ロータリエンコーダ

3I14 公文健太

実験の目的

ロータリエンコーダは軸の回転方向および回転角度を検出するセンサである。ここでは 2 相の信号を出力するインクリメント型ロータリエンコーダを用い、マニュアルの読み方、使用方法を学ぶ。

インターフェースの理解

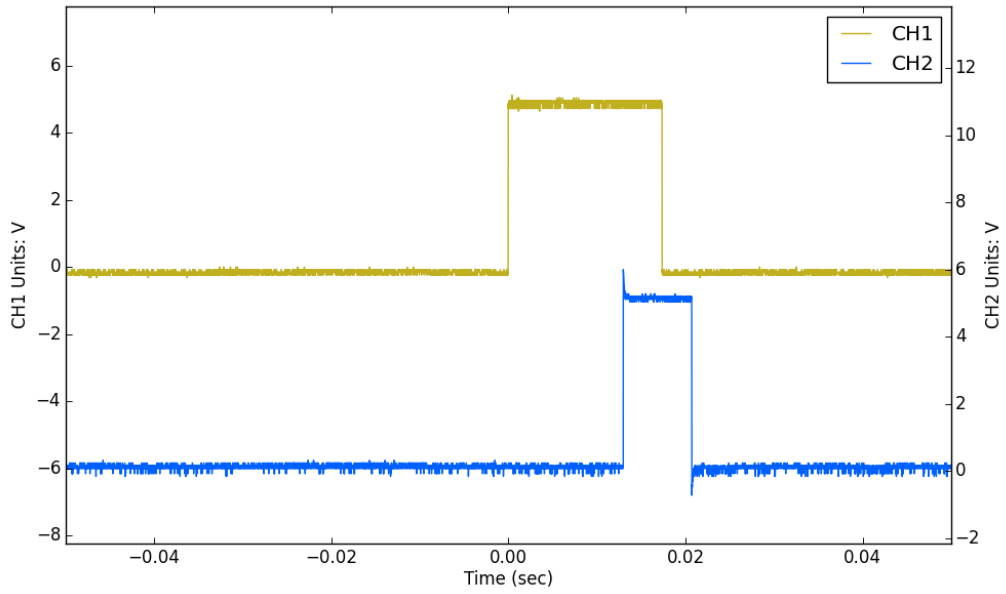
実験1

1. 右回転するときは、 A 端子が先に H レベルになり、続いて B 端子が H レベルになる。
2. 逆に左回転するときは、 B 端子が先に H レベルになり、続いて A 端子が Hレベルになる。

実験2

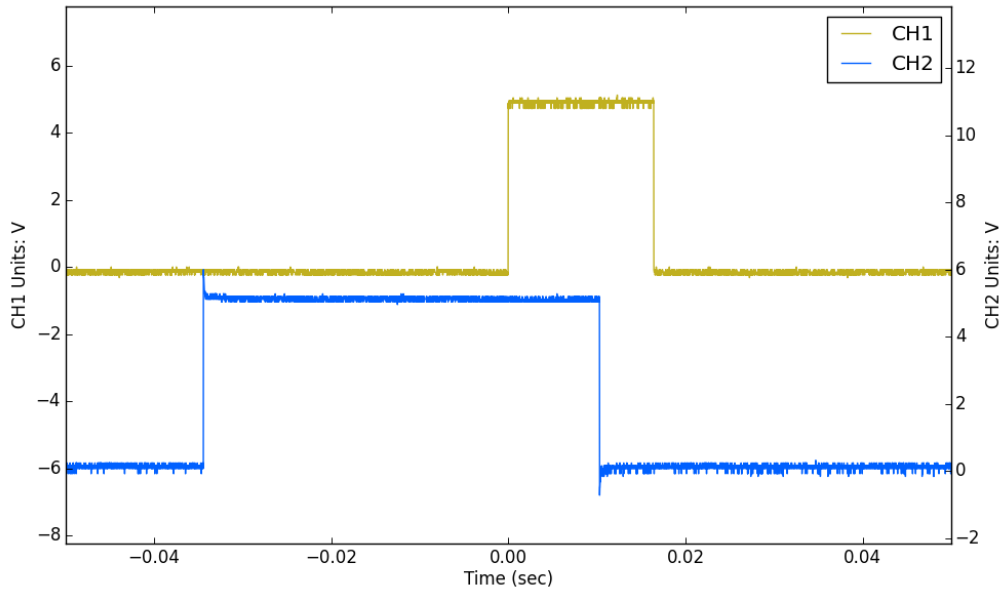
(a)

- オシロスコープ



(b)

- オシロスコープ



オシロスコープより、右回転は `(a)` である。

実験3

1.

