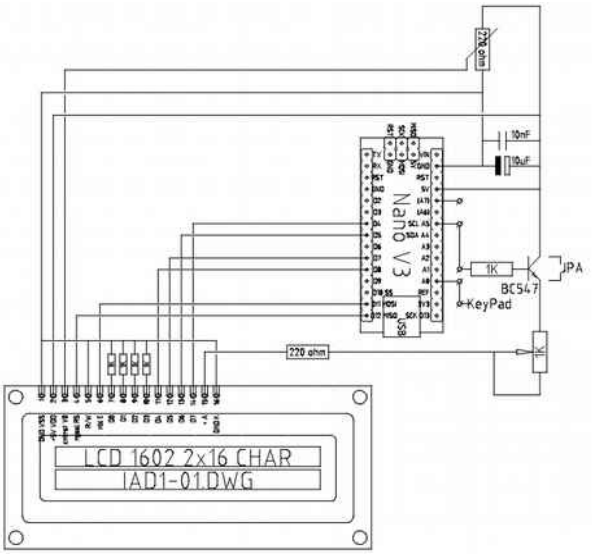
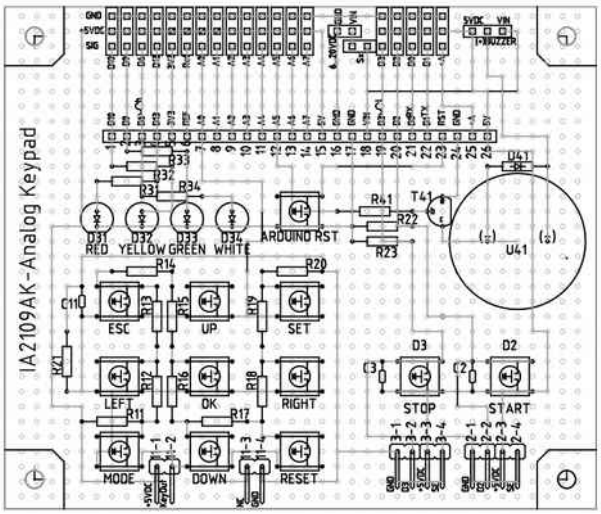
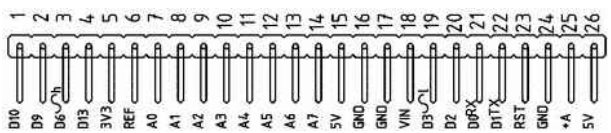
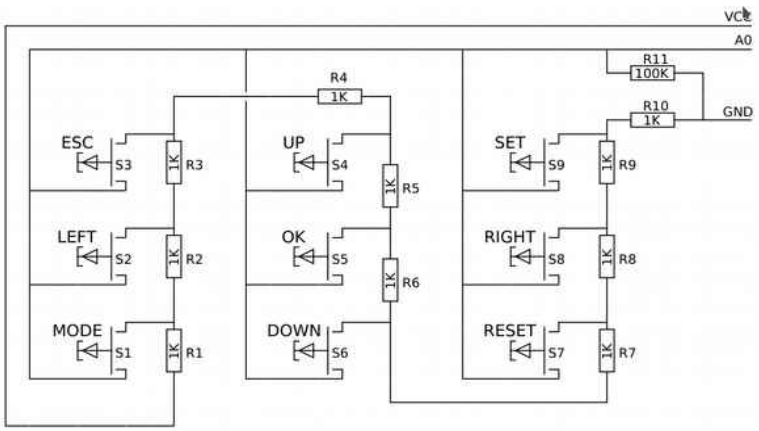
IA2109LCD-module In dit project wordt gebruik gemaakt van de IA2109 LCD-module. De opbouw en werking is in deel 1 behandeld.



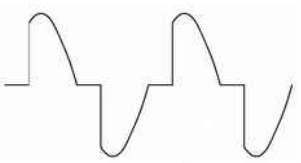
Er wordt gebruik gemaakt van de universele test- en experimenteerprint IA2109AK. De sketches geven voldoende informatie voor het direct correct aansluiten op een Nano of Uno of andere controller. Dus ook wanneer deze experimenteer-print niet gebruikt wordt.



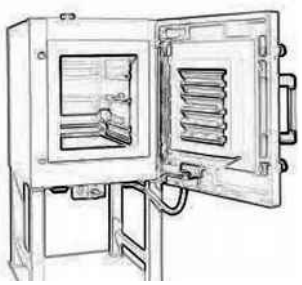




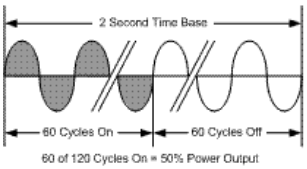
Dynamische Perioden-regeling (DPC)



Inleiding. Een groot nadeel van fase-aansnijding is dat deze techniek door de nogal hoge en zeer steile inschakelstromen extra warmte-ontwikkkeling en radio-stoor-signalen (Radio Frequency Interference; RFI) opwekt. Dit maakt fase-aansnijding eigenlijk alleen geschikt is voor weerstandslasten zoals gloeilampen, of voor lichte transformator gekoppelde lasten. Voor capacitieve lasten is fase-aansnijding ongeschikt, en bij inductieve lasten zullen de nodige ontstoor/correctie-voorzieningen getroffen moeten worden. Zodra er een **zware last** geregeld moet worden, zal al snel een andere (industriële!) techniek moeten worden gebruikt.



Perioden-regeling. Bij zwaar elektrische belastingen zoals verwarmings-elementen van boilers en keramiek-ovens in de industrie, vraagt de hoge inschakelstroom bij fase-aansnijding al snel om dure speciaal-oplossingen. De eenvoudigste manier van regelen, is dan Aan/Uit-regelen. Bij 10 seconden Aan gevolgd door 10 seconden Uit is dit een aansturing van 50%. Op deze manier kan de stroom rustig met de spanning meebewegen, en zal de inschakelstroom getemperd en RFI minimaal zijn. Een Aan/Uit-regeling kan ook gezien worden als een in lengte regelbare **perioden-trein** *(period/cycle sequence; train)*, zodat deze stuurregeltechniek **perioden-regeling** genoemd wordt.



Overigens, een CV-monteur zal een perioden-regeling al snel kwalificeren als 'pendelen', wat voor een CV-ketel zeer ongewenst is. Bij een automotor is de periodenregeling dan vergelijkbaar met zoveel seconde plankgas en vervolgens zoveel seconden gas los, zodat er een 'gemiddelde rij-snelheid' gestuurd kan worden.

Wordt een triac aangestuurd met een 50% PWM-signaal van ca. 1000 Hz, is dat al voldoende om zo meerdere perioden van een 50 Hz netspanning achter elkaar op tijd te kunnen schakelen. Bij 0% PWM wordt de triac niet ontstoken, en zo kan een periodentrein-wachtcyclus worden gemaakt. De verhouding tussen Aan- en Uit-geschakelde periodentreinen is dan de vermogensregeling. Kortom, er wordt een ultra lage PWM gebruikt, om een hogere PWM te kunnen schakelen. Op het internet wordt dit regelmatig 'Burst Mode' genoemd, terwijl er eigenlijk sprake is van (misbruik van) **geschakelde PWM *(keyed PWM)***.

Nul-op-de-meter. Gebruik de term 'nul-op-de-meter', en er ontstaan misverstanden over wat dat zou zijn en hoe dat werkt. Wat een **Watt** is, is vastgelegd in de **IJkwet**, die op zijn beurt verwijst naar het **SI-stelsel** en diens natuur-wetenschappelijke definities, èn zoals de Watt is vastgelegd in de internationale **ISO, IEC, DIN en NEN-normen**. Punt.

De kWh-meter heeft als **meetsensor** een **Watt-meter**, waarbij he netto aantal getelde **energiepakketjes (joules) per seconde** wordt gemeten in Watt.

De Watt-meter als meet-sensor moet dus de **per seconde** totaal ingaande energiestroom (import in Joules) en totaal uitgaande energiestroom (export in Joules) bij elkaar optellen (ofwel met elkaar verrekenen), en het resultaat dan toevoegen aan het import- of export-telwerk. Bedenk wel, de meetmethode van energie-meting **per seconde** ligt vast in de wet; daar kan geen meter-fabrikant, netbeheerder of energieleverancier ook maar iets aan veranderen. Op welke manier (chemisch, mechanisch, magnetisch of digitale processor) de kWh-meter dat doet, doet daarbij totaal niet ter zake.

Kortom, een kWh-meter **sommeert** en **saldeert** het energieverbruik **per seconde**; het maakt immers gebruik van een **Watt**-meter. Wanneer binnen één seconde er net zoveel energie geïmporteerd als geëxporteerd wordt (maakt niet uit hoeveel energie) dan is de effectieve stroom nul, en zullen de telwerken stilstaan; dit wordt **nul-op-de-meter** genoemd.

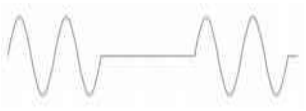
Alle woningen moeten van het gas. Dus zonnepanelen op het dak, elektrische kookplaat, boiler en warmtepomp er in, en de particulier ziet zijn stroomverbruik omhoog vliegen. Zolang de kWh's gesaldeerd kunnen worden, is de financiële pijn nog beperkt.

Wanneer de zonne-energie-kWh's niet gesaldeerd kan worden, is de traditionele periodenregeling een hele dure grappenmaker. Stel, neem een elektrische boiler van 4 kW, en de PV-installatie levert 2 kW aan het net. Wanneer de boiler Uit is, wordt de eigen duur opgewekte energie gratis aan de energieleverancier weggeven. Even later gaat de boiler Aan, wordt er 4 kW verbruikt, waarvan 2 kW weer ingekocht moet worden, inclusief winstopslag, millieubelasting en btw. De kWh-import-en-export-telwerken brengen dit alles tegen de hoofdprijs keurig in rekening.

Het energieverbruik van de boiler kan met triac-fase-aansnijding zodanig teruggeregeld worden, zodat alleen de eigen opgewekte zon-energie voor de boiler gebruikt wordt. Dit wordt dan **regelen** naar **nul-op-de-meter** genoemd. De import- en export-telwerken staan dan stil. Dat niets geteld wordt is correct, immers de eigen opgewekte energie wordt voor de boiler gebruikt. Kortom; bij nul-op-de-meter hoeft de energie-leverancier hoeft niet meer betaald te worden voor het gebruik van de eigen opgewekte energie. Een nadeel is wel dat bij nul-op-de-meter het opwarmen van de boiler langzamer gaat.

