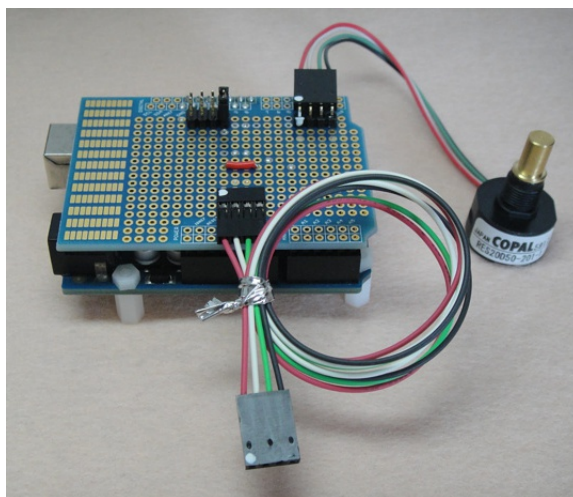


ロータリエンコーダパルスマルチプライヤ 技術資料

ロータリエンコーダのパルスをソフトウェア的に倍増し、一回転毎のパルス数が比較的少ないロータリエンコーダを、アーケード基板で使用できるようにします。

また、ロータリエンコーダの信号が、プルアップかプルダウンかに関係なく使用できるようにします。トラックボールにも使用できます。



本資料は「クリエイティブ・コモンズ 表示-継承 3.0 非移植 (CC BY-SA 3.0)」として公開します。<https://creativecommons.org/licenses/by-sa/3.0/deed.ja>

本資料に掲載されているプログラムのソースコードは MIT License として公開します。<http://opensource.org/licenses/mit-license.php>

これらのライセンスを要約すると次のようになります。

本資料は無償で利用できます。本資料は無保証です。

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。

2016年05月22日 初版

2016年05月28日 ロータリエンコーダに関する記述追加

2016年10月22日 ジャンパピンに関する記述変更 Rev.2.0に関する記述追加
Kazumasa ISE

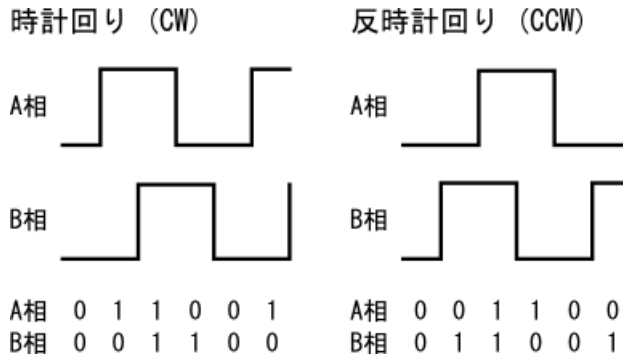
Twitter: @kaz_ise

<http://magicpuppet.org>

Mail: kzms.ise@gmail.com

ロータリエンコーダの回転方向の検出

ロータリエンコーダには、A相とB相の信号があり、B相の信号は、A相から1/4周期位相がずれています。



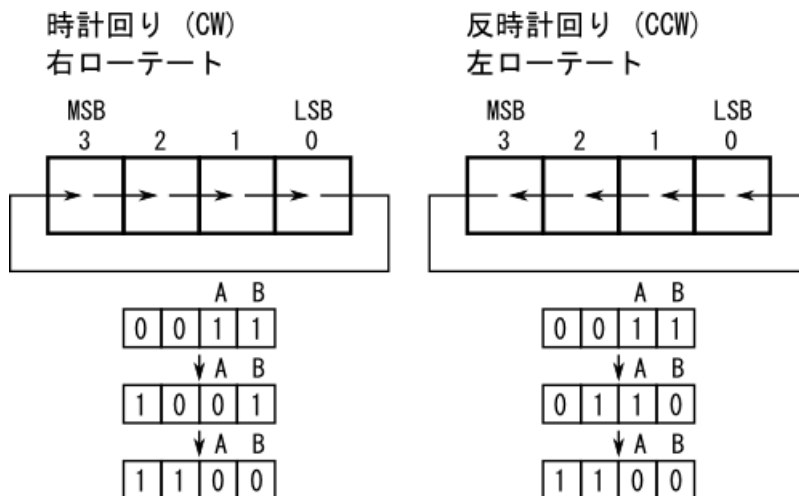
A相の信号と、信号が変化する直前のB相の信号を比較すると、時計回り (CW) では常に異なり、反時計回り (CCW) では常に同じとなっています。