A	B	rotary_q1	rotary_q2
L	L	0	-
L	H	-	1
H	L	-	0
H	H	1	-

2. 回転方向検出：rotary_q1 の立ち上がりにおいて、rotary_q2 が `1` なら右回転、 `0` なら左回転である。

プログラムの作成

実験4

- コード

```
#include <asf.h>

void io_init(void);
void loop(void);

int output_A = 0;
int output_B = 0;
int current_rotary_q1 = 0;
int previous_rotary_q1 = 0;
int rotary_q2 = 0;

int main(void)
{
```

```
    io_init();

    while (true)
    {
        loop();
    }
}

void io_init(void)
{
    DDRD = 0b00000000;
    DDRB = 0b00100000;
    PORTB = 0b00000000;
}

void loop(void)
{
    // 入力
    output_A = (PIND & (1 << PIND2)) ? 1 : 0;
    output_B = (PIND & (1 << PIND3)) ? 1 : 0;
    if (output_A == 0 && output_B == 0) current_rotary_q1 = 0;
    if (output_A == 1 && output_B == 1) current_rotary_q1 = 1;
    if (output_A == 1 && output_B == 0) rotary_q2 = 1;
    if (output_A == 0 && output_B == 1) rotary_q2 = 0;

    // 状態が変化したら
    if (current_rotary_q1 != previous_rotary_q1) {
        // 立ち上がりを検出したら
        if (current_rotary_q1 == 1)
        {
            // 右回転なら
            if (rotary_q2 == 1) {
                // 点灯
                PORTB = 0b00100000;
            }
            // 左回転なら
            else {
                // 消灯
                PORTB = 0b00000000;
            }
        }
        previous_rotary_q1 = current_rotary_q1;
    }
}
```

- 動作確認パターン
    - 右回転したらLEDが点灯するか。
    - 左回転したらLEDが消灯するか。
- 結果
    - 点灯した。

    - 消灯した。

実験5

- コード

```
#include <asf.h>

void io_init(void);
void loop(void);

int output_A = 0;
int output_B = 0;
int current_rotary_q1 = 0;
int previous_rotary_q1 = 0;
int rotary_q2 = 0;
int right_count = 0;

int main(void)
{
    io_init();

    while (true)
    {
        loop();
    }
}

void io_init(void)
{
    DDRD = 0b00000000;
    DDRB = 0b00100000;
    PORTB = 0b00000000;
}

void loop(void)
{
    // 入力
    output_A = (PIND & (1 << PIND2)) ? 1 : 0;
    output_B = (PIND & (1 << PIND3)) ? 1 : 0;
    if (output_A == 0 && output_B == 0) current_rotary_q1 = 0;
    if (output_A == 1 && output_B == 1) current_rotary_q1 = 1;
    if (output_A == 1 && output_B == 0) rotary_q2 = 1;
    if (output_A == 0 && output_B == 1) rotary_q2 = 0;

    // 状態が変化したら
    if (current_rotary_q1 != previous_rotary_q1) {
        // 立ち上がりを検出したら
        if (current_rotary_q1 == 1)
        {
            // 右回転なら
            if (rotary_q2 == 1) {
```

```
                right_count++;
                if (right_count == 3)
                {
                    PORTB = 0b00100000;
                }
                else if (right_count >= 4)
                {
                    PORTB = 0b00000000;
                    right_count = 0;
                }
            }
            // 左回転なら
            else {
                PORTB = 0b00000000;
                right_count = 0;
            }
        }
        previous_rotary_q1 = current_rotary_q1;
    }
}
```

- リスタート
  リスタートでは、右に回した回数が4回以上ならPORTB5から0を出力しLEDを消灯し、まわした回数を0回にリセットする。

- 動作確認パターン
  - 右に3クリック回したらPORTB5から1が出力されLEDが点灯するか。
  - 左に回したらカウントがリセットされるか。
  - 右に4クリック以上回したらカウントがリセットされるか。