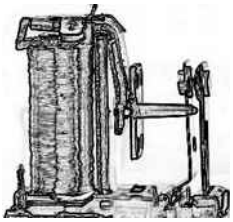
Helaas is triac-fase-aansnijding -vooral vanwege de piek-inschakelstromen- niet zo geschikt voor het aansturen van zware lasten, zoals een boiler, olieradiatorkachel, infra-rood paneel, of …

(A)synchrone periodentrein-regeling. Wanneer met een PWM-stuursignaal op een willekeurig moment een triac wordt ingeschakeld, zal bij 50 Hz VAC periodenregeling de eerste periodehelft vrijwel zeker mèt fase-aansnijding starten, en zullen alle overige periodehelften wel voldoende op tijd gestart worden. Dit is een **a-synchrone periodentrein-regeling**. Deze methode zien we bij nogal wat arduino-voorbeelden op het internet; de triac wordt bediend door een grof bombardement aan ontsteekpulsen. Geheel RFI storingsvrij zal dit niet zijn, en ook de inschakelpiekstroom kan nog steeds fors zijn.



Een betere variant is de **synchrone periodentrein-regeling**, waarbij een nuldoorgangsdetector *(zero cross)* wordt gebruikt om een periodentrein op de nuldoorgang te laten beginnen. Op het internet wordt deze methode aangeduid als 'On/Off-control', 'Proportional On/Off Control', 'zero-crossing control mode', ' zero cross voltage switching', 'fast cycling', 'integral cycle', 'burst firing', 'zero-cross time proportion control', 'skip cycle', of ..., en bedenk het maar.

Voor grote belastingen zoals een 4 kW boiler lijkt de DC-PWM-periodenregeling een ideale oplossing, maar dit is wel sterk afhankelijk van een aantal factoren, zoals kan de belasting een dergelijk aansturing verdragen? Verkort deze methode de levensduur van schakeling en/of belasting? DC-spanningen in een vochtige omgeving (boiler) veroorzaakt zeer sterke metaalcorrrosie (galvano-effect). Wat is de invloed en zijn de effecten op het voedingsnet? Veroorzaakt de belasting een netspannings-asymmetrie?



Solid State Relay, SSR. In de elektrotechniek is een relais een elektro-mechanische schakelaar. De spoel veroorzaakt Electro Magnetische Interferentie (EMI), en de schakel-contacten veroorzaken als vonkenzender RFI.

De tot dusver gebruikte triac-module is in feite de basis-uitvoering van een zogeheten **Solid State Relay (SSR)**. Een SSR is een 'elektronisch relais' inclusief de stuurelektronica, opgebouwd met normale componenten, zoals weerstanden, leds, opto-coupler, diac, triac/thyristor, enz. Deze zijn als complete module verkrijgbaar.

LET OP; controleer wel of de SSR direct schakelt, of bij een nuldoorgang, of bij een continu ingangssignaal, of bij een PWM, of bij een AC-ingangssignaal, of bij…

Ook belangrijk om te weten is hoe de SSR schakelt; als triac met fase-aansnijding? Als perioden-trein-schakelaar? Wel of niet inschakelen op de nuldoorgang?

Model: SSR-DD41F	**Load Current:** 6A
Control Method: DC controlled DC	**Off-state Time:** ⩽10mS
Control Voltage: 3-32VDC	**Installation:** Rail mounting
Load Voltage: 5-60VAC	

Model: SSR-DA41F	**Load Current:** 6A
Control Method: DC controlled AC	**Off-state Time:** ⩽10mS
Control Voltage: 3-32VDC	**Installation:** Rail mounting
Load Voltage: 24-250VAC	

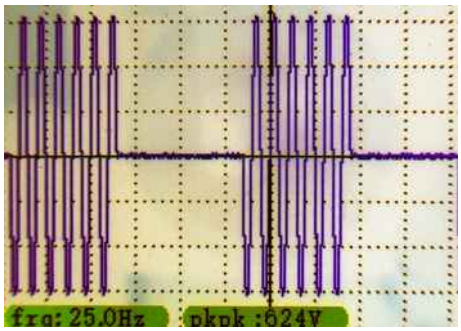
Hoewel SSR's op de nuldoorgang kunnen schakelen, hebben de meeste geen nuldoorgang-uit-gangs-signaal.

Amplitude Modulatie. Een totaal andere benadering van vermogensregeling is te stellen dat bij een zuivere spannings-amplitude-regeling het opgenomen vermogen ook geregeld kan worden. Naast de geëigende variac, is het in principe mogelijk een dergelijke schakeling te bedenken die dat kan; AM-zenders, sinusgeneratoren, en schakelende voedingen kunnen dat immers al. Vraag is wel of dat niet de meest moeilijke en gecompliceerde weg van vermogensregeling is wanneer in principe een triac het al aan kan.

Dynamische Perioden-regeling. Triac-fase-aansnijding is niet optimaal voor een elektrische boiler, en de industriële perioden-regeling is niet snel genoeg om nul-op-de-meter te kunnen krijgen. In een totaal andere **dynamische(!)** manier van aansturen van de triac, kan het schakelprincipe van de periodenregeling gebruikt worden, om toch nul-op-de-meter te kunnen realiseren. Voor alle duidelijkheid, de dynamische periodenregeling is géén op zichzelf staande regeling of sensor-meet-techniek, maar is slechts een 'energie-aanstuur-techniek' (net zoals Aan/Uit-sturing, Fase-aansnijding, of PWM dat ook zijn). Het leuke daarbij is dat de traditionele fase-aansnij-circuits daarvoor gebruikt kunnen worden.

Voor nul-op-de-meter regelen moet de **energie-sturing effectief dus korter zijn dan één seconde**. Bij de triac-fase-aansnijding is elke halve periode gelijk aan de volgende, zodat deze vorm van vermogensregeling eenvoudig te doorgronden is.

Een digitale signaalgenerator (DAC) gebruikt meerdere opeenvolgende digitale blokpulsen om een bepaald signaal te construeren. De dynamische periodensturing halve perioden om een **energie-periode** te construeren.



Hiernaast als voorbeeld de weergave van de 48% energie-periode. Deze bestaat uit twee **periodetreinen** van geschakelde halve perioden, een **wachttijd** na de eerste periodetrein, èn een **wachttijd** na de tweede periodetrein. De wachttijden hoeven niet gelijk aan elkaar te zijn. De totale energie-periode bestaat uit Periodetrein plus Wachttijd1 plus Periodetrein plus Wachttijd2 (dus het gehele plaatje, waarvan wachttijd2 iets korter is dan wachttijd1).

Tijdens een periodetrein zal de energiestroom naar de last maximaal zijn. Bij een PV-installatie zal bij onvoldoende zon-energie dit aangevuld worden door energie vanuit het net. Tijdens de wachttijden is de last afgeschakeld en zal de zon-energie aan het net teruggeleverd worden. Voor de k-Watt-h-meter geldt dat wanneer deze energie-import-en-export binnen één seconde in balans is, er nul-op-de-meter staat. Kortom, met de dynamische periodenregeling kunnen zware lasten als nul-op-de-meter gestuurd worden.

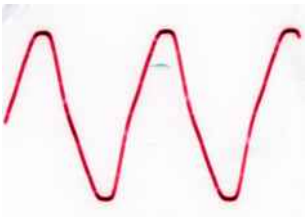
In de regeltechiek heeft een van nul tot honderd procent regelwaarde -vanwege het intuïtieve karakter- de voorkeur. Het is niet eenvoudig om een procentuele perioden-regeling te realiseren, wanneer slechts 50 wisselstroom-perioden per seconde beschikbaar zijn. De **dynamische perioden-regeling** ***(Dynamic Period/Power Control; DPC)*** is een verbeterde variant van de industriële seconden-schakelende periodentrein-regeling. De basis van de DPC-regeling zijn honderd halve perioden, ofwel honderd inschakelmomenten.

Wanneer honderd halve perioden lang niets ingeschakeld wordt, is de regelwaarde 0%, ofwel alles is Uit. Wordt eerst 1 halve periode doorgelaten, en dan 99 halve perioden tegengehouden, is de regelwaarde 1%. Bij 2 halve perioden doorlaten en dan 98 halve perioden tegenhouden, is de regelwaarde 2%. 3 halve perioden doorlaten is 3%, enz. Bij 50 Hz netspanning geeft dit een nul tot honderd procent regelbereik binnen een Watt-meetcyclus van 1 seconde. Theoretisch! Nu de praktijk nog.

Zeer belangrijk bij zware netbelastingen -zoals ovens en boilers- is om het net -en de last zelf- zo gelijkmatig mogelijk te belasten. Ofwel een **positieve** halve periode **moet** gevolgd worden door een **negatieve** halve periode. Wanneer men de twee halve perioden zo egaal binnen de energie-periode verspreid schakelt, zal de belasting beter verdeeld worden en de piekinschakelstromen gehalveerd. Om RFI en EMI te voorkomen mag alleen op de nuldoorgang worden ingeschakeld.

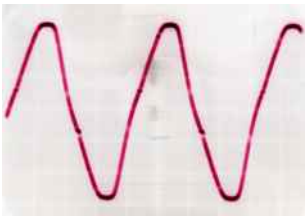
Een groot voordeel van deze schakeldynamiek is dat bij een toenemende (regel)sturing de belasting steeds gelijkmatiger wordt. Met discrete componten een dynamische perioden-regeling bouwen is een behoorlijk flinke klus, echter met een MCU en de al eerder gebruikte triac-module is dit 'relatief eenvoudig' te realiseren.

IA2109DPC DynamicPeriodControl mk1 Voor de besturing wordt gebruik gemaakt van de bij de AC-fase-aansnijding gebruikte elektronica (Arduino Nano, analog keypad, LCD, op IA2109AK-experimenteerprint) en de AC-triac-(dimmer)module met zero-cross detector.



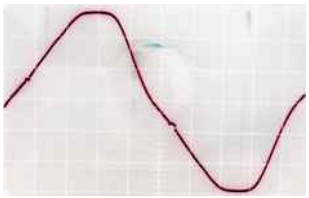
Hiernaast de **eerste nul-meting** van de schone netspanning van 230 VAC 50Hz, die tijdens deze meting enigszins uit balans is. Blijkbaar draaien er hier in de buurt nogal wat fase-aansnijding geregelde motoren (warmtepompen?) en schakelende voedingen die een versneld inzakken van de netspanning in het tweede en vierde kwadrant veroorzaken.

Als last wordt een 75 Watt gloeilamp gebruikt. Bij de **tweede nul-meting** wordt de netspanning door de triac-module 'geschakeld'.



'Geschakeld' in zoverre dat de triac-module-ingang continu Hoog gemaakt is. De triac-module veroorzaakt standaard dus schakelhikjes op de nuldoorgang.