A相を上位ビット、B相を下位ビットとする2ビットの値を考えると、信号が変化する直前の下位ビットと、信号が変化した直後の上位ビットの排他的論理和 (XOR) は、CWでは常に1、CCWでは常に0です。

これにより、回転の方向を検出することができます。

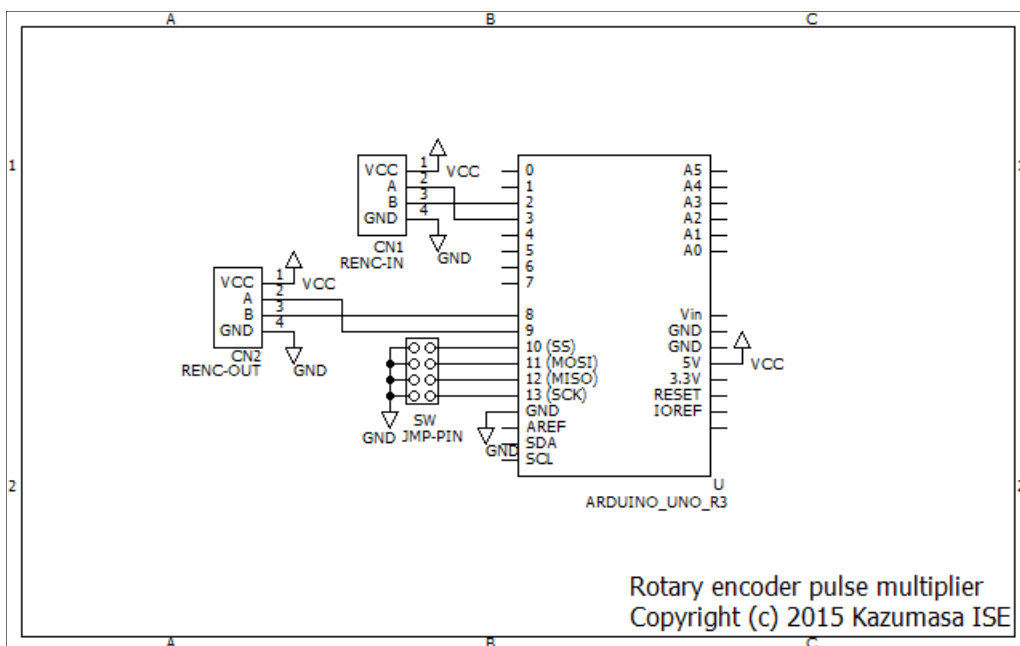
ロータリエンコーダの信号生成

4ビットの値をビットローテーションすることにより、下位2ビットにA相とB相の信号の値が得られます。時計回り (CW) は右ローテート、反時計回り (CCW) は左ローテートです。ロータリエンコーダの回転を検出する毎に、指定した回数の信号生成を行うことで、パルスを倍増することができます。



