AC電力制御

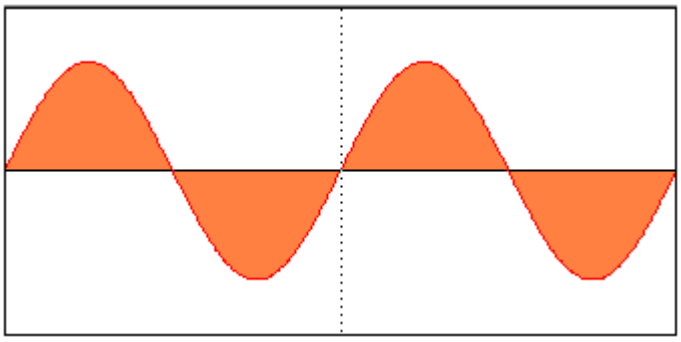
概要

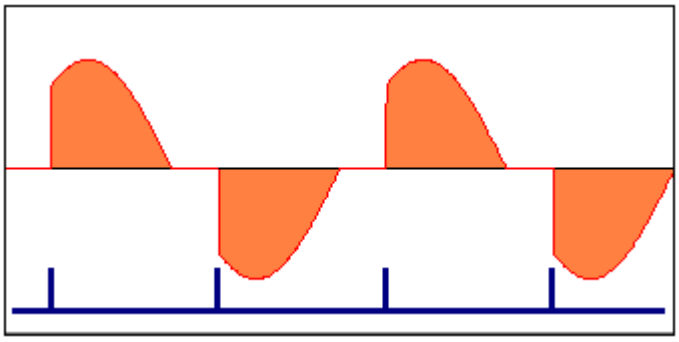
単純なAC電力の制御は、トライアックを使って簡単に実現できます。 今回は、PIC16F88を使用して、制御を行います。そして電力量をバーグラフおよび%表示でLCDに表示させる ようにしました。 こうすることにより、今まではボリュームの回転角度または感のみで制御していたものがデジタル的に制御でき るようになりました。

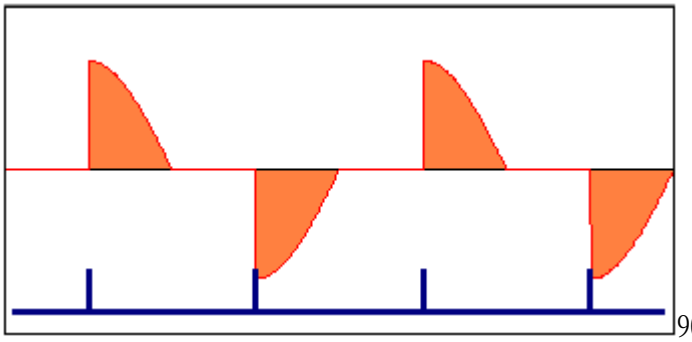
制御方法

トライアックはゲートを、ONにすると後はゲートをOFFにしても電流が流れ続けます。 そして交流のサイクルで0Vになったところで電流がOFFになります。

非ゼロクロス型では、電源の位相にあわせて短いパルスでONにします。 すると次にくる0Vの時間まで電流が流れます。 この短いパルス(トリガ)を半サイクルの立ち上がり直後にすると100%近い時間電流が流れます。 逆に半サイクルの終わり近くでトリガすると0%に近い時間だけ電流が流れます。 つまり半サイクルごとに通電時間を決めることにより電力を制御できることになります。