Merk op dat de hikjes door de diac-zenerspanning niet op de nullijn liggen.



```
void setup()
  { Serial.begin(115200);
    pinMode(ZeroCrossIn, INPUT);
    pinMode(ZeroSecond, OUTPUT);
    //pinMode(ZeroMonitor, OUTPUT);
    pinMode(TriacGate,OUTPUT);
    lcd.begin(16, 2);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("  IA2109DPC mk1  ");
    lcd.setCursor(0,1);
    lcd.print("   Firing Test   ");
    ZeroFrequency = ZeroFrequency * 2;
    delay(2000);
    digitalWrite(TriacGate, HIGH);
  }

void loop()
  {  }
```

Voor het op tijd ontsteken van de triac *kàn* gebruik worden van geprogrammeerde timers en/ of interrupts. Een normale analoge triac-dimmer-regeling gebruikt simpelweg de net-frequentie. Bij de dynamische periodenregeling wordt een nuldoorgangsdetector gebruikt.

Het eerste wat geregegeld moet worden is het tijdig (-op de nuldoorgang-) in en uit kunnen schakelen van de halve perioden. Voor het binnenhalen van het nuldoorgangs-signaal wordt gebruik gemaakt van de functie `ReadZeroCross()`, die in `IA2109ACP mk1 AC Powercontrol` al behandeld is

Toegevoegd is uitgang `ZeroSecond`, die tijdens testen en ontwikkeling gebruikt kan worden als triggersignaal voor logic-analyser of de scoop.

```
void ReadZeroCross()
  { ZeroStatus = digitalRead(ZeroCrossIn);
    if ((ZeroStatus == HIGH) &&
        (ZeroPrev == LOW))
      { ZeroPrev = HIGH;
        ZeroFireFree = false;
        digitalWrite(TriacGate, LOW);
        //digitalWrite(ZeroMonitor, LOW);
        ZeroCrossCounter++;
        if (ZeroCrossCounter > ZeroFrequency)
          { ZeroCrossCounter = 1;
            digitalWrite(ZeroSecond, HIGH);
            //ZeroSeconds++;
          }
      }
    if ((ZeroStatus == LOW) &&
        (ZeroPrev == HIGH))
      { ZeroPrev = LOW;
        ZeroFireFree = true;
        //digitalWrite(ZeroMonitor, HIGH);
        digitalWrite(ZeroSecond, LOW);
      }
  }
```

De **derde nul-meting** is de programma-timing-test. We willen gebalanceerd halve perioden schakelen. Het volgen van dit soort signalen op een normale analoge scoop bij 50 Hz of lager, is zonder externe triggering nauwelijks fatsoenlijk zichtbaar te krijgen.

Als tussen-oplossing wordt gebruik gemaakt van Arduino's Seriele Monitor. `ReadZeroCross()` is compleet uitgeschakeld, en vervangen door een stukje testtimer-programmacode. De bedoeling is als test de periodehelften 1 en 4 in te schakelen tijdens een energie-periode van 6 halve perioden.

```
  delay(2000);
  //digitalWrite(TriacGate, HIGH);
  Serial.println("=====IA2109DPC mk1=====");
}

void loop()
{ //ReadZeroCross();
  Serial.print(HalfPeriodNumber);

  if (ZeroFireFree == true)
    { digitalWrite(TriacGate,HIGH);
      delayMicroseconds(TriacPulseLength);
      digitalWrite(TriacGate, LOW);
      ZeroFireFree = false;
      Serial.print(" fired ");
    }

  // ---testtimer-replaces ReadZeroCross--
  HalfPeriodNumber++;
  if (HalfPeriodNumber > 6)
    { HalfPeriodNumber = 1; }
  if ((HalfPeriodNumber == 1) ||
      (HalfPeriodNumber == 4))
    { ZeroFireFree = true; }
  delay(100);
  Serial.println("");
}
```

Via de seriele monitor is het inschakelgedrag over 6 halve perioden goed te volgen.

```
/dev/ttyUSB0
=====IA2109DPC mk1=====
1 fired
2
3
4 fired
5
6
1 fired
2
3
4 fired
5
6
1 fired
```

Straks, tijdens het daadwerkelijk aansturen van de triac-module kan de Seriele Monitor niet gebruikt worden, omdat deze te veel vertraging in de programma-executie geeft.

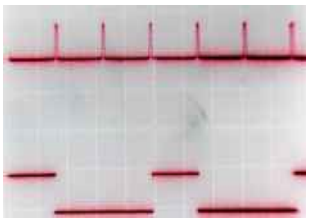
Als **vierde 'nul-meting'** wordt het daadwerkelijke schakelgedrag gecontroleerd.

```
// IA2109DPC mk1 DynamicPeriodControl; Harm J Seef, The Netherlands
//              Testing zero cross and half period firing control
//
// 2024 dec 17  HJS   Proof of concept  AC_Power_Control (0...100%)
//              mk1
// ============================================================================================

#include <LiquidCrystal.h>                          // Add library
const int rs=12, en=11, d4=8, d5=7, d6=5, d7=4;     // vars for IA2109 interface pinning
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);          // Activate LiquidCrystal
                                                    //
// ---controller vars----------------------------- //
// bool ControlStatus = 0;                          // 0 = Off , 1 = On
// float ControlPercent = 0;                        // Power control setting in % (0...100%)
                                                    //
int HalfPeriodNumber = 1;                           // half period counter
bool HalfPeriodOn = true;                           // true  = fire half period
                                                    // false = do not fire
                                                    //
// ---triac module-------------------------------- //
const int ZeroCrossIn = 9;                          // designate D9 as zero cross detector input
const int ZeroSecond = 13;                          // one pulse per second as external trigger out
int ZeroFrequency = 50;                             // mains frequency; 50 or 60 Hz
bool ZeroStatus = LOW;                              // signal level on D9
bool ZeroPrev = LOW;                                // previous level on D9
int ZeroCrossCounter = 0;                           // cross zero counter
unsigned long ZeroSeconds = 0;                      // seconds counter
bool ZeroFireFree = true;                           // block firing while zero crossing
                                                    //
const int TriacGate = 10;                           // designate D10 as output to triac gate
int TriacPulseLength = 4;                           // specs BTA16 says minimal 2 microseconds
int TriacFired = false;                             // false = not fired
const int TriacMonitor = 6;                         // monitoring
                                                    //
// ------------------------------------------------ //
void ReadZeroCross()                                // Detecting zero crossing...
  { ZeroStatus = digitalRead(ZeroCrossIn);          // read zero cross input
    if ((ZeroStatus == HIGH) &&                     // on raising edge
        (ZeroPrev == LOW))                          // (end current half period)
      { ZeroPrev = HIGH;                            // (re)set control vars
        ZeroFireFree = false;                       //
        digitalWrite(TriacGate, LOW);               // switch triac off (just in case ..)
        digitalWrite(TriacMonitor, LOW);            //
        //digitalWrite(ZeroMonitor, LOW);           //
        ZeroCrossCounter++;                         // raise counter
        HalfPeriodNumber++;                         //
        if (ZeroCrossCounter > ZeroFrequency)       // after n crossings; 50 Hz = 100, 60 Hz = 120
          { ZeroCrossCounter = 1;                   // reset cross counter
            digitalWrite(ZeroSecond, HIGH);         // switch trigger output High
            //ZeroSeconds++;                        // raise seconds counter (future clock)
          }                                         //
      }                                             //
    if ((ZeroStatus == LOW) &&                      // on falling edge
        (ZeroPrev == HIGH))                         // (start current half period)
      { ZeroPrev = LOW;                             // reset vars
        ZeroFireFree = true;                        //
        //digitalWrite(ZeroMonitor, HIGH);          //
        digitalWrite(ZeroSecond, LOW);              //
      }                                             //
  }                                                 //
                                                    //
void setup()                                        //
  { //Serial.begin(115200);                        //
    pinMode(ZeroCrossIn, INPUT);                    // zero cross input
```

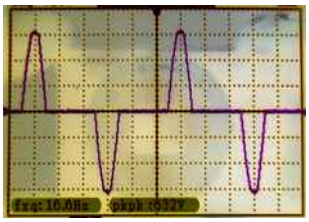
```
    pinMode(ZeroSecond, OUTPUT);                  // as external trigger out
    //pinMode(ZeroMonitor, OUTPUT);               //
    pinMode(TriacGate,OUTPUT);                    // signal to triac
    pinMode(TriacMonitor, OUTPUT);                //
    lcd.begin(16, 2);                             // Initialize LCD...
    lcd.clear();                                  // Reset LCD
    lcd.setCursor(0,0);                           // First line first position
    lcd.print("  IA2109DPC mk1  ");               // Show sketch name...
    lcd.setCursor(0,1);                           // Second line first position
    lcd.print("   Firing Test   ");               // Show application...
    ZeroFrequency = ZeroFrequency * 2;            // calculate default zero crossings per second
    delay(2000);                                  //
    //digitalWrite(TriacGate, HIGH);              //
    //Serial.println("=====IA2109DPC mk1=====");   //
  }                                               //
                                                  //
void loop()                                       //
  { ReadZeroCross();                              // read zero cross input
    //ZeroFireFree = true;                        //
    //Serial.print(HalfPeriodNumber);             // echo to monitor
                                                  //
    if ((ZeroFireFree == true) &&                 // if allowed to fire
        (HalfPeriodOn == true))                   //
      { digitalWrite(TriacGate,HIGH);             // triac on
        delayMicroseconds(TriacPulseLength);      // wait some microseconds
        digitalWrite(TriacGate, LOW);             // triac off
        digitalWrite(TriacMonitor, HIGH);         //
        ZeroFireFree = false;                     // reset control var
        HalfPeriodOn = false;                     //
        //Serial.print(" fired ");                //
      }                                           //
                                                  //
    // ---testtimer-replaces ReadZeroCross--      //
    //HalfPeriodNumber++;                         //
    if (HalfPeriodNumber > 6)                     //
      { HalfPeriodNumber = 1; }                   //
    if ((HalfPeriodNumber == 1) ||                //
        (HalfPeriodNumber == 4))                  //
      { HalfPeriodOn = true; }                    //
      else                                        //
      { HalfPeriodOn = false; }                   //
    //delay(100);                                 //
    //Serial.println("");                         //
  }                                               //
```

Op de afbeelding hiernaast is het bovenste signaal de nuldoorgangspuls van de triac-module. Het onderste signaal is de `TriacMonitor`, ofwel op een vermogensperiode van zes perioden, worden de halve perioden 1 en 4 worden doorgelaten. Dit maakt een verhouding 2 op 6, ofwel van 1 op 3, ofwel een vermogensaansturing van 33%.



Op de afbeelding hiernaast -afkomstig van een parallel geschakelde 'meettransformator' laat duidelijk de geschakelde 230 VAC halve perioden zien.

De nullijn-uitschakel-overshoot hiernaast wordt veroorzaakt door de meettransformator.



Ter controle is een differentieel-meting over de last zonder meettransformator uitgevoerd. Dit laat zien dat met een standaard triac-module in principe dus een halve-periodensturing te realiseren is.

