1キーリモコン

概要

実家の古い台所についていた天吊り照明が古くなったので、新しいものに交換することになりました。ただし、元々のはひもで引っ張ってONOFF切り替える照明で、新しいのはリモコン式にしました。 うちの照明もリモコン式ですが、元々はリモコンなんてないものなので、取り付けたらつきっぱなしになってしまう照明を、 町の電器屋で購入した下記の装置を間に挟んでいます。製品型番はOCR-CRS01Wとなっています。 リモコンスイッチ 天井照明器具専用 [品番]04-9447｜株式会社オーム電機 リモコンは小さいカードサイズなのですが、わざわざホルダーからリモコンを取り出してON/OFFするのは、普段使いとしては大変です。 なので、部屋の入り口に専用のリモコンを作成することにしました。

設計仕様

小さなスイッチを高齢者が暗くなったら見分けられないので、ユーザビリティを考慮して大きめのボタン1つで操作するようにして ONの信号とOFFの信号を押下するたびに切り替えて送信するようにします。 また、どこに取り付けてもよいように、リモコンのLEDは最大3つを別角度て取り付けます。 木製の柱に付けるので、サイズとしては同様にカードリモコンサイズ、マイコン含めてコイン電池の3Vで動かします。 小さくする必要があるので、目標として秋月電子のFRISK基板を参考に、FRISKの容器にはいるようにします。

この設計で重要なのは、コイン電池で動かすの為、待機状態の消費電流をいかに小さくするかが重要になります。 昔の参考書だとOSCCONマイコンのクロックスピード）を未使用時にはLC(32.768kHz)にして、使用する時にHS(4～16MHz)に一時的に変更して処理するという方法です。 ただネットで探していると、アセンブラのsleepとnopを使ってOSCCONはそのままで、アセンブラのsleepとnopを入れて省電力化されている方法をとっておられたので、こちらを参考にさせていただきました。 PICを使った小型赤外線学習リモコン - Qiita

もう1つ重要なのは、いかにLEDの出力を上げられるかです。 というのも、コイン電池3Vで動作させているので、出力が大きいとLED点灯時に電圧が落ちてマイコンの動作電圧を下回ってBORになってしまいます。 とりあえず100uFのチップコンデンサをLED毎につける事にしましたが、こればかりは基板が届いてから追加検証が必要です。

回路図

（作成中）

ソースコード

開発環境 MPLAB X v5.50 XC8 V2.31 MPLAB Code Configurator 4

device_config.c

```
// CONFIG1
#pragma config FOSC = INTOSC    // Oscillator Selection->INTOSC
```

```
oscillator: I/O function on CLKIN pin
#pragma config WDTE = OFF    // Watchdog Timer Enable->WDT disabled
#pragma config PWRTE = ON    // Power-up Timer Enable->PWRT enabled
#pragma config MCLRE = OFF    // MCLR Pin Function Select->MCLR/VPP pin
function is digital input
#pragma config CP = OFF    // Flash Program Memory Code
Protection->Program memory code protection is disabled
#pragma config CPD = OFF    // Data Memory Code Protection->Data memory
code protection is disabled
#pragma config BOREN = ON    // Brown-out Reset Enable->Brown-out Reset
enabled
#pragma config CLKOUTEN = OFF    // Clock Out Enable->CLKOUT function
is disabled. I/O or oscillator function on the CLKOUT pin
#pragma config IESO = OFF    // Internal/External
Switchover->Internal/External Switchover mode is disabled
#pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable->Fail-
Safe Clock Monitor is enabled

// CONFIG2
#pragma config WRT = OFF    // Flash Memory Self-Write
Protection->Write protection off
#pragma config PLLEN = OFF    // PLL Enable->4x PLL disabled
#pragma config STVREN = ON    // Stack Overflow/Underflow Reset
Enable->Stack Overflow or Underflow will cause a Reset
#pragma config BORV = LO    // Brown-out Reset Voltage
Selection->Brown-out Reset Voltage (Vbor), low trip point selected.
#pragma config LVP = OFF
```

pin_manager.c

```
LATA = 0x00;
TRISA = 0x3F;
ANSELA = 0x17;
WPUA = 0x00;
OPTION_REGbits.nWPUEN = 1;
APFCON = 0x01;
```