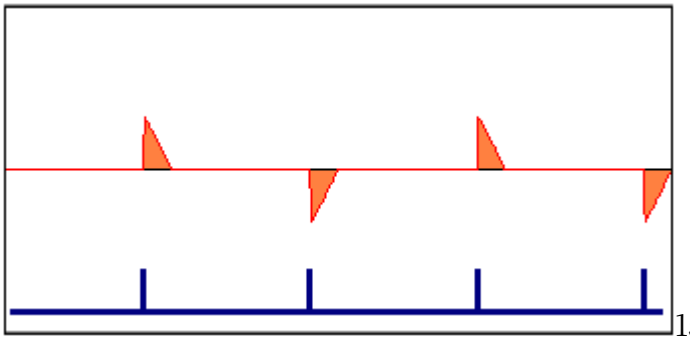






30°分カット

90°分カット

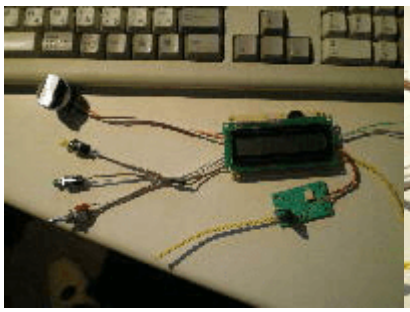
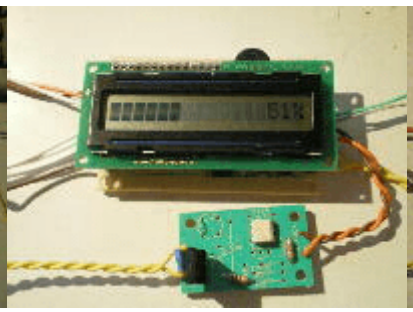
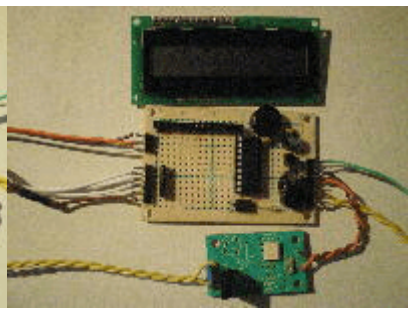