Gloeilamp-inschakelgedrag. Bij de hier gebruikte sturing van 33% is het gedrag van de 75 Watt gloeilamp opmerkelijk. Na het inschakelen van de lamp gaat deze niet direct gloeien, maar wordt een inschakelstroom gemeten van 0,35 A_{RMS} (ca. 80 Watt). Na een willekeurig aantal zwakke inschakelgloeiingen, begint de lamp eerst zwak te knipperen, om even later wat sterker te gaan knipperen. Duidelijk is dat de gloeidraad eerst voor- en doorgewarmd moet zijn, voordat ze daadwerkelijk gaat gloeien. Dit betekent dat de normale inschakelstroom inderdaad gehalveerd is. Na enige tijd stabiliseert de lamp zich op 0,24 A_{RMS} (ca. 55 Watt).

Verwarming-inschakelgedrag. Het inschakelgedrag is ook geobserveerd met een Eurom Safe-T-Shine 900 Watt infra-rood verwarming. Hier zien we dezelfde inschakelverschijnselen. In eerste instantie lijkt er niets te gebeuren, en is de stroom 1,5 A_{RMS}. Even later begint de kachel wat warmte uit te stralen, en nog even later beginnen de neon-schakelaars af en toe te knipperen. Na een aantal minuten is de temperatuur van de elementen opgelopen tot ca. 370 °C, flikkeren de neon-schakelaars continu en wordt 2,2 A_{RMS} verbruikt (ca. 506 Watt).

Conclusie. De opzet en wijze van halve perioden-regeling zorgt voor gedempte inschakelstromen en een evenwichtige belasting, zowel voor de last als voor het net. Wat ook duidelijk is, is dat de vermogenskarakteristiek van de last niet lineair is. Welk type grafiek dat dan is, kan met een gecontroleerde periodenregeling herleid worden.

Omdat de regeling effectief binnen één seconde schakelt, zal een kWh-meter bij gebruik van zonnestroom de energiestromen salderen. ~~"Zal"! Of dat ook daadwerkelijk gebeurt moet in de praktijk nog getest worden.~~

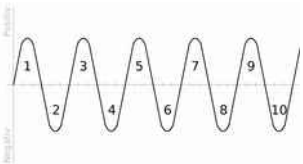
IA2109DPC DynamicPeriodControl mk2

In versie mk1 zijn de nodige nul-metingen uitgevoerd, en is het regelprincipe voor een halve-periodenregeling beproefd. In versie mk2 wordt de DPC verder ontwikkeld.

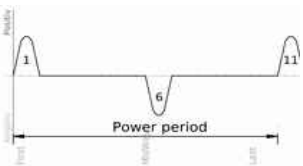
Voor de besturing wordt voorlopig gebruik gemaakt van de bij de AC-fase-aansnijding gebruikte elektronica (Arduino Nano, analog keypad, LCD, op IA2109AK-experimenteerprint) en de AC-triac-(dimmer)module met zero-cross detector.

Voor de bediening en verwerking via het analoge keypad wordt nu ook gebruik gemaakt van de eerder gebruikte functies `ReadKeyVal(), ReadKeyPad() en KeyHandle()`.

IntervalControl(). Een nieuwe toevoeging is functie `IntervalControl()`, waarin de schakel-intervaltijd 'geconstrueerd' wordt. 'Geconstrueerd' omdat er voor een regelbereik met integers van 0 tot 100% bij 50 te schakelen perioden geen eenvoudige simpele wiskundige (booleaanse) formule voor de regeling te herleiden is (tenminste niet dat ik het weet of kan). De oplossing is daarom gezocht in getalreeksen en (golf)patronen.



Wanneer grafisch meerdere perioden achter elkaar worden weergegeven, waarbij de eerste halve periode positief is, kunnen de opeenvolgende halve perioden genummerd worden als 1, 2, 3, 4, 5, 6, … enz. Merk op dat alle positieve halve perioden **oneven** genummerd zijn, en alle negatieve perioden **even** genummerd zijn.



Stel, dat van een reeks van 10 halve perioden, de eerste en de zesde doorgelaten worden, dan wordt de effectieve waarde van de eerste halve positieve periode verdeeld over 5 halve perioden. Hetzelfde geldt voor de zesde negatieve halve periode; ook die wordt verdeeld over 5 halve perioden. De eerste vijf en tweede vijf vanaf zes vormen samen één nieuwe **energie-periode**.

De dynamische perioden-regeling begint in het lage regelbereik met het **interval**-schakelen van **halve-perioden**, waarmee een **energie-periode** wordt geconstrueerd. Voor het gebalanceerd belasten van last en net (het moet geen 'gelijkgerichte wisselspanning' worden), **moet** een **oneven** periodehelft altijd gevolgd worden door een **even** periodehelft.

Dit betekent ook dat een energieperiode altijd een **even** aantal perioden groot is. Dit vereenvoudigt de programmering, en vermindert ook een eventuele kans op 'terugslag' (of op-/uitslingering) bij niet zuiver ohmse lasten. Om dit met een MCU-programmering binnen een regelcyclustijd van 1 seconde te realiseren is complex vanwege de vele nodige mitsen, maren, uitzonderingen en correcties. Wat men ook probeert, er ontstaan altijd afrondingsverschillen, limieten en schakel-hikken.

Energie-periode. Een veel praktischer oplossing is focussen op het gebruik van starten met één oneven periodehelft (‘First’) en een even interval tussen de periodehelften (D, ‘MidWay’); elke energie-periode-helft is dan automatisch afwisselend ‘positief en negatief’.

De energie-periode is in principe dan gelijk aan tweemaal de intervaltijd, en dus altijd een even getal (E, ‘Last’). Ofwel, bij de ‘constructie’ wordt de 1 seconde-watt-meettijd even compleet genegeerd. De focus ligt nu helemaal op het aanmaken van een energie-periode (E).

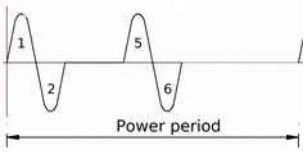
Voorwaarde bij ‘één-periodehelft-schakelen’ (‘TrainLength=1’) is dat een (oneven) positieve periodehelft gevolgd moet worden door een (even) negatieve periodehelft, en andersom, en dat na de energie-periodetijd (E, ‘Last’), de energie-periode eindigt (resetten halve periodenteller).

In de tabel hiernaast zijn eerst de theoretische waarden uitgerekend, die in een eerste versie van functie `IntervalControl()` geprogrameerd was. In de praktijk blijkt dat op deze wijze een halve-periode-regeling van 0 tot 17% gemaakt kan worden. Vanaf 17% begint de ‘regelbaarheid’ echter vast te lopen.

A Procent	B Train-Length	C interval	D MidWay	E P-periode-duur (2xD) (Last)
1	1	100,0	100	200
2	1	50,00	50	100
3	1	33,33	34	68
4	1	25,00	24	50
5	1	20,00	20	40
6	1	16,67	16	34
7	1	14,29	14	28
8	1	12,50	12	26
9	1	11,11	10	22
10	1	10,00	10	20
11	1	9,09	8	18
12	1	8.33	8	16
13	2	7,69	15	15
14	2	7,14	15	14
15	2	6,67	13	13

In de praktijk ontstaan de problemen echter nog eerder. Omdat de oneven en even halve periode elkaar altijd opvolgen, moet de energie-periodeduur dus altijd een even aantal perioden beslaan.

TrainLength. Als oplossing wordt de halve-periode-regeling vanaf 17% opgeschakeld naar een **twee-halve-periode-regeling**, ofwel we zetten er ‘een tandje’ bij. Het resultaat is dan een **halve-perioden-treintje** van twee halve perioden (TrainLength=2 Wagons).



De eerste helft van de energie-periode bestaat dan uit het treintje van de halve perioden 1 en 2, en de volgende helft uit treintje (bijvoorbeeld) perioden 5 en 6, of een veelvoud daarvan.

Door afronding van de allerlaatste halve-periodehelft op een even nummer kan de energie-periode wat a-symmetrisch worden, maar het effectief geregeld vermogen is nog steeds redelijk correct. Een groot verschil met het bereik van 0-17% is dat nu iedere helft van de energie-periode (MidWay) met een oneven periode gestart moet worden.

De oplossing van een langere halve-periodentrein lijkt eenvoudig, maar de regelbaarheid loopt nu nog sneller vast. Bij elke verlenging van de halve-periodentrein (TrainLength) worden de problemen groter en groter.

Wat veel erger is, is dat door de gewenste specificaties, aanvullende eisen van de nodige mitsen, maren en uitzonderingen de regeling niet meer te programmeren is. Conclusie; idee is goed, praktische uitwerking onhaalbaar.

```
Status:Off  Control:13%  Sequence:1  Interval:7.69  Rounded as:8  Last:15
Status:Off  Control:14%  Sequence:1  Interval:7.14  Rounded as:6  Last:14
Status:Off  Control:15%  Sequence:1  Interval:6.67  Rounded as:6  Last:13
Status:Off  Control:16%  Sequence:1  Interval:6.25  Rounded as:6  Last:12
Status:Off  Control:17%  Sequence:1  Interval:5.88  Rounded as:6  Last:11
Status:Off  Control:18%  Sequence:2  Interval:11.11  Rounded as:11  Last:22
Status:Off  Control:19%  Sequence:2  Interval:10.53  Rounded as:11  Last:21
Status:Off  Control:20%  Sequence:2  Interval:10.00  Rounded as:11  Last:20
Status:Off  Control:21%  Sequence:2  Interval:9.52  Rounded as:11  Last:19
Status:Off  Control:22%  Sequence:2  Interval:9.09  Rounded as:9  Last:18
Status:Off  Control:23%  Sequence:2  Interval:8.70  Rounded as:9  Last:17
Status:Off  Control:24%  Sequence:2  Interval:8.33  Rounded as:9  Last:16
Status:Off  Control:25%  Sequence:2  Interval:8.00  Rounded as:9  Last:16
Status:Off  Control:26%  Sequence:2  Interval:7.69  Rounded as:9  Last:15
Status:Off  Control:27%  Sequence:2  Interval:7.41  Rounded as:7  Last:14
Status:Off  Control:28%  Sequence:2  Interval:7.14  Rounded as:7  Last:14
Status:Off  Control:29%  Sequence:2  Interval:6.90  Rounded as:7  Last:13
Status:Off  Control:30%  Sequence:2  Interval:6.67  Rounded as:7  Last:13
Status:Off  Control:31%  Sequence:2  Interval:6.45  Rounded as:7  Last:12
Status:Off  Control:32%  Sequence:2  Interval:6.25  Rounded as:7  Last:12
Status:Off  Control:33%  Sequence:2  Interval:6.06  Rounded as:7  Last:12
Status:Off  Control:34%  Sequence:2  Interval:5.88  Rounded as:7  Last:11
Status:Off  Control:35%  Sequence:2  Interval:5.71  Rounded as:7  Last:11
```