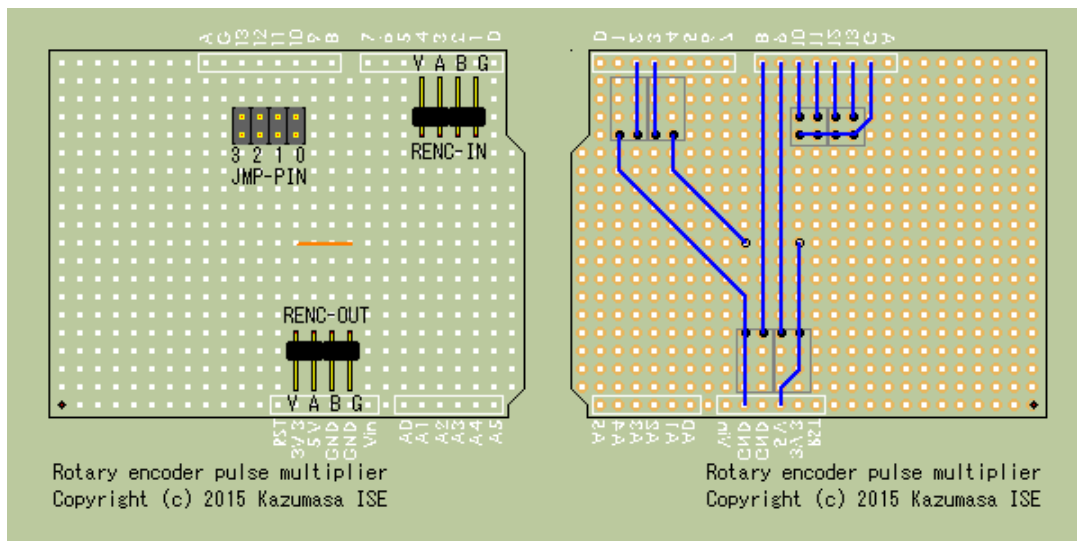
回路図

Arduino を用いて製作しました。Arduino の電源は 5V で、信号出力側から供給されるものとします。ロータリエンコーダの電源も共通です。ロータリエンコーダの信号線は、ロータリエンコーダ側でプルアップ/プルダウンなどにより、論理レベルが確定しているものとします。



シールド基板部品配置図

ジャンパピンで倍率 (x1~15) を変更できます。ジャンパピンは倍率の値である 4 ビットの数値を表しており、ショートの状態が 1 です。



ジャンパピンの設定が無い場合、デフォルト倍率 (x9) となります。

□□□□	□□□■	□□■□	□□■■
□□□□=0000= x9	□□□■=0001= x1	□□■□=0010= x2	□□■■=0011= x3
□■□□	□■□■	□■■□	□■■■
□■□□=0100= x4	□■□■=0101= x5	□■■□=0110= x6	□■■■=0111= x7
■□□□	■□□■	■□■□	■□■■
■□□□=1000= x8	■□□■=1001= x9	■□■□=1010=x10	■□■■=1011=x11
■■□□	■■□■	■■■□	■■■■
■■□□=1100=x12	■■□■=1101=x13	■■■□=1110=x14	■■■■=1111=x15

ロータリエンコーダ

入力側に使用するロータリエンコーダは、日本電産コパル電子株式会社製ロータリエンコーダ RES20D-50-201-1 が好適です。このロータリエンコーダは、PS2 専用ローリングスイッチのローリングスイッチユニットである「新方式ローリングスイッチ改」にも使用しています。



RES20D-50-201-1



新方式ローリングスイッチ改

ロータリエンコーダパルスマルチプライヤを使用することで、新方式ローリングスイッチ改をロストワールド基板で使用できるようになります。また、RES20D-50-201-1 をパドルコントローラとしてアルカノイド基板で使用できるようになります。

ソースコード

```
// Rotary encoder pulse multiplier
//
// Copyright (c) 2015 Kazumasa ISE
// Released under the MIT license
// http://opensource.org/licenses/mit-license.php
//
// PULSE_RATIO PB2~5
// RENC_IN      A:PD3(INT1), B:PD2(INT0)
// RENC_OUT     A:PB1,      B:PB0
```

```

#include <util/delay.h>
#include <avr/sleep.h>

//#define DEBUG
#define LOW_2BIT(x)      ((x) & B00000011)
#define LOW_4BIT(x)      ((x) & B00001111)
#define ROT_4BIT_RIGHT(x) LOW_4BIT(((x) >> 1) | ((x) << 3))
#define ROT_4BIT_LEFT(x)  LOW_4BIT(((x) << 1) | ((x) >> 3))

#define RENC_IN          LOW_2BIT(PIND >> 2)
#define SET_RENC_OUT(x) (PORTB = (PORTB & B11111100) | LOW_2BIT(x))

#define PULSE_INTERVAL 32
#define RATIO_DEF       9
#define RATIO_VAL       LOW_4BIT(~PINB >> 2)
#define PULSE_RATIO     ((RATIO_VAL > 0) ? RATIO_VAL : RATIO_DEF)

inline bool rotationDirection(byte curr) {
    static byte prev = 0;
    const bool cw = ((prev << 1) ^ curr) & B00000010;
    prev = curr;
    return !cw;
}

inline void rotationPulseOut(bool ccw, byte ratio) {
    static byte bits = B00000011;
    for (int i = 0; i < ratio; i++) {
        bits = ccw ? ROT_4BIT_LEFT(bits) : ROT_4BIT_RIGHT(bits);
        SET_RENC_OUT(bits);
        _delay_us(PULSE_INTERVAL);
    }
}

#ifdef DEBUG
volatile byte COUNT;
#endif

void interruptHandler() {
    const bool ccw = rotationDirection(RENC_IN);
    rotationPulseOut(ccw, PULSE_RATIO);
#ifdef DEBUG
    ccw ? COUNT++ : COUNT--;
#endif
}

void setup() {
    set_sleep_mode(SLEEP_MODE_IDLE);
    PORTC |= B00111111; // Unused PC0~5 pullup
    PORTD |= B11110011; // Unused PD0~1, PD4~7 pullup
    PORTB |= B00111100; // PB2~5 Input pullup
    DDRB  |= B00000011; // PB0~1 Output
    attachInterrupt(0, interruptHandler, CHANGE);
    attachInterrupt(1, interruptHandler, CHANGE);
#ifdef DEBUG
    Serial.begin(115200);
#endif
}