mcc.c

```
WDTCON = 0x16;
```

main.c

```
#include "mcc_generated_files/mcc.h"

char IR_ON[1] = {0x87};
char IR_OFF[1] = {0x4B};
unsigned int arnum;
```

```
//IR信号送信処理
void SendIR (char *s, int snum) {

    //一時処理のデータ箱
    int i,j;
    char w;

    //動作確認用LED
    RA2 = 1;
    TMR2 = 0;

    //リーダーコードを送信する
    TMR2ON = 1;
    __delay_us(2560);
    TMR2ON = 0;
    __delay_us(2640);

    //カスタマーコードとデータ送信
    for (i=0;i<1;i++) {

        w = s[i]; //1バイトずつ取り出す

        //取り出したデータを1ビットずつ処理する
        for (j=0;j<8;j++) {

            TMR2ON = 1;
            __delay_us(840);
            TMR2ON = 0;

            //データ0と1とはOFF時間の長さが異なるので分ける
            if (w & (0b00000001 << j)) {

                __delay_us(840); //ビット1のとき

            } else {

                __delay_us(1840); //ビット0のとき
            }
        }

    }

    //ストップビットの送信
    TMR2ON = 1;
    __delay_us(840);
    TMR2ON = 0;
    __delay_ms(36); //タイムアウト

    //動作確認用LED
    RA2 = 0;
```

```
}

void main(void)
{
    unsigned short mode = 1; //スイッチが押されたら送信するコードを選択

    // initialize the device
    SYSTEM_Initialize();

    OSCCON = 0b01101010; //内部オシレータ 4MHz
    APFCON = 0b00000001; //CCP1=RA5
    TRISA = 0b00001000; //RA3のみ入力
    ANSELA = 0; //アナログを使用しない
    WPUA = 0b00001000; //RA3のみプルアップ

    //割り込み設定
    INTCON = 0; //割り込み無効
    IOCAN = 0b00001000; //RA3の立下り検出

    //PWMキャリアの設定
    PR2 = 0b00011001; //38khz
    T2CON = 0b00001000; //ポストスケーラ1:2 Timer2 = OFF プリスケーラー1
    CCP1CON = 0b00111100; //デューティサイクル0b11PWM
    CCPR1L = 0x08; //デューティーサイクル35%

    while (1) {

        //ボタンが押下されたら実行
        if (RA3 == 0) {

            //配列をポインタに置き換えると、正しい要素数が計算できなくなる。
            //https://www.sejuku.net/blog/24793
            //「関数に配列を渡すときの注意点」を参照。
            //対策としてarnumで計算した値を一緒に渡す。

            if (mode == 0) {

                arnum = sizeof(IR_OFF);
                SendIR(&IR_OFF,arnum);
                __delay_ms(60);
                SendIR(&IR_OFF,arnum);
                mode = 1;

            } else {

                arnum = sizeof(IR_ON);
                SendIR(&IR_ON,arnum);
                __delay_ms(60);
                SendIR(&IR_ON,arnum);
```

```
                mode = 0;

            }

            //チャタリング対策
            __delay_ms(100); //ちょっと待つ
            while (RA3 == 0) {

                __delay_ms(100); //押下されなくなるまで待つ

            }
        }

        //ボタン押下されるまで停止

        IOCAF = 0; //各ピンの状態変化割り込みクリア
        IOCIE = 1; //状態変化割り込みを有効
        SLEEP();
        NOP();
        IOCIE = 0;
        IOCAF = 0;
    }
}
```

信号解析とデバックについて

リモコンといえばNEC方式とか家電協方式とかあるのですが、最近の家電はそんな常識にとらわれない信号を出力するものがあると聞きます。 知識では知っていたものの、まさかこのリモコンもそれに該当するものとはおもってもみませんでした。 昔から愛用しているこの解析装置の値で設計すると、まったく動作しません。 マルチメーカー対応赤外線リモコンコード解析器 PIC16F648A版

なので最近はONとOFF時間をusで計測して、そのまま信号を送ったりするのがトレンドのようです。
数ある方法のうちM5Atomと赤外線ユニットを使った方法で解析とデバックを行いました。

（作成中）

動作確認

（作成中）