LookUp Table. Om de programmeerproblemen te omzeilen is daarom gekozen om gebruik te maken van een **opzoektabel** ***(LookUp Table, LUT)***. Net als in het pré-rekenmachine-tijdperk, zijn de diverse instellingen vooraf uitgerekend, en zijn de resultaten als 'opzoekwaarden' in een tabel geplaatst.

```
// define a 2D array: 51 rows, 3 columns in EPROM
// 0 percent, TrainLength=0 , MidWay=0   , Last=0   // Off
// 1 percent, TrainLength=1 , MidWay=100 , Last=200
// 2 percent, TrainLength=2 , Mid....
// 3 percent, Trai...
// 4 perc..
// 5
// 6
```

Om Arduino's RAM-geheugen zoveel mogelijk vrij te houden van overbidig rekenwerk, wordt de lookup tabel met PROGMEM in het programmageheugen geplaatst.

```
// ---LookUp Table----------------
int LookUpRow = 0;
//int LookUpCol = 0;
//int LookUp[m][n] =
const int LookUp[51][3] PROGMEM =
  { { 0, 0, 0 },
    { 1, 100, 200 },
    { 1, 50, 100 },
    { 1, 34, 68 },
    { 1, 24, 50 },
    { 1, 20, 40 },
```

Met functie `IntervalControl()` wordt op basis van de gewenste procentuele aansturing, de daarvoor benodigde parameters vanuit het programma-geheugen ingelezen.

```
void IntervalControl()
  { LookUpRow = ControlPercent;
    TrainLength = pgm_read_word(&
      LookUp[LookUpRow][0]);
    MidWay = pgm_read_word(&
      LookUp[LookUpRow][1]);
    Last = pgm_read_word(&
      LookUp[LookUpRow][2]);
    //Serial.print("LookUp: ");
```

```
// IA2109DPC mk2 DynamicPeriodControl; Harm J Seef, The Netherlands
//
// 2024 dec 17  HJS   Proof of concept  AC_Period_Power_Control (0...100%)
// 2025 jan 27  HJS   LookUp Table added (<50%).
//
// ===============================================================================================

#include <LiquidCrystal.h>                      // Add library
const int rs=12, en=11, d4=8, d5=7, d6=5, d7=4;     // vars for IA2109 interface pinning
```

```
      if ((ZeroFireFree == true) &&             // if released to fire and
          (HalfPeriodOn == true))               // half period should be fired
        { digitalWrite(TriacGate,HIGH);         // triac on
          delayMicroseconds(TriacPulseLength);  // wait some microseconds
          digitalWrite(TriacGate, LOW);         // triac off
          digitalWrite(TriacMonitor, HIGH);     //
          //Serial.print(HalfPeriodCounter);    //
          //Serial.print(" fired seq:");        //
          //Serial.print(TrainWagon);           //
          //Serial.println();                   //
          ZeroFireFree = false;                 // block firing
          HalfPeriodOn = false;                 //
          TrainWagon--;                         // lower counter
          if (TrainWagon > 0 )                  // if sequence not finished yet
            { HalfPeriodOn = true; }            // set control var
        }                                       //
                                                //
      // ---2e half powerperiod---------------- //
      if (HalfPeriodCounter == MidWay)          // at next sequence
        { if ( (TrainWagon == 0) &&             //
               (ZeroFireFree == true) )         //
            { TrainWagon = TrainLength;         // start down counting from...
              HalfPeriodOn = true;              // set default
            }                                   //
        }                                       //
                                                //
      // ---reset to first half period---       //
      if (HalfPeriodCounter > Last)             // if last half period past by
        { TrainWagon = TrainLength;             //
          HalfPeriodCounter = 1;                // reset default vars
          HalfPeriodOn = true;                  //
        }                                       // restart at first half period
      //Serial.print(" fired seq:");            //
      //Serial.print(TrainWagon);               //
      //Serial.println();                       //
    }                                           //
}                                               //
```

```
=====IA2109DPC mk2=====
LookUp: 1%  Trainlength: 1   Midway: 100   Last: 200   Real Procent: 1.00
LookUp: 2%  Trainlength: 1   Midway: 50   Last: 100   Real Procent: 2.00
LookUp: 3%  Trainlength: 1   Midway: 34   Last: 68   Real Procent: 2.94
LookUp: 4%  Trainlength: 1   Midway: 24   Last: 50   Real Procent: 4.00
LookUp: 5%  Trainlength: 1   Midway: 20   Last: 40   Real Procent: 5.00
LookUp: 6%  Trainlength: 1   Midway: 16   Last: 34   Real Procent: 5.88
LookUp: 7%  Trainlength: 1   Midway: 14   Last: 28   Real Procent: 7.14
LookUp: 8%  Trainlength: 1   Midway: 12   Last: 26   Real Procent: 7.69
LookUp: 9%  Trainlength: 1   Midway: 10   Last: 22   Real Procent: 9.09
LookUp: 10%  Trainlength: 1   Midway: 10   Last: 20   Real Procent: 10.00
LookUp: 11%  Trainlength: 1   Midway: 8   Last: 18   Real Procent: 11.11
LookUp: 12%  Trainlength: 1   Midway: 8   Last: 16   Real Procent: 12.50
LookUp: 13%  Trainlength: 2   Midway: 15   Last: 30   Real Procent: 13.33
LookUp: 14%  Trainlength: 2   Midway: 15   Last: 28   Real Procent: 14.29
LookUp: 15%  Trainlength: 2   Midway: 13   Last: 26   Real Procent: 15.38
LookUp: 16%  Trainlength: 3   Midway: 18   Last: 38   Real Procent: 15.79
LookUp: 17%  Trainlength: 2   Midway: 11   Last: 24   Real Procent: 16.67
LookUp: 18%  Trainlength: 2   Midway: 11   Last: 22   Real Procent: 18.18
LookUp: 19%  Trainlength: 3   Midway: 16   Last: 32   Real Procent: 18.75
LookUp: 20%  Trainlength: 1   Midway: 4   Last: 10   Real Procent: 20.00
LookUp: 21%  Trainlength: 3   Midway: 16   Last: 28   Real Procent: 21.43
LookUp: 22%  Trainlength: 4   Midway: 17   Last: 36   Real Procent: 22.22
LookUp: 23%  Trainlength: 3   Midway: 14   Last: 26   Real Procent: 23.08
LookUp: 24%  Trainlength: 4   Midway: 17   Last: 34   Real Procent: 23.53
LookUp: 25%  Trainlength: 1   Midway: 4   Last: 8   Real Procent: 25.00
LookUp: 26%  Trainlength: 5   Midway: 20   Last: 38   Real Procent: 26.32
LookUp: 27%  Trainlength: 3   Midway: 10   Last: 22   Real Procent: 27.27
LookUp: 28%  Trainlength: 5   Midway: 18   Last: 36   Real Procent: 27.78
LookUp: 29%  Trainlength: 2   Midway: 7   Last: 14   Real Procent: 28.57
LookUp: 30%  Trainlength: 3   Midway: 10   Last: 20   Real Procent: 30.00
LookUp: 31%  Trainlength: 4   Midway: 13   Last: 26   Real Procent: 30.77
LookUp: 32%  Trainlength: 7   Midway: 22   Last: 44   Real Procent: 31.82
LookUp: 33%  Trainlength: 1   Midway: 4   Last: 6   Real Procent: 33.33
LookUp: 34%  Trainlength: 8   Midway: 23   Last: 46   Real Procent: 34.78
LookUp: 35%  Trainlength: 7   Midway: 20   Last: 40   Real Procent: 35.00
```

~~Hiernaast wordt voor elke procent-instelling de daarvoor gebruikte parameters weergegeven, en de na-berekening van het daadwerkelijke percentage.~~

~~Merk op dat bepaalde percentage-instellingen bijzonder lastig te realiseren zijn.~~

~~In dat geval is gekozen voor een instelling zo veel mogelijk tussen de voorgaande en volgende percentage.~~

~~Helaas geldt wel dat hoe langer het treintje wordt, des te stotender het regelen wordt.~~

Gloeilamp-regelgedrag. De hierboven weergegeven regeling is getest op een 75 Watt gloeilamp. Na het inschakelen bij 1% gaat de lamp zwak gloei-knipperen bij een stroom van 0,07...0,112 A_{RMS}. Dit is uiteraard niet de juiste effectieve waarde, maar geeft wel een indruk. Bij 3% is de flikkering redelijk stabiel. De zwakke flikkering neemt toe tot 12%, om bij 13% te veranderen in een intensievere langzamere knippering. De grotere intensiteit wordt veroorzaakt doordat vanaf 13% een twee-halve-perioden-treintje gebruikt wordt. Deze afwisseling van flikker-frequentie en intensiteit loopt door tot 50%, waarbij dan 0,277A_{RMS} (ca. 63 Watt) gemeten wordt.

IR-verwarming-schakelgedrag. De regeling is ook getest met een Eurom Safe-T-Shine 900 Watt infra-rood verwarming. Hier zien we met de gloeilamp vergelijkbare schakelverschijnselen.

In eerste instantie lijkt er niets te gebeuren, en is de stroom 0,1...0,5 A_{RMS}. en wordt zonder gloeien verwarmd tot 32°C. Vanaf 5% (123°C) is een schakeltikkend geluid te horen, die bij elke volgende instelling steeds zachter wordt. 10% levert 205°C. Vanaf ca 18% fluctueert de tl-verlichting maar is het schakelgeluid verdwenen. Vanaf 31% (379°C) beginnen de elementen roze te kleuren. Bij 33% zijn de tl-verlichtings-fluctueringen verdwenen, die met verder omhoog regelen weer verschijnen. Vanaf 36% kon de temperatuur niet meer gemeten worden (einde schaal). Bij 44% is er een hinderlijke tl-licht-resonantie, die bij 50% weer helemaal verdwenen is.

Tafel bbq. De meeste boilers zijn voorzien van een dompel-verwarmingselement, ofwel een 'nat-verwarmingselement' (eenzelfde type als ook in de wasmachine). Verwonderd door het schakelgedrag van de infra-rood-straler, een snelle controlemeting uitgevoerd met een elektrische tafel-bbq van 500 Watt; niet helemaal vergelijkbaar, maar wel indicatief.

Hoewel de ohmse weerstand van een gloeispriraal behoorlijk snel 'last'-variabel is, is het niet zo erg als bij een IR-straler. De tl-licht-fluctuaties waren fors minder erg.

Conclusies. De dynamische halve perioden-sturing is rekenkundig correct, alleen de oplossing van 'tandje bijschakelen' veroorzaakt bij bepaalde percentages wel vervelende piekpiek stotende 'knipper-resonanties', die bij zware belastingen op het net terugwerken (wat te verwachten is). In welke mate dit is, is weer afhankelijk van het type belasting. En in hoeverre de overige netgebruikers daarvan hinder zullen ondervinden is dan weer een andere vraag.