```

```
void loop() {
  sleep_mode();
#ifdef DEBUG
  static byte prev = 0;
  if (prev != COUNT) {
    Serial.println(COUNT);
    prev = COUNT;
  }
#endif
}
```

LOW_2BIT()

下位 2 ビットの値を取得します。

LOW_4BIT()

下位 4 ビットの値を取得します。

ROT_4BIT_RIGHT()

4 ビットの値を右ローテートした値を取得します。

ROT_4BIT_LEFT()

4 ビットの値を左ローテートした値を取得します。

RENC_IN

ロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で取得します。

SET_RENC_OUT()

生成したロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で出力します。

PULSE_INTERVAL

生成するロータリエンコーダのパルスの 1/4 周期に相当する時間を、マイクロ秒単位で指定します。ここでは 32μsec を指定しています。

RATIO_DEF

生成するロータリエンコーダのパルスの倍率のデフォルト値を指定します。ここでは x9 を指定しています。

RATIO_VAL

生成するロータリエンコーダのパルスの倍率の値を、ジャンパピンの設定から、ポート操作により一括で取得します。

PULSE_RATIO

生成するロータリエンコーダのパルスの倍率の値を取得します。ジャンパピンの設定が無い場合、デフォルト値を使用します。

rotationDirection()

ロータリエンコーダの回転方向を取得します。
直前の A 相 B 相の値を保持し、回転の方向を検出します。
CCW なら true を、CW なら false を出力します。

rotationPulseOut()

生成したロータリエンコーダの信号を出力します。
4 ビットの値を保持し、指定した倍率の値に従って信号を生成し、出力します。

interruptHandler()

割り込みハンドラです。

ロータリエンコーダの回転方向を取得し、生成したロータリエンコーダの信号を出力します。

コメントアウトされている DEBUG を有効にした場合、検出したロータリエンコーダのパルスをカウントします。

setup()

初期化を行います。

消費電力低減のためのスリープモードを設定します。

未使用ピンをプルアップします。

ロータリエンコーダの A 相 B 相の信号ピンを設定します。

生成したロータリエンコーダの信号の A 相 B 相の信号ピンを設定します。

割り込みハンドラを設定します。

コメントアウトされている DEBUG を有効にした場合、ログ出力の準備をします。

loop()