- 結果
  - 点灯した。
  - リセットされた。
  - リセットされた。

実験6

- コード

```
#include <asf.h>
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);
void loop(void);
void start(void);

int output_A = 0;
int output_B = 0;
```

```
int current_rotary_q1 = 0;
int previous_rotary_q1 = 0;
int rotary_q2 = 0;
int right_count = 0;
int left_count = 0;
enum Phase { right = 1, left = 2 };
int phase = right;

int main(void)
{
    io_init();
    start();

    while (true)
    {
        loop();
    }
}

void io_init(void)
{
    DDRD = 0b00000000;
    DDRB = 0b00100000;
    PORTB = 0b00000000;
}

void loop(void)
{
    // 入力
    output_A = (PIND & (1 << PIND2)) ? 1 : 0;
    output_B = (PIND & (1 << PIND3)) ? 1 : 0;
    if (output_A == 0 && output_B == 0) current_rotary_q1 = 0;
    if (output_A == 1 && output_B == 1) current_rotary_q1 = 1;
    if (output_A == 1 && output_B == 0) rotary_q2 = 1;
    if (output_A == 0 && output_B == 1) rotary_q2 = 0;

    switch (phase)
    {
    case right :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    right_count++;
                    if (right_count == 3)
                    {
                        right_count = 0;
                        phase = left;
                    }
                }
                // 左回転なら
```

```
                else {
                    start();
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;

    case left :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    start();
                }
                // 左回転なら
                else {
                    left_count++;
                    if (left_count == 2)
                    {
                        PORTB = 0b00100000;
                        _delay_ms(1000);
                        start();
                    }
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;
    }
}

void start(void)
{
    PORTB = 0b00000000;
    right_count = 0;
    left_count = 0;
    phase = right;
}
```

- 動作確認パターン
    - 右に3クリックし、続いて左に2クリック回したらLEDが1秒点灯するか。
    - 右に回転している最中に左に回転するとカウントがリセットされるか。
    - 左に回転している最中に右に回転するとカウントがリセットされるか。
- 結果
    - 1秒間LEDが点灯した。

    - リセットされた。
    - リセットされた。

実験7

- コード

```
#include <asf.h>
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);
void loop(void);
void start(void);

int output_A = 0;
int output_B = 0;
int current_rotary_q1 = 0;
int previous_rotary_q1 = 0;
int rotary_q2 = 0;
int right_count = 0;
int left_count = 0;
enum Phase { right_1 = 1, left_1 = 2, right_2 };
int phase = right_1;

int main(void)
{
    io_init();
    start();

    while (true)
    {
        loop();
    }
}

void io_init(void)
{
    DDRD = 0b00000000;
    DDRB = 0b00100000;
    PORTB = 0b00000000;
}

void loop(void)
{
    // 入力
    output_A = (PIND & (1 << PIND2)) ? 1 : 0;
    output_B = (PIND & (1 << PIND3)) ? 1 : 0;
    if (output_A == 0 && output_B == 0) current_rotary_q1 = 0;
    if (output_A == 1 && output_B == 1) current_rotary_q1 = 1;
    if (output_A == 1 && output_B == 0) rotary_q2 = 1;
    if (output_A == 0 && output_B == 1) rotary_q2 = 0;
```

```
    switch (phase)
    {
    case right_1 :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    right_count++;
                    if (right_count == 3)
                    {
                        right_count = 0;
                        phase = left_1;
                    }
                }
                // 左回転なら
                else {
                    start();
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;

    case left_1 :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    start();
                }
                // 左回転なら
                else {
                    left_count++;
                    if (left_count == 2)
                    {
                        left_count = 0;
                        phase = right_2;
                    }
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;

    case right_2 :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
```