Op zich zijn de 'knipper-resonanties' (op het einde van een lange leiding van een meterkast-groep) niet vreemd. Een procent-regeling op 50 Hz willen gebruiken is ongeveer hetzelfde als één orgelpijp geschikt te willen maken voor alle tonen (de gekozen oplossing lijkt het meest op de werking van een schuiftrompet).

De belangrijkste conclusie is dat de dynamische periodensturing precies dat doet waarvoor ze ontworpen is, namelijk het 'salderen' van de energiestromen. Maar …, als gevolg van DPC is een correcte effectieve stroommeting met een digitale TRMS-multimeter niet meer mogelijk. Voor een goede indicatie zou eigenlijk I_{gem} gemeten moeten worden over 2 seconden.

~~Al met al is de verwachting dat een feraris-kWh-meter inderdaad stil kan staan en dus nul-op-de-meter laat zien. Ook alle digitale kWh-metera zouden nul-op-de-meter moeten laten zien. De praktijk zal het moeten leren.~~ Met de tot dusver gebruikte test-opstellingen kon allemaal nul-op-de-meter geregeld worden.

IA2109DPC DynamicPeriodControl mk3 In versie mk3 is de lookup table uitgebreid naar 100%, is de seriële monitor, alle terugmeldingen en debug/monitor-uitgangen uitgezet. Hieronder een overzicht van de gewenste en gerealiseerde percentages. Op de een of andere manier lukt 34% nog niet.

```
LookUp: 1   Real Procent: 1.00
LookUp: 2   Real Procent: 2.00
LookUp: 3   Real Procent: 2.94
LookUp: 4   Real Procent: 4.00
LookUp: 5   Real Procent: 5.00
LookUp: 6   Real Procent: 5.88
LookUp: 7   Real Procent: 7.14
LookUp: 8   Real Procent: 7.69
LookUp: 9   Real Procent: 9.09
LookUp: 10   Real Procent: 10.00
LookUp: 11   Real Procent: 11.11
LookUp: 12   Real Procent: 12.50
LookUp: 13   Real Procent: 13.33
LookUp: 14   Real Procent: 14.29
LookUp: 15   Real Procent: 15.38
LookUp: 16   Real Procent: 15.79
LookUp: 17   Real Procent: 16.67
LookUp: 18   Real Procent: 18.18
LookUp: 19   Real Procent: 18.75
LookUp: 20   Real Procent: 20.00
LookUp: 21   Real Procent: 21.43
LookUp: 22   Real Procent: 22.22
LookUp: 23   Real Procent: 23.08
LookUp: 24   Real Procent: 23.53
LookUp: 25   Real Procent: 25.00
```

```
LookUp: 26   Real Procent: 26.32
LookUp: 27   Real Procent: 27.27
LookUp: 28   Real Procent: 27.78
LookUp: 29   Real Procent: 28.57
LookUp: 30   Real Procent: 30.00
LookUp: 31   Real Procent: 30.77
LookUp: 32   Real Procent: 31.82
LookUp: 33   Real Procent: 33.33
LookUp: 34   Real Procent: 34.78
LookUp: 35   Real Procent: 35.00
LookUp: 36   Real Procent: 36.00
LookUp: 37   Real Procent: 36.84
LookUp: 38   Real Procent: 37.50
LookUp: 39   Real Procent: 38.89
LookUp: 40   Real Procent: 40.00
LookUp: 41   Real Procent: 40.91
LookUp: 42   Real Procent: 41.67
LookUp: 43   Real Procent: 42.86
LookUp: 44   Real Procent: 44.44
LookUp: 45   Real Procent: 45.00
LookUp: 46   Real Procent: 46.15
LookUp: 47   Real Procent: 47.06
LookUp: 48   Real Procent: 48.00
LookUp: 49   Real Procent: 48.57
LookUp: 50   Real Procent: 50.00
```

```
LookUp: 50   Real Procent: 50.00
LookUp: 51   Real Procent: 51.43
LookUp: 52   Real Procent: 52.00
LookUp: 53   Real Procent: 52.94
LookUp: 54   Real Procent: 53.85
LookUp: 55   Real Procent: 55.00
LookUp: 56   Real Procent: 56.00
LookUp: 57   Real Procent: 57.14
LookUp: 58   Real Procent: 58.06
LookUp: 59   Real Procent: 58.82
LookUp: 60   Real Procent: 60.00
LookUp: 61   Real Procent: 61.11
LookUp: 62   Real Procent: 62.07
LookUp: 63   Real Procent: 62.96
LookUp: 64   Real Procent: 64.00
LookUp: 65   Real Procent: 65.22
LookUp: 66   Real Procent: 65.52
LookUp: 67   Real Procent: 66.67
LookUp: 68   Real Procent: 68.00
LookUp: 69   Real Procent: 69.23
LookUp: 70   Real Procent: 70.00
LookUp: 71   Real Procent: 70.83
LookUp: 72   Real Procent: 72.00
LookUp: 73   Real Procent: 73.08
LookUp: 74   Real Procent: 74.07
LookUp: 75   Real Procent: 75.00
```

```
LookUp: 76   Real Procent: 76.00
LookUp: 77   Real Procent: 76.92
LookUp: 78   Real Procent: 77.78
LookUp: 79   Real Procent: 78.95
LookUp: 80   Real Procent: 80.00
LookUp: 81   Real Procent: 80.95
LookUp: 82   Real Procent: 81.82
LookUp: 83   Real Procent: 83.33
LookUp: 84   Real Procent: 84.21
LookUp: 85   Real Procent: 85.00
LookUp: 86   Real Procent: 85.71
LookUp: 87   Real Procent: 86.67
LookUp: 88   Real Procent: 88.00
LookUp: 89   Real Procent: 88.89
LookUp: 90   Real Procent: 90.00
LookUp: 91   Real Procent: 90.91
LookUp: 92   Real Procent: 92.00
LookUp: 93   Real Procent: 93.00
LookUp: 94   Real Procent: 94.00
LookUp: 95   Real Procent: 95.00
LookUp: 96   Real Procent: 96.00
LookUp: 97   Real Procent: 97.00
LookUp: 98   Real Procent: 98.00
LookUp: 99   Real Procent: 99.00
LookUp: 100   Real Procent: 100.00
```

```
// IA2109DPC mk3 DynamicPeriodControl; Harm J Seef, The Netherlands
//
// 2024 dec 17  HJS   Proof of concept  AC_Period_Power_Control (0...100%)
// 2025 jan 27  HJS   LookUp Table added (<50%).
// 2025 jan 30  HJS   LookUp Table finished (0...100%)
//
// ================================================================================================

#include <LiquidCrystal.h>                          // Add library
const int rs=12, en=11, d4=8, d5=7, d6=5, d7=4;     // vars for IA2109 interface pinning
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);          // Activate LiquidCrystal
                                                    //
// ---keypad vars---------------------------------- //
const int KeyPadIn = A0;                            // designate A0 als keypad input
int KeyVal = 0;                                     // Analog ADC-value (0...1023)
int KeyValCount = 0;                                // samples counter
int KeyValCountMax = 20;                            // max number of samples
int KeyValMax = 0;                                  // recorded maximal keyval
int KeyKey = 0;                                     // Key pressed; 0 (none) ... 9
bool KeyStatus = false;                             // key pressed; false=released, true = pressed down
unsigned long KeyMarker = 0;                        // time marker when pressed key is executed (used for key repeat)
                                                    //
#define DefKeyNone  0                               // replace "DefKeyNone" with "0"
#define DefKeySet   1                               // replace...
#define DefKeyRight 2                               //
#define DefKeyReset 3                               //
#define DefKeyDown  4                               //
#define DefKeyOk    5                               //
#define DefKeyUp    6                               //
#define DefKeyEsc   7                               //
#define DefKeyLeft  8                               //
#define DefKeyMode  9                               //
                                                    //
// ---controller vars------------------------------ //
bool ControlStatus = 0;                             // 0 = Off , 1 = On
float ControlPercent = 0;                           // Power control setting in % (0...100%)
int HalfPeriodCounter = 1;                          // half period counter
int TrainLength = 1;                                // number of half periods to fire
int TrainWagon = 1;                                 // current half period counter to fire
int MidWay = 0;                                     // rounded even number of half periods
int Last = 0;                                       // total numbers of half periods
bool HalfPeriodOn = true;                           // true  = fire period
                                                    // false = do not fire
                                                    //
// ---triac module--------------------------------- //
const int ZeroCrossIn = 9;                          // designate D9 as zero cross detector input
//const int ZeroSecond = 6;                         // one pulse per second as external trigger out
//const int TriacMonitor = 13;                      // monitoring
int ZeroFrequency = 50;                             // mains frequency; 50 or 60 Hz
bool ZeroStatus = LOW;                              // signal level on D9  ???
bool ZeroPrev = LOW;                                // previous level on D9   ???
int ZeroCrossCounter = 0;                           // cross zero counter
unsigned long ZeroSeconds = 0;                      // seconds counter
bool ZeroFireFree = false;                          //
const int TriacGate = 10;                           // designate D10 as output to triac gate
int TriacPulseLength = 4;                           // specs BTA16 says minimal 2 microseconds
int TriacFired = 0;                                 // 0 = not fired
                                                    // 1 = fired
// ---lcd vars------------------------------------- //
bool SyncLcd = true;                                // control var for synchronising LCD
                                                    //
// ---LookUp Table-------------------------------- //
int LookUpRow = 0;                                  // Row used as percentage-index 0 ... 100
//int LookUpCol = 0;                                //
//int LookUp[m][n] =                                // define a 2D array; m-rows, n-columns
const int LookUp[101][3] PROGMEM =                  // define a 2D array; 101 rows, 3 columns in EPROM
  { { 0, 0, 0 },                                    // 0 percent, TrainLength=0 , MidWay=0   , Last=0   // Off
    { 1, 100, 200 },                                // 1 percent, TrainLength=1 , MidWay=100 , Last=200
    { 1, 50, 100 },                                 // 2 percent, TrainLength=2 , Mid....
    { 1, 34, 68 },                                  // 3 percent, Trai...
    { 1, 24, 50 },                                  // 4 perc..
    { 1, 20, 40 },                                  // 5
```

```
    { 1, 16, 34 },                          // 6
    { 1, 14, 28 },                          // 7
    { 1, 12, 26 },                          // 8
    { 1, 10, 22 },                          // 9
    { 1, 10, 20 },                          // 10
    { 1,  8, 18 },                          // 11
    { 1,  8, 16 },                          // 12
    { 2, 15, 30 },                          // 13
    { 2, 15, 28 },                          // 14
    { 2, 13, 26 },                          // 15
    { 3, 18, 38 },                          // 16
    { 2, 11, 24 },                          // 17
    { 2, 11, 22 },                          // 18
    { 3, 16, 32 },                          // 19
    { 1,  4, 10 },                          // 20
    { 3, 16, 28 },                          // 21
    { 4, 17, 36 },                          // 22
    { 3, 14, 26 },                          // 23
    { 4, 17, 34 },                          // 24
    { 1,  4,  8 },                          // 25
    { 5, 18, 38 },                          // 26
    { 3, 10, 22 },                          // 27
    { 5, 18, 36 },                          // 28
    { 2,  7, 14 },                          // 29
    { 3, 10, 20 },                          // 30
    { 4, 13, 26 },                          // 31
    { 7, 22, 44 },                          // 32
    { 1,  4,  6 },                          // 33
    { 8, 23, 46 },                          // 34  >>>HOLD
    { 7, 20, 40 },                          // 35
    { 9, 24, 50 },                          // 36
    { 7, 18, 38 },                          // 37
    { 3,  8, 16 },                          // 38
    { 7, 18, 36 },                          // 39
    { 2,  5, 10 },                          // 40
    { 9, 22, 44 },                          // 41
    { 5, 12, 24 },                          // 42
    { 9, 20, 42 },                          // 43
    { 4,  9, 18 },                          // 44
    { 9, 20, 40 },                          // 45
    { 6, 13, 26 },                          // 46
    { 8, 17, 34 },                          // 47
    { 12, 25, 50 },                         // 48
    { 17, 32, 70 },                         // 49
    {  1,  2,  4 },                         // 50
    { 18, 35, 70 },                         // 51
    { 13, 24, 50 },                         // 52
    {  9, 16, 34 },                         // 53
    {  7, 12, 26 },                         // 54
    { 11, 20, 40 },                         // 55
    { 14, 25, 50 },                         // 56
    {  4,  7, 14 },                         // 57
    { 18, 31, 62 },                         // 58
    { 10, 17, 34 },                         // 59
    {  3,  4, 10 },                         // 60
    { 11, 18, 36 },                         // 61
    { 18, 28, 58 },                         // 62
    { 17, 26, 54 },                         // 63
    { 16, 24, 50 },                         // 64
    { 15, 22, 46 },                         // 65
    { 19, 28, 58 },                         // 66
    {  4,  7, 12 },                         // 67
    { 17, 24, 50 },                         // 68
    {  9, 12, 26 },                         // 69
    {  7,  9, 20 },                         // 70
    { 17, 24, 48 },                         // 71
    { 18, 25, 50 },                         // 72
    { 19, 26, 52 },                         // 73
    { 20, 27, 54 },                         // 74
    {  3,  4,  8 },                         // 75
    { 19, 24, 50 },                         // 76
    { 10, 13, 26 },                         // 77
    { 14, 17, 36 },                         // 78
    { 15, 18, 38 },                         // 79
    { 12, 15, 30 },                         // 80
    { 17, 20, 42 },                         // 81
    {  9, 10, 22 },                         // 82
    { 10, 11, 24 },                         // 83
    { 16, 19, 38 },                         // 84
    { 17, 20, 40 },                         // 85
    { 12, 13, 28 },                         // 86
    { 26, 29, 60 },                         // 87
    { 22, 25, 50 },                         // 88
    {  8,  9, 18 },                         // 89
    {  9, 10, 20 },                         // 90
    { 10, 11, 22 },                         // 91
    { 23, 24, 50 },                         // 92
    { 93,  100, 200 },                      // 93
    { 47,   50, 100 },                      // 94
    { 19,   20,  40 },                      // 95
    { 24,   25,  50 },                      // 96
    { 97,  100, 200 },                      // 97
    { 49,   50, 100 },                      // 98
    { 99,  100, 200 },                      // 99
    {  2,    3,   4 }                       // 100
  };                                        //
                                            //