150°分カット

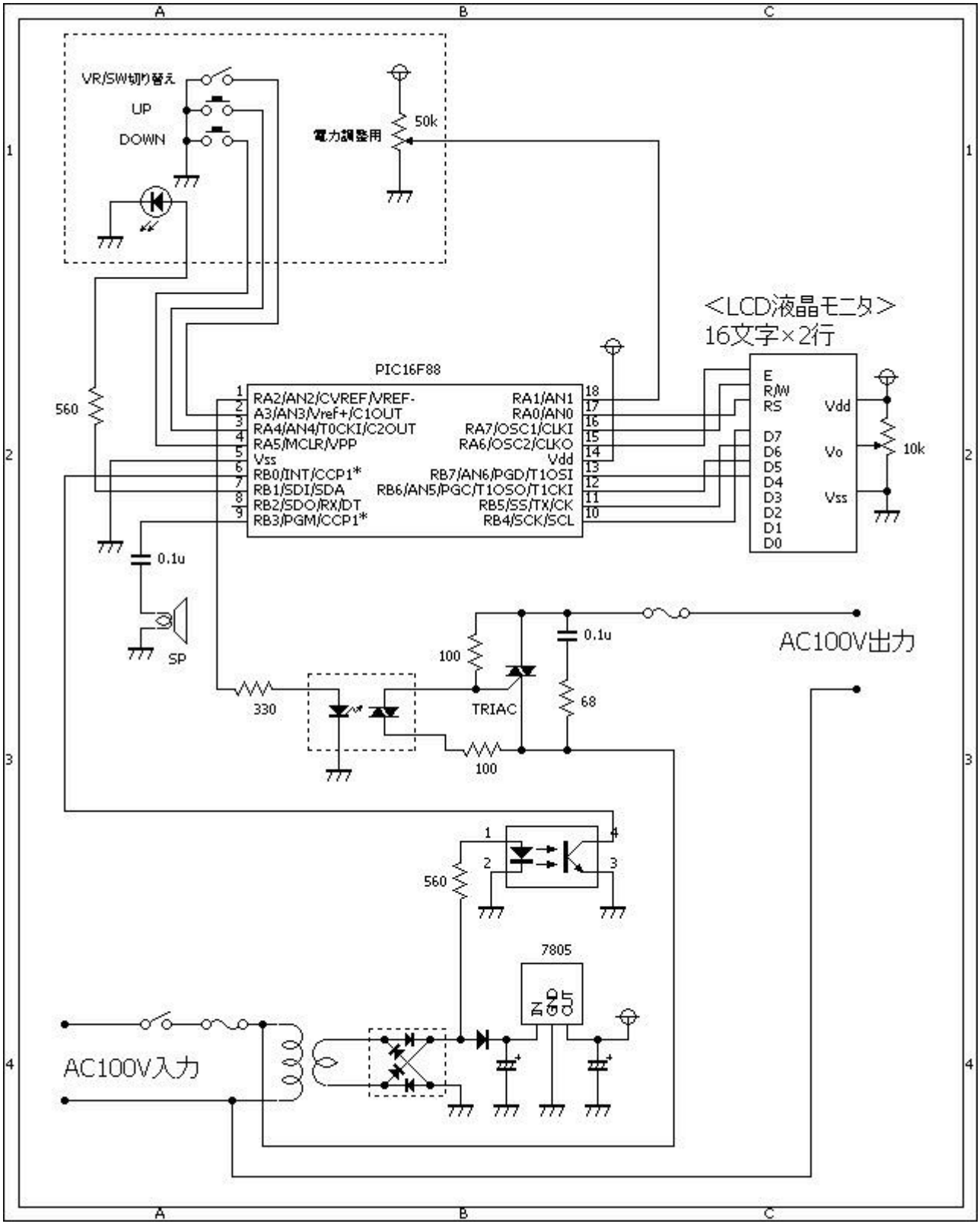
ハードウェア構成

主要な機能は

- 正弦波の正(+)の開始および負(-)の開始を検出するために全波整流した直後に電圧でフォトカプラを駆動する部分。これにより60Hzの2倍での120のパルスを得ることが出来る。
- 東芝のTLP560Gというフォトトライアック(非ゼロクロス)とトライアックにより電力を最終的に制御する。
  フォトトライアックへのトリガ信号は、PICよる出力される。
- 設定する制御値は、VRおよび二つのプッシュSWによりUP/DOWNされる。
- VR設定するかSWで設定するかの切り替えはトグルSWにより行う。
- PIC16F88では、全体制御(フォトトライアックの制御やLCDへの表示等)を行う。

回路図



処理説明

処理は以下のブロックより構成されます。

- ポート、クロック、タイマ□□PWM□LCD等を初期化する。
- LCDへタイトルを表示する。
- 制御をVRで設定するかSWで設定するかをチェックする。

- VRであれば□AD変換した値を制御値(sleepTime)とする。
- SWであれば、更にUP/DOWNをチェックし、制御値をUP/DWONする。
- 電力をバー(12個)で表示する。
- 電力を%で表示する。
- 割り込み処理では、以下の3つの処理をする。
  - TIMER1割り込み(約200msec周期)では□LEDをON/OFFさせる。(パイロットランプ)
  - 60Hzのトリガ信号割り込み(全波整流なので秒あたり120回)では、制御値を一時変数にセットする。
  - TIMER0割り込み(約60usec周期)では、制御値経過後にトライアックをONする。

ソースコード

acControl2.c

```
//*****************************************************************************
*

/*
    □AC電力制御＞
*/

//*****************************************************************************
*

static unsigned    short    controlFlag;
static unsigned    short    sleepTimeTemp;
static unsigned    short    sleepTime;

void   interrupt()
{
    if (INTCON.T0IF == 1) {         //約60usec周期
        INTCON.T0IF = 0;
        TMR0 = 170;
        //
        if (controlFlag == 1) {
            if (sleepTimeTemp > 0) {
                if (sleepTimeTemp == 1) {
                    PORTA.F2 = 1;
                }
                sleepTimeTemp--;
            } else {
                PORTA.F2 = 0;
            }
        }
    }
    if (PIR1.TMR1IF == 1) {         //約200msec周期
        PIR1.TMR1IF = 0;
        PORTB.F1 = ~PORTB.F1;
    }
```

```
    if (INTCON.INTF == 1) {         //約8msec周期(60Hz)
        INTCON.INTF = 0;
        sleepTimeTemp = sleepTime;
    }
}

//*********************************************************************
*

void   Pwm_Change_DutyEx(unsigned int duty_ratio)
{
    CCPR1L = duty_ratio >> 2;
    CCP1CON.F6 = duty_ratio & 0b00000001;
    CCP1CON.F7 = (duty_ratio & 0b00000010) >> 1;
}

//*********************************************************************
*

void   main()
{
    static    unsigned    int        ad1;
    static    unsigned    short    cnt, i;
    static    unsigned    long    l;
    static    unsigned    char    buf[10];
    //
    CMCON = 0b00000111;             //コンパレータは使用しない。
    ANSEL = 0b00000010;             // AN1を使用する。
    TRISA = 0b00111010;             //ポートAを設定する。
    TRISB = 0b00000101;             //ポートBを設定する。
    OSCCON = 0b01110000;          //クッロクを8Mhzに設定する。
    OPTION_REG = 0b00001000;     //ポートBをプルアップするTimer0のプリスケーラは1/1に設定する。
    T1CON = 0b00110001;             // Timer1のプリスケーラは1/8に設定する。
    OPTION_REG.INTEDG = 0;         // INTピンPORTB.F0の立ち下がりエッジで割り込みをかける。
    INTCON.GIE = 0;                 //割り込みを無効にする。
    INTCON.PEIE = 1;              // Peripheral Interrupt Enable
    INTCON.TMR0IE = 1;              // TMR0 Overflow Interrupt Enable
    INTCON.TMR0IF = 0;              // TMR0 Overflow Interrupt Flag
    INTCON.INTE = 1;              // RB0/INT External Interrupt Enable
    PIE1.TMR1IE = 1;              // TMR1 Overflow Interrupt Enable
    PIR1.TMR1IF = 0;              // TMR1 Overflow Interrupt Flag
    T2CON.T2CKPS0 = 0;              // TMR2のプリスケーラを0にする。
    T2CON.T2CKPS1 = 0;              //
    //
    Pwm_Init(2000);    // 2Khz
    Pwm_Change_DutyEx(1024 / 2);
    //
    Lcd_Custom_Config(&PORTB,4,5,6,7,&PORTA,0,7,6);
    TRISB = 0b00000101;
```

