ポーリングタイマーをリセットします。
bool PollingTimer::Check()
ポーリングタイマーをチェックし、タイムアウト時間が経過したかを確認します。
DirMod::pollingTimer
PollingTimer クラスのインスタンスを保持します。
uint8_t DirMod::ModifyDirectionBits(uint8_t mode, uint8_t bits)
mode が 1010 または 1011 であれば、ポーリングタイマーをセットし、斜め入力 OFF モードの処理結果を取得します。
uint8_t DirMod::ModifyDirectionBitsDelay(uint8_t mode, uint8_t bits)
レバー入力の値 bits のディレイインプットモードの処理結果を取得します。
ポーリングタイマーをリセットします。
mode が 1010 であれば、上下優先モードの処理結果を取得します。

mode が 1011 であれば、左右優先モードの処理結果を取得します。
mode が上記以外であれば、そのままの値を取得します。

void loop()

レバーＡとレバーＢのポーリングタイマーをチェックし、タイムアウト時間が経過していたならば、ディレイインプットモード処理を行います。

部品

秋月電子通商

Arduino Uno Rev3 ￥2940
Arduino 用ユニバーサル プロトシールド基板 ￥200
ピンヘッダ 1×40 (40P) ￥35
DIP スイッチ 4P ￥50
耐熱通信機器用ビニル電線 2m×10 色 導体径 0.65mm 単芯 ￥620
スズメッキ線(0.6mm 10m) ￥210
はんだ 0.8mm ￥210
耐熱電子ワイヤー 2m×7 色 外径 1.22mm(UL3265 AWG24) ￥480

マルツ

NH コネクター 2.5mm ピッチベース付ポストトップ型 2 極(10 個入)【B2P-SHF-1AA*10】 ￥230
NH コネクター 2.5mm ピッチベース付ポストトップ型 5 極(10 個入)【B5P-SHF-1AA*10】 ￥480
NH コネクター 2.5mm ピッチハウジング 2 極(10 個入)【H2P-SHF-AA*10】 ￥230
NH コネクター 2.5mm ピッチハウジング 5 極(10 個入)【H5P-SHF-AA*10】 ￥350
SM コネクタ用ソケットコンタクト (100 個入)【BHF-001T-0.8BS】 ￥646

ソフトウェア

Arduino IDE: Arduino - Software
http://arduino.cc/en/Main/Software
文書作成: ホーム | LibreOffice - オフィススイートのルネサンス
https://ja.libreoffice.org/
回路図: 水魚堂の回路図エディタ
http://www.suigyodo.com/online/schsoft.htm
シールド基板部品配置図: Ｐａｓ Ｓ
http://www.geocities.jp/uaubn/pass/
プロトシールドの PasS 用データ: BANANAWANI MICOM. CULB: Arduino
http://bananawani-mc.blogspot.jp/2010/10/arduinopass.html

工具

太洋電機産業 goot 温調はんだこて PX-201(70W)
太洋電機産業 goot 替こて先 PX-2RT-3CR
太洋電機産業 goot 水がいらない こて先クリーナー ST-40
エンジニア 精密圧着ペンチ PA-21
VESSEL ワイヤーストリッパー No.3500E-2
トップ工業 ラジオペンチ RA3-150
太洋電機産業 goot 精密ニッパー フラットカット YN-10