スリープモードへ移行します。ロータリエンコーダの信号を検出すると、スリープモードから復帰します。

コメントアウトされている DEBUG を有効にした場合、ログを出力します。

注意点

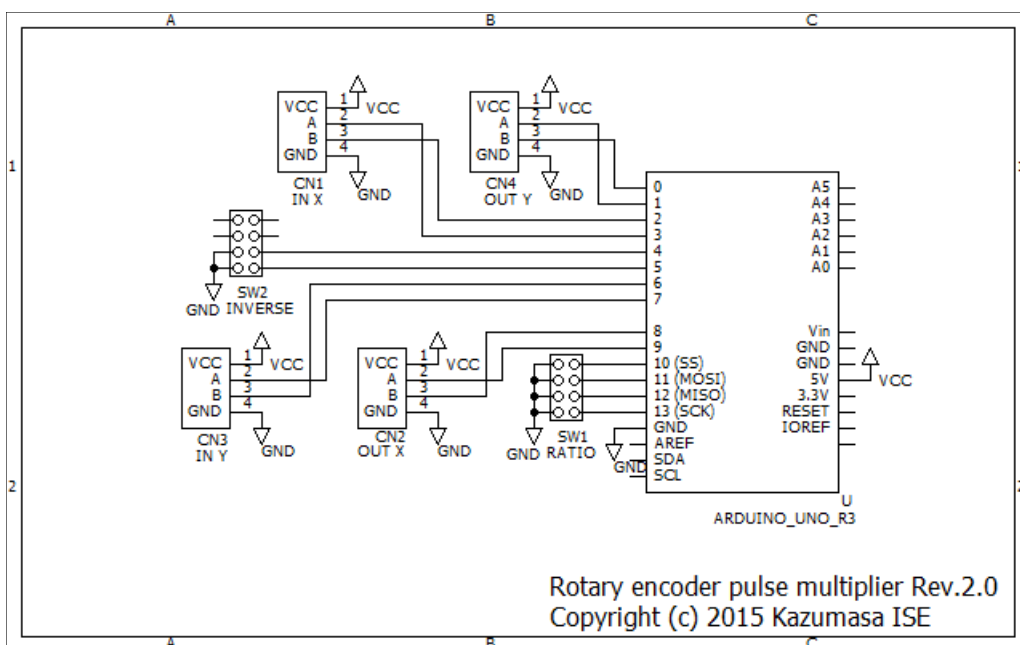
ロータリエンコーダを高速で回転した場合、処理が間に合わない可能性があります。生成するロータリエンコーダのパルスの 1/4 周期に相当する時間は、既定では 32μsec です。RES20D-50-201-1 を使用した場合、倍率を最大の x15 に設定すると、

$$50\text{pulse} \times 4 \times 32\mu\text{sec} \times 15 = 96\text{msec}$$

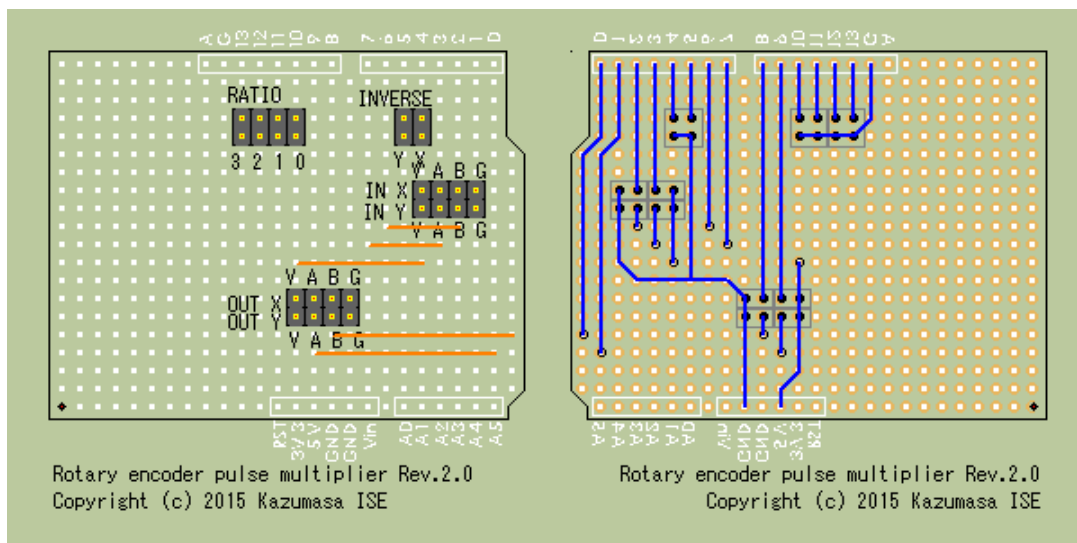
となり、一回転での処理時間は、約 0.1 秒となります。これを超える速度で使用する場合、PULSE_INTERVAL の値を 32μsec より小さい値に変更して下さい。