// ------------------------------------------------ //
void ReadZeroCross()                        // Detecting zero crossing...
  { ZeroStatus = digitalRead(ZeroCrossIn);  // read zero cross input
    if ((ZeroStatus == HIGH) &&             // on raising edge
        (ZeroPrev == LOW))                  // (end current half period)
      { ZeroPrev = HIGH;                    // (re)set control vars
```

```
        ZeroFireFree = false;                       //
        digitalWrite(TriacGate, LOW);               // switch triac off (just in case ..)
        //digitalWrite(TriacMonitor, LOW);          //
        ZeroCrossCounter++;                         // raise counters
        HalfPeriodCounter++;                        //
        if (ZeroCrossCounter > ZeroFrequency)       // after n crossings; 50 Hz = 100, 60 Hz = 120
          { ZeroCrossCounter = 1;                   // reset cross counter
            //digitalWrite(ZeroSecond, HIGH);       // switch sync output on
            //ZeroSeconds++;                        // raise seconds counter (future clock)
          }                                         //
      }                                             //
    if ((ZeroStatus == LOW) &&                      // on falling edge
        (ZeroPrev == HIGH))                         // (start current half period)
      { ZeroPrev = LOW;                             // reset vars
        ZeroFireFree = true;                        // triac released for firing
        //digitalWrite(ZeroSecond, LOW);            // sync output off
      }                                             //
  }                                                 //
                                                    //
void ReadKeyVal()                                   // 'debouncing function'...
  { KeyVal = analogRead(KeyPadIn);                  // Read Analog input; returns a value from 0 ... 1023
    if (KeyVal < 90)                                // If no key pressed,
      { KeyValCount = 0;                            // reset vars
        KeyValMax = 0;                              //
        KeyVal = 0;                                 //
        KeyStatus = false;                          //
      }                                             //
      else                                          // if a key is pressed
      { if (KeyValCount < KeyValCountMax)           // if not enough key samples
          { KeyValCount++;                          // raise sample counter
            if (KeyVal >= KeyValMax)                // if current value higher as last measured highest
              { KeyValMax = KeyVal; }               // updat highest value
            KeyVal = 0;                             // reset var
          }                                         //
          else                                      // if enough samples
          { KeyVal = KeyValMax;                     // pass keyval
            KeyValCount = 0;                        // reset vars
            KeyValMax = 0;                          //
          }                                         //
        //Serial.print("KeyVal: ");                 //
        //Serial.println(KeyVal);                   //
      }                                             //
    //Serial.print("No Key pressed");               //
  }                                                 //
                                                    //
void ReadKeyPad()                                   // Detect what key pressed...
  { if (KeyVal < 90)                                // If no key pressed, then...
      { KeyKey = DefKeyNone; }                      // set alias-value
      else                                          //
      if (KeyVal < 110)                             // If value less then...
        { // KeyKey = 1;                            // first key integer value
          KeyKey = DefKeySet; }                     // .. written here as alias...
        else                                        // if other key pressed
        if (KeyVal < 210)                           // ...
          { KeyKey = DefKeyRight; }                 //
          else                                      //
          if (KeyVal < 310)                         //
            { KeyKey = DefKeyReset; }               //
            else                                    //
            if (KeyVal < 410)                       //
              { KeyKey = DefKeyDown; }              //
              else                                  //
              if (KeyVal < 510)                     //
                { KeyKey = DefKeyOk; }              //
                else                                //
                if (KeyVal < 610)                   //
                  { KeyKey = DefKeyUp; }            //
                  else                              //
                  if (KeyVal < 710)                 //
                    { KeyKey = DefKeyEsc; }         //
                    else                            //
                    if (KeyVal < 815)               //
                      { KeyKey = DefKeyLeft; }      //
                      else                          //
                      if (KeyVal < 920)             //
                        { KeyKey = DefKeyMode; }    //
  }                                                 //
                                                    //
void KeyHandle()                                    // Executing a pressed key...
  { if (KeyStatus == true)                          // if key already has been handled
      { if (millis() > (KeyMarker + 500))           // 0,5 second before
          { KeyStatus = false; }                    // reset to handle key again
      }                                             // (= key repeater)
    if ((KeyKey > 0) && (KeyStatus == false))       // first time pressed key found
      { KeyStatus = true;                           // set control var
        KeyMarker = millis();                       // mark key handle time
        //Serial.print("Key: ");                    //
        //Serial.println(KeyKey);                   //
        if (KeyKey == DefKeyOk)                     // if OK-key pressed
          { ControlStatus = 1;                      // switch On
            if (ControlPercent == 0)                //
              { ControlStatus = 0;                  //
                //Serial.println("Power On");       //
              }                                     //
          }                                         //
        if (KeyKey == DefKeyEsc)                    // if Esc-key pressed
          { ControlStatus = 0;                      // switch Off
            //Serial.println("Power Off");          //
          }                                         //
        if (KeyKey == DefKeyUp)                     // if Up-key pressed
          { ControlPercent++;                       // raise percentage
            if (ControlPercent > 100)               // above 100%
              { ControlPercent = 100; }             // set max
```

```
           //Serial.println("Up percent");         //
         }                                        //
       if (KeyKey == DefKeyDown)                  // if Down-key pressed
         { ControlPercent--;                      // lower percentage
           if (ControlPercent < 1)                // minimal
             { ControlPercent = 0;                // value
               ControlStatus = 0;                 // switch off
             }                                    //
           //Serial.println("Down percent");       //
         }                                        //
       IntervalControl();                         // Update settings
       SyncLcd = true;                            // set var
     }                                            //
  }                                               //
                                                  //
void IntervalControl()                            // lookup half period interval
  { LookUpRow = ControlPercent;                   // Use as LookUp Table index
    TrainLength = pgm_read_word(&                 // read value's
      LookUp[LookUpRow][0]);                      //
    MidWay = pgm_read_word(&                      //
      LookUp[LookUpRow][1]);                      //
    Last = pgm_read_word(&                        //
      LookUp[LookUpRow][2]);                      //
    //Serial.print("LookUp: ");                   //
    //Serial.print(LookUpRow);                    //
    //Serial.print("%  Trainlength: ");           //
    //Serial.print(TrainLength);                  //
    //Serial.print("   Midway: ");                //
    //Serial.print(MidWay);                       //
    //Serial.print("   Last: ");                  //
    //Serial.print(Last);                         //
    //Serial.print("   Real Procent: ");          //
    //Serial.println( ((1.0 * (TrainLength * 2)) /  //
    //  (1.0 * Last) * 100) );                    //
  }                                               //
                                                  //
void LcdMessage()                                 //
  { if (SyncLcd == true)                          // whenever lcd display should be synchronized
     { SyncLcd = false;                           //
       lcd.clear();                               //
       lcd.setCursor(0,0);                        //
       lcd.print(" Power   :  ");                 //
       if (ControlStatus == 0)                    //
         { lcd.print("Off  "); }                  //
         else                                     //
         { lcd.print("On   "); }                  //
       lcd.setCursor(0,1);                        //
       lcd.print(" Setpoint: ");                  //
       if (ControlPercent < 10)                   //
         { lcd.print(" "); }                      //
       if (ControlPercent < 100)                  //
         { lcd.print(" "); }                      //
       lcd.print(ControlPercent,0);               //
       lcd.print("%      ");                      //
       //delay(1);                                //
     }                                            //
  }                                               //
                                                  //
void setup()                                      //
  { //Serial.begin(115200);                       //
    pinMode(KeyPadIn,INPUT);                      // Activate Keypad
    pinMode(ZeroCrossIn, INPUT);                  // zero cross input
    //pinMode(ZeroSecond, OUTPUT);                // one pulse per second as external trigger out
    pinMode(TriacGate,OUTPUT);                    // signal to triac
    //pinMode(TriacMonitor,OUTPUT);               //
    lcd.begin(16, 2);                             // Initialize LCD...
    lcd.clear();                                  // Reset LCD
    lcd.setCursor(0,0);                           // First line first position
    lcd.print("  IA2109DPC mk3  ");               // Show sketch name...
    lcd.setCursor(0,1);                           // Second line first position
    lcd.print("Dyn.Period.Contrl");               // Show application...
    ZeroFrequency = ZeroFrequency * 2;            // calculate default zero crossings per second
    //Serial.println("");                         //
    //Serial.println("=====IA2109DPC mk3=====");  //
    delay(2000);                                  //
  }                                               //
                                                  //
void loop()                                       //
  { ReadKeyVal();                                 // read analog input
    ReadKeyPad();                                 // read pressed key
    KeyHandle();                                  // execute key
    LcdMessage();                                 // refresh LCD
    ReadZeroCross();                              // read zero cross input
                                                  //
    if (ControlStatus == 1)                       // if switched On
      { //Serial.print(HalfPeriodCounter);         //
        //if (HalfPeriodCounter ==1)              // syncing scoop or analyzer
        //  { digitalWrite(ZeroSecond, HIGH); }   //
        //if (HalfPeriodCounter == 2)             //
        //  { digitalWrite(ZeroSecond, LOW); }    //
                                                  //
        if ((ZeroFireFree == true) &&             // if released to fire and
            (HalfPeriodOn == true))               // half period should be fired
          { digitalWrite(TriacGate,HIGH);         // triac on
            delayMicroseconds(TriacPulseLength);  // wait some microseconds
            digitalWrite(TriacGate, LOW);         // triac off
            //digitalWrite(TriacMonitor, HIGH);   //
            //Serial.print(HalfPeriodCounter);    //
            //Serial.print(" fired seq:");        //
            //Serial.print(TrainWagon);           //
            //Serial.println();                   //
            ZeroFireFree = false;                 // block firing
            HalfPeriodOn = false;                 //
```

```
          TrainWagon--;                          // lower counter
          if (TrainWagon > 0 )                   // if train not passed by yet
            { HalfPeriodOn = true; }             // set control var
        }                                        //
                                                 //
      // ---2e half powerperiod----------------  //
      if (HalfPeriodCounter == MidWay)           // at next train
        { if ( (TrainWagon == 0) &&              //
               (ZeroFireFree == true) )          //
            { TrainWagon = TrainLength;          // start down counting from...
              HalfPeriodOn = true;               // set default
            }                                    //
        }                                        //
                                                 //
      // ---reset to first half period---        //
      if (HalfPeriodCounter > Last)              // if last half period (wagon) past by
        { TrainWagon = TrainLength;              // reset default vars
          HalfPeriodCounter = 1;                 // restart at first half period
          HalfPeriodOn = true;                   //
        }                                        //
    }                                            //
}                                                //
```