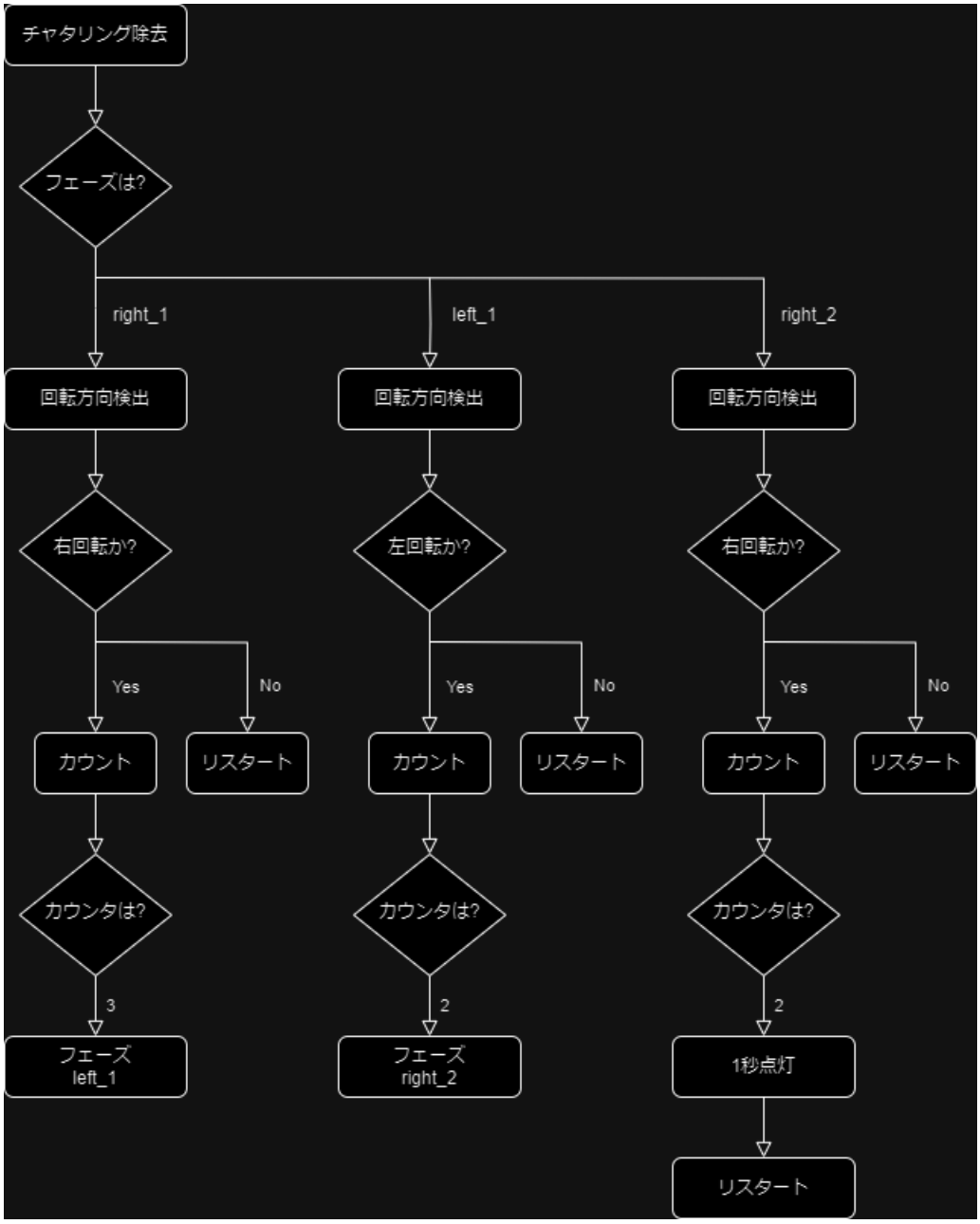
```
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    right_count++;
                    if (right_count == 2)
                    {
                        PORTB = 0b00100000;
                        _delay_ms(1000);
                        start();
                    }
                }
                // 左回転なら
                else {
                    start();
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;
    }
}

void start(void)
{
    PORTB = 0b00000000;
    right_count = 0;
    left_count = 0;
    phase = right_1;
}
```

- 動作確認パターン
    - 右に3クリックし、続いて左に2クリック回し、さらに右に2クリック回したらLEDが1秒点灯するか。
    - 右に回転している最中に左に回転するとカウントがリセットされるか。
    - 左に回転している最中に右に回転するとカウントがリセットされるか。
- 結果
    - 1秒間LEDが点灯した。
    - リセットされた。
    - リセットされた。