回路図 Rev.2.0

入出力を追加し、2つの信号を同時に処理できるようにしました。
また、信号反転用ジャンパピンを追加し、回転方向を反転できるようにしました。



シールド基板部品配置図 Rev.2.0



ソースコード Rev.2.0

```
// Rotary encoder pulse multiplier
```



```

//
// Copyright (c) 2015 Kazumasa ISE
// Released under the MIT license
// http://opensource.org/licenses/mit-license.php
//
// 2016-08-23 2.0 Add another rotary encoder input.
// 2016-08-22 1.1 Add INVERSE_ROT.
// 2016-05-22 1.0 Created.
//
// PULSE_RATIO    PB2-5
// INVERSE_ROT_X  PD4
// INVERSE_ROT_Y  PD5
// RENC_IN_X      A:PD3(INT1), B:PD2(INT0)
// RENC_IN_Y      A:PD7,      B:PD6
// RENC_OUT_X     A:PB1,      B:PB0
// RENC_OUT_Y     A:PD1,      B:PD0

#include <avr/sleep.h>

#define LOW_1BIT(x)      ((x) & B00000001)
#define LOW_2BIT(x)      ((x) & B00000011)
#define LOW_4BIT(x)      ((x) & B00001111)
#define ROT_4BIT_RIGHT(x) LOW_4BIT(((x) >> 1) | ((x) << 3))
#define ROT_4BIT_LEFT(x)  LOW_4BIT(((x) << 1) | ((x) >> 3))
#define SIGN(x)          (((x) < 0) ? -1 : ((x) > 0) ? 1 : 0)

#define RENC_IN_X        LOW_2BIT(PIND >> 2)
#define RENC_IN_Y        LOW_2BIT(PIND >> 6)
#define SET_RENC_OUT_X(x) (PORTB = (PORTB & B11111100) | (x))
#define SET_RENC_OUT_Y(x) (PORTD = (PORTD & B11111100) | (x))

#define PULSE_INTERVAL  32
#define RATIO_DEF        9
#define RATIO_VAL        LOW_4BIT(~PINB >> 2)
#define PULSE_RATIO      ((RATIO_VAL > 0) ? RATIO_VAL : RATIO_DEF)

#define INVERSE_ROT_X    LOW_1BIT(~PIND >> 4)
#define INVERSE_ROT_Y    LOW_1BIT(~PIND >> 5)

class RotaryEncoder {
private:
    int8_t countBuffer;
    uint8_t prevAB;
    uint8_t signalBits;

public:
    RotaryEncoder() {
        countBuffer = 0;
        prevAB = 0;
        signalBits = B00000011;
    }

    inline void countPulse(uint8_t currAB, uint8_t ratio, bool inverse) {
        if (prevAB != currAB) {
            const bool cw = ((prevAB << 1) ^ currAB) & B00000010;
            const int8_t signum = (inverse ? !cw : cw) ? -1 : 1;
            countBuffer += signum * ratio;
        }
    }
};

```

```

    prevAB = currAB;
}
}

inline uint8_t getSignalBits() {
    if (countBuffer != 0) {
        const int8_t signum = SIGN(countBuffer);
        signalBits = (signum > 0) ? ROT_4BIT_LEFT(signalBits) :
ROT_4BIT_RIGHT(signalBits);
        countBuffer -= signum;
    }
    return LOW_2BIT(signalBits);
}

inline bool isCountBufferEmpty() const {
    return countBuffer == 0;
}
};

RotaryEncoder rencX;
RotaryEncoder rencY;

void interruptHandler() {
    rencX.countPulse(RENC_IN_X, PULSE_RATIO, INVERSE_ROT_X);
}

ISR(PCINT2_vect) {
    rencY.countPulse(RENC_IN_Y, PULSE_RATIO, INVERSE_ROT_Y);
}

void setup() {
    set_sleep_mode(SLEEP_MODE_IDLE);
    PORTC |= B00111111; // PC0~5 Pullup
    PORTD |= B00110000; // PD4~5 Pullup
    PORTB |= B00111100; // PB2~5 Pullup
    DDRB |= B00000011; // PB0~1 Output
    DDRD |= B00000011; // PD0~1 Output
    PCMSK2 |= B11000000; // Enable PCINT22~23
    PCICR |= B00000100; // Enable PCI2
    attachInterrupt(0, interruptHandler, CHANGE);
    attachInterrupt(1, interruptHandler, CHANGE);
}

void loop() {
    while (!rencX.isCountBufferEmpty() ||
        !rencY.isCountBufferEmpty())
    {
        SET_RENC_OUT_X(rencX.getSignalBits());
        SET_RENC_OUT_Y(rencY.getSignalBits());
        delayMicroseconds(PULSE_INTERVAL);
    }
    sleep_mode();
}

```

LOW_1BIT()

下位1ビットの値を取得します。

LOW_2BIT()

下位 2 ビットの値を取得します。

LOW_4BIT()

下位 4 ビットの値を取得します。

ROT_4BIT_RIGHT()

4 ビットの値を右ローテートした値を取得します。

ROT_4BIT_LEFT()

4 ビットの値を左ローテートした値を取得します。

SIGN()