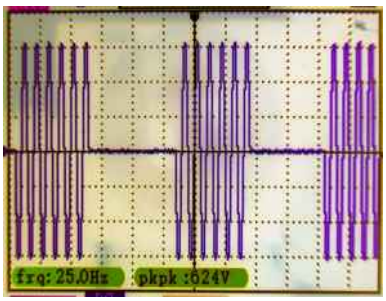
~~Tot zover lijkt de 'proof of concept' correct te werken. Nu moet in de praktijk de werkzaamheid nog getest worden.~~

Praktijktest. Helaas ontbreekt de mogelijkheid de dynamische periodensturing uit te testen op allerlei verschillende kWh-meters. Daarvoor ontbreken simpelweg de nodige meet-instrumenten (en alle in nederland gebruikte diverse typen kWh-meters).

De meterkast is voorzien van een digitale kWh-meter (ISKRA ME162). Deze meter heeft gescheiden opname- en teruglevertelwerken op 1 kWh nauwkeurig. Voor het controleren van de juiste werking is dus heeeel veeeeeeel geduld nodig.

Wel is de meter voorzien van een impulsled, met een waarde van 1000 imp/kWh, maar de led geeft niet de energierichting aan. Heel bijzonder is dat de impulsled Aan gaat wanneer er geen (energie)stroom gemeten wordt, ofwel de impulsled kan ook gebruikt worden als 'nul-op-de-meter'-indicator.

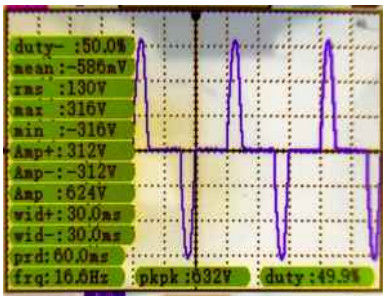
De huis-installatie is voorzien van een schakelbare PV-installatie, en een real-time energie-monitor (de IA2109EM mk3 Energy Monitor), zodat op een zonnige dag handmatig de dynamische periodenregeling getest is om 'nul op de meter' te krijgen, met een elektrische olieradiatorkachel van 1500 Watt als last.



Hiernaast de bij 48% uitsturing gebruikte periodentreintjes. Merk op dat nu een frequentie van '25 Hz' gemeten wordt voor de gehele energie-periode.

Met de op deze manier getestte nul-op-de-meter geeft de kWh-meter inderdaad nauwelijks nog led-impulsen af. En af en toe brandt de impulsled continu, als nul-op-de-meter-indicatie. De telwerken mogen dus niets meer tellen. ~~(Maar of dit correct is?)~~

Conclusies. De dynamische perioden-regeling (DPC) is een uitstekend alternatief is voor het aansturen van een boiler-warmteaccu-opslag (of een elektrische kachel, of….).

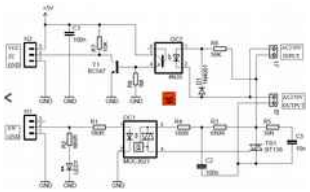


Wel zal een geschikte zwaardere triac-module moeten worden ontwikkelt.

Of, in hoeverre een dynamische periodenregeling ook inductief belast kan worden is nog even een open vraag.

Bij het meten aan een dynamische periodenregeling moet goed rekening gehouden met de meetmethode van de gebruikte instrumenten. Dit hiernaast lijkt het 50%-plaatje te zijn, maar niets is minder waar; dit zijn de weergegeven waarden bij 33% sturing.

IAAC-01 Heavy Duty Triac Module

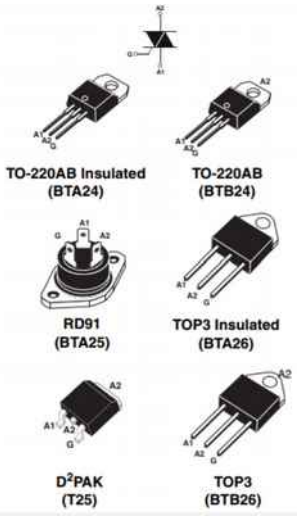


Via internet goedkoop verkrijgbaar zijn zogeheten 'MCU AC-dimmer-modules'. Goedkoop komt vaak met een prijs, zoals elektrisch te krap bemeten, onvoldoende koeling, geen of onveilige montage, geen ontstoring, en ongeschikt voor inductieve lasten. Hiernaast viermaal een ongekoelde 16 ampere triac die elkaar lekker warm stoken. Aangesloten op de 230 VAC netspanning nogal (brand)gevaarlijk.

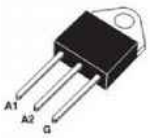
Voor zwaardere triac-modules (10...25 Ampere) wordt meestal uitgeweken naar zogeheten Solid State Relay (SSR)-modules, waarvan er nogal wat soorten en typen bestaan (en heel veel chinese namaak). Omdat SSR's en module-spaningsregelaars (SS) op elkaar lijken worden ze vaak met elkaar verward. Een spanningsregelaar gebruikt een variabele ingangsspanning (bijvoorbeeld 0-10Volt) om (fase-aansnijding-geregeld) de spaning te regelen. Een SSR vraagt om een schakelsignaal. Om een SSR fase-aansnijding- of perioden-gestuurd te kunnen schakelen, moet dit een 'non-zero-crossing' type zijn. Voor een optimaal RFI-vrije regeling moet er dan nog wel een nuldoorgangsdetector *(zero cross detector; ZCD)* aan toegevoegd worden.

Een universele heavy duty SSR, bestaat dus uit een 'triac-stuurcircuit' en een 'zero-cross-detector', waarvoor men dus zelf aan de slag moet.

Maximaal spanning. De eerste selectie bij een bidirectionele triode thyristor alternating current (=triac) is de maximaal-spanning, die afhankelijk is van de last en het schakelgebruik. Bij enkelfase 230 VAC zoals een gloeilamp is een 600 Volt-type voldoende. Bij 230 VAC inductieve lasten (motoren, trafo's, relais) moet een 800 VAC-type gekozen worden, en bij 400 VAC drie-fase minimaal de 1000 Volts typen.



Als tweede selectie kiest men de triac op basis van de triac-maximaal-stroom. Een elektrische boiler zal meestal maximaal 16 ampere vragen, maar 25 ampere komt ook voor. Vergeet daarbij de inschakelpiekstromen niet. Vooral bij fase-aansnijding kunnen deze zeer fors zijn. Bovendien heeft een triac een behoorlijk langere levensduur en veel grotere bedrijfszekerheid wanneer deze op ca 50% van de maximaalstroom werkt. Voor een 16A apparaat kiest men dus minimaal een 25 Ampere triac (40 Amp is nog beter).



Triac's zijn relatief gemakkelijk verkrijgbaar. De BT-series zijn algemene triac-typen, die ook inductieve inschakelpieken kunnen verdragen. De BTA-'s hebben een 5000 Volt **geïsoleerde behuizing *(insulated)***; wel zo veilig dus. Als 230 VAC 16A boiler-/motor-triac is de BTA41-800B een prima keuze.

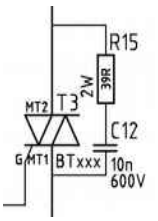
Vrije afstand. Vooral bij inductieve lasten en/of grote stromen kan over- of doorslag letterlijk een knallend probleem worden. Bij printmontage van de triac moet goed gelet worden op de vrije afstand (géén hart-op-hart!) **tussen** de netspanning-voerende leidingen; deze moet bij 230 VAC minimaal 5 mm zijn. Wanneer motoren geschakeld worden moet de vrije afstand verhoogd worden naar 10 mm. Omdat in sper de spanning tussen A1 en A2 maximaal is, moeten de triac-aansluitingen zo snel mogelijk van elkaar weg geleid worden. De 5 of 10 mm afstand geldt ook voor de afstand tussen de triac-