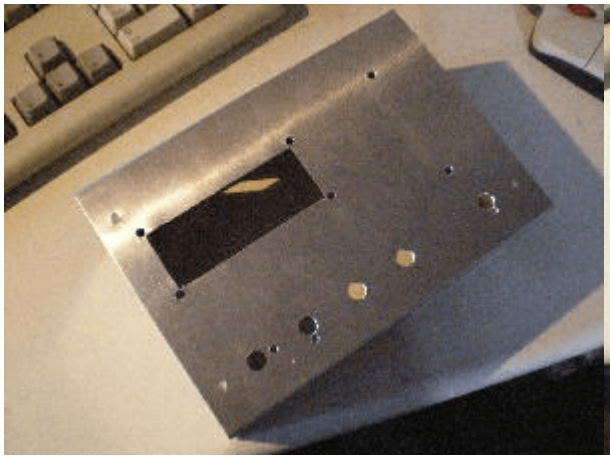
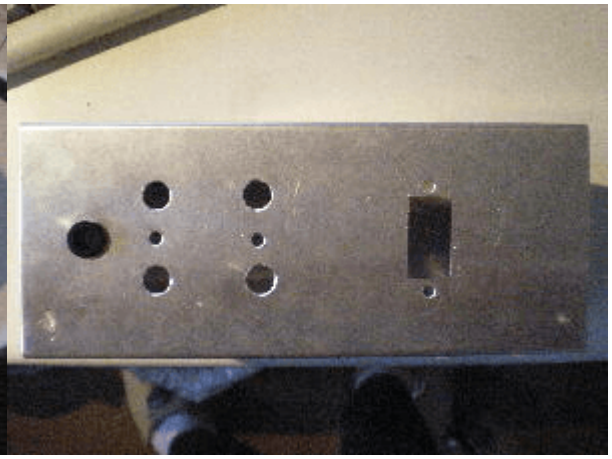
```
    TRISA = 0b00111010;
    //
    controlFlag = 0;
    sleepTime = 0;
    sleepTimeTemp = 0;
    //
    Lcd_Custom_Cmd(LCD_CURSOR_OFF);
    for (cnt = 0; cnt < 5; cnt++) {
        Lcd_Custom_Out(1, 1, "AC contr");
        Lcd_Custom_Out(2, 1, "ol  R1.1");
        Pwm_Start();
        Delay_ms(200);
        Pwm_Stop();
        Lcd_Custom_Cmd(LCD_CLEAR);
        Delay_ms(200);
    }
    //
    INTCON.GIE = 1;     // これ以降の処理で割り込みを許可する。
    //
    while (1) {
        // VRで設定するかSWで設定するかをチェックする。
        if (PORTA.F3 == 1) {
            ad1 = Adc_Read(1);                  //設定値を取り込む。
            sleepTime = 127 - (ad1 / 8);     //127段階で調整できる（0…127）
        } else {
            if (PORTA.F4 == 0) {             //暗くなる！
                if (sleepTime < 127)
                    sleepTime++;
                //ブザーを鳴らす。
                Pwm_Start();
                Delay_ms(50);
                Pwm_Stop();
            }
            if (PORTA.F5 == 0) {             //明るくなる！
                if (sleepTime > 0)
                    sleepTime--;
                //ブザーを鳴らす。
                Pwm_Start();
                Delay_ms(50);
                Pwm_Stop();
            }
        }
        //
        switch (sleepTime) {
        case 0:         //電力が100%の時は、強制ONとする。
            controlFlag = 0;
            PORTA.F2 = 1;
            break;
        case 127:     //電力が0%の時は、強制OFFとする。
            controlFlag = 0;
```

```
            PORTA.F2 = 0;
            break;
        default:
            controlFlag = 1;
        }
        //電力をバー(12個)で表示する。
        cnt = (127 - sleepTime) / 10;         // 0...12
        for (i = 1; i < 9; i++) {
            if (cnt > 0) {
                cnt--;
                Lcd_Custom_Chr(1, i, 0xFF);
            } else {
                Lcd_Custom_Chr(1, i, ' ');
            }
        }
        for (i = 1; i < 5; i++) {
            if (cnt > 0) {
                cnt--;
                Lcd_Custom_Chr(2, i, 0xFF);
            } else {
                Lcd_Custom_Chr(2, i, ' ');
            }
        }
        //電力を%で表示する。
        l = (long)sleepTime;
        l = ((127 - sleepTime) * 100) / 127;
        WordToStr(l, buf);
        buf[5] = '%';
        buf[6] = 0x00;
        Lcd_Custom_Out(2, 5, &buf[2]);
        //
        Delay_ms(1);
    }
}

//*****************************************************************************
*
```