符号関数です。

RENC_IN_X

ロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で取得します。

RENC_IN_Y

ロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で取得します。

SET_RENC_OUT_X()

生成したロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で出力します。

SET_RENC_OUT_Y()

生成したロータリエンコーダの A 相 B 相の信号の値を、ポート操作により一括で出力します。

PULSE_INTERVAL

生成するロータリエンコーダのパルスの 1/4 周期に相当する時間を、マイクロ秒単位で指定します。ここでは 32 μ sec を指定しています。

RATIO_DEF

生成するロータリエンコーダのパルスの倍率のデフォルト値を指定します。ここでは x9 を指定しています。

RATIO_VAL

生成するロータリエンコーダのパルスの倍率の値を、ジャンパピンの設定から、ポート操作により一括で取得します。

PULSE_RATIO

生成するロータリエンコーダのパルスの倍率の値を取得します。ジャンパピンの設定が無い場合、デフォルト値を使用します。

INVERSE_ROT_X

回転方向を反転するかどうかを、ジャンパピンの設定から取得します。

INVERSE_ROT_Y

回転方向を反転するかどうかを、ジャンパピンの設定から取得します。

RotaryEncoder

ロータリエンコーダの入出力の信号を処理し、状態を保持するクラスです。未出力のロータリエンコーダのパルス、信号が変化する直前の A 相 B 相の値、ロータリエンコーダの信号を生成するための 4 ビットの値を保持します。

RotaryEncoder::RotaryEncoder()

RotaryEncoder クラスを初期化します。

RotaryEncoder::countPulse()

ロータリエンコーダの回転方向を取得し、指定した倍率の値に従ってパルスをカウントし、カウントバッファに加算します。

RotaryEncoder::getSignalBits()

ロータリエンコーダの A 相 B 相の信号の値を取得します。

ロータリエンコーダの信号を生成するための 4 ビットの値を更新し、カウントバッファを減算します。

RotaryEncoder::isCountBufferEmpty()

未出力のロータリエンコーダのパルスが無い事を確認します。

カウントバッファが 0 であれば true を返します。

interruptHandler()

割り込みハンドラです。

指定した倍率の値に従ってロータリエンコーダのパルスをカウントします。

ISR()

割り込みハンドラです。

指定した倍率の値に従ってロータリエンコーダのパルスをカウントします。

setup()

初期化を行います。

消費電力低減のためのスリープモードを設定します。

未使用ピンをプルアップします。

ロータリエンコーダの A 相 B 相の信号ピンを設定します。

生成したロータリエンコーダの信号の A 相 B 相の信号ピンを設定します。

ピンチェンジ割り込みを設定します。

割り込みハンドラを設定します。

loop()

未出力のロータリエンコーダのパルスがあれば、ロータリエンコーダの信号を出力します。無ければ、スリープモードへ移行します。

ロータリエンコーダの信号を検出すると、スリープモードから復帰します。

注意点

入力側にロータリエンコーダを接続しない場合、ノイズにより誤動作する可能性がありますので、A 相 B 相の信号を 5V か GND に接続してください。ジャンパピンを使用して、A 相を 5V に、B 相を GND に接続するのが手軽で良いと思います。

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) ￥40

ピンヘッダ 2×40 (80P) ￥50

ジャンパーピン黒(2.54mm ピッチ)(25 個入) ￥100

TJC8 ピンターミナル(メス) (10 本入) ￥50×2

耐熱電子ワイヤー 2m×7 色 外径 1.22mm(UL3265 AWG24) ￥480

スズメッキ線(0.6mm 10m) ￥210

鉛フリーハンダ 0.8mm ￥280

千石電商

【QI コネクタ】 信号伝達コネクタ (黒) 1×4 2550-1×4 ￥21×3

2545-1×40 ピンヘッダ 1 列×40P (L 型・標準ピッチ) ￥84

ビットレードワン

日本電産コパル電子 ロータリエンコーダ[RES20D-50-201-1] ￥1250

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/>

図形描画: Draw Freely. | Inkscape

<http://www.inkscape.org/ja/>

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: P a s S

<http://www.geocities.jp/uabn/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/arduinoypass.html>

工具

太陽電機産業 goot ニクロムはんだこて KS-30R(30W)

太陽電機産業 goot こて先クリーナー ST-30

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

太陽電機産業 goot 精密ニッパー フラットカット YN-10