応用1

- コード
  実験7と同様

- 動作フロー



応用2

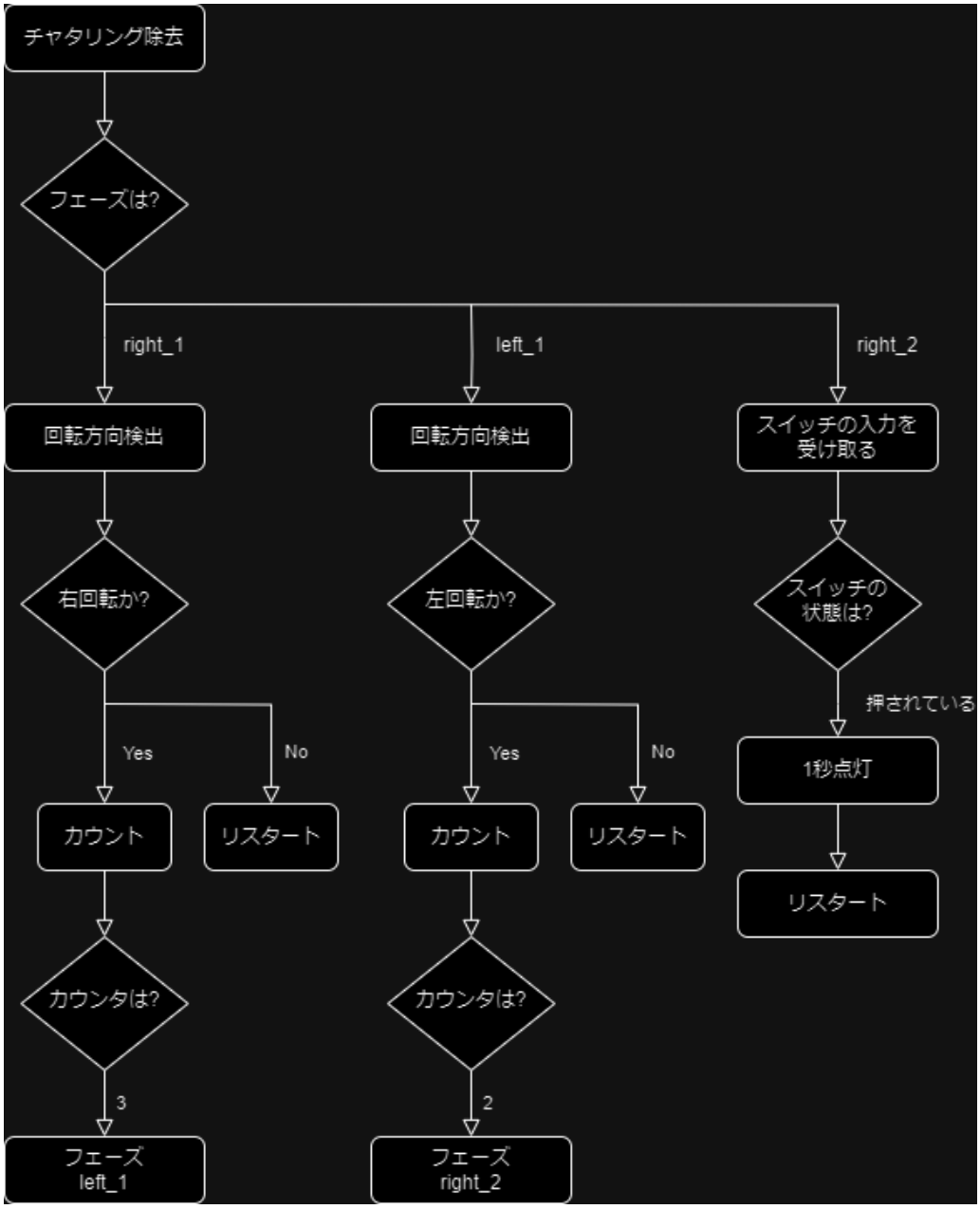
- コード

```
#include <asf.h>
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);
void loop(void);
void start(void);

int output_A = 0;
int output_B = 0;
int output_button = 0;
int current_rotary_q1 = 0;
```

```
int previous_rotary_q1 = 0;
int rotary_q2 = 0;
int right_count = 0;
int left_count = 0;
enum Phase { right = 1, left = 2, button = 3 };
int phase = right;

int main(void)
{
    io_init();
    start();

    while (true)
    {
        loop();
    }
}

void io_init(void)
{
    DDRD = 0b00000000;
    DDRB = 0b00100000;
    PORTB = 0b00000000;
    // PORTD4をプルアップに設定
    PORTD = 0b00010000;
}

void loop(void)
{
    // 入力
    output_A = (PIND & (1 << PIND2)) ? 1 : 0;
    output_B = (PIND & (1 << PIND3)) ? 1 : 0;
    if (output_A == 0 && output_B == 0) current_rotary_q1 = 0;
    if (output_A == 1 && output_B == 1) current_rotary_q1 = 1;
    if (output_A == 1 && output_B == 0) rotary_q2 = 1;
    if (output_A == 0 && output_B == 1) rotary_q2 = 0;

    switch (phase)
    {
    case right :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    right_count++;
                    if (right_count == 3)
                    {
                        right_count = 0;
                        phase = left;
                    }
                }
```

```
                // 左回転なら
                else {
                    start();
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;

    case left :
        // 状態が変化したら
        if (current_rotary_q1 != previous_rotary_q1) {
            // 立ち上がりを検出したら
            if (current_rotary_q1 == 1)
            {
                // 右回転なら
                if (rotary_q2 == 1) {
                    start();
                }
                // 左回転なら
                else {
                    left_count++;
                    if (left_count == 2)
                    {
                        left_count = 0;
                        phase = button;
                    }
                }
            }
            previous_rotary_q1 = current_rotary_q1;
        }
        break;

    case button :
        output_button = (PIND & (1 << PIND4)) ? 0 : 1;
        // スイッチが入力されたら
        if (output_button)
        {
            PORTB = 0b00100000;
            _delay_ms(1000);
            start();
        }
        break;
    }
}

void start(void)
{
    PORTB = 0b00000000;
    right_count = 0;
    left_count = 0;
    phase = right;
}
```