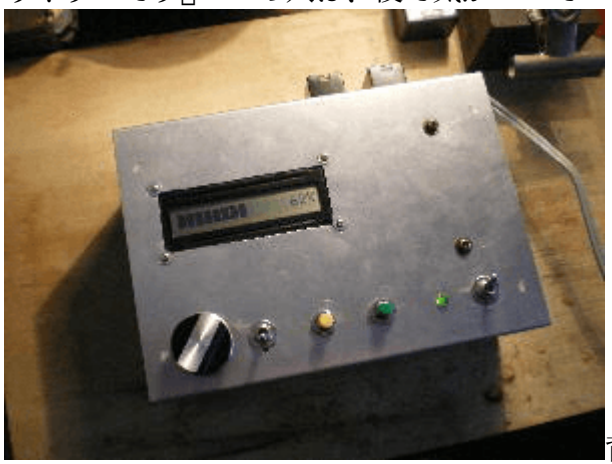
ケース加工

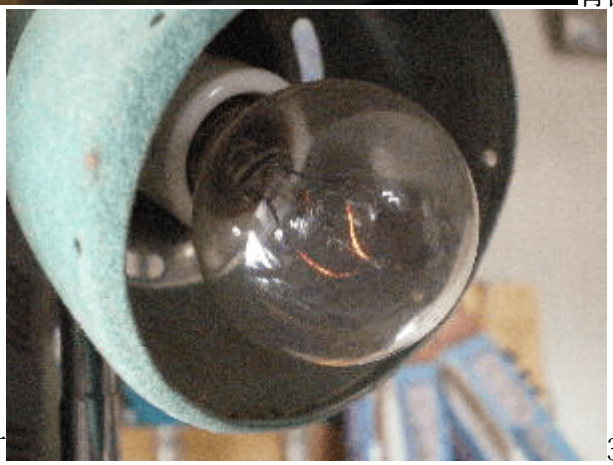
ドリル、リーマ、ニブリングツール等を利用してケース加工をしました。一番苦手な作業です。

動作確認

本装置を使用して、豆電灯を制御してみました。 バーグラフと%表示が出来るので直感的でとてもわかりやすいです LEDの穴は、後で気がついてハンドドリルで急遽穴あけしました。 上面


背面

10%点灯

30%点灯