- 動作フロー



- 動作確認パターン
    - 右に3クリックし、続いて左に2クリック回し、さらにシャフトボタンを押したらLEDが1秒点灯するか。
    - 右に回転している最中に左に回転するとカウントがリセットされるか。
    - 左に回転している最中に右に回転するとカウントがリセットされるか。
- 結果
    - 1秒間LEDが点灯した。
    - リセットされた。
    - リセットされた。

課題

1. チャタリングとはなにか
   エンコーダーの出力信号に一時的な不安定な変化が出て、本来の値とは違う信号が出力されること。

2. チャタリングとコンタクトバウンスの違い
   チャタリングとは機械的な接点の問題に加えて、電気的ノイズや回路設計の問題により信号が一時的に不安定になる現象のことで、コンタクトバウンスは機械的なスイッチや接点が接触または分離する際に開閉が短期間に何度か繰り返される現象のこと。

3. チャタリングの除去法
   一般にソフトウェア的手法とハードウェア的手法がある。

   - ソフトウェア的手法
     1. デバウジングアルゴリズム
        - 短期間の信号の変動をフィルタリングする方法
        - 具体的には、タイマーを使用して信号の変化が終わってから処理を行う、または信号の変化を検出してから一定期間後に状態を確定させる方法がある。
     2. ヒステリシス
        - ヒステリシスを導入することで、信号のスムージングや不安定な変化の影響を抑えることができる。
        - 例えば、エンコーダーの信号が特定の閾値を超えてからのみ変化を受け入れるような閾値設定や、信号が一定期間安定してからのみ処理を行うなどが考えられる。
   - ハードウェア的手法
     1. キャパシタ
        - キャパシタをエンコーダーの出力に接続することで、一時的な急激な変化を抑制することができる。
        - キャパシタは高周波ノイズや瞬間的な変動を吸収する能力があり、これによりチャタリングの影響を軽減することができる。
     2. RCフィルタ
        - 抵抗器とキャパシタを組み合わせたRCフィルタを使用することで、信号の変化を滑らかにすることができる。
        - RCフィルタは特定の周波数成分をカットすることで、チャタリングやノイズの影響を減少させる役割を果たす。

感想

ロータリーエンコーダーを用いて回転信号を入力としてLEDを点灯させる処理ができた。チャタリングフィルタを `rotary_q1, rotary_q2` を用いて実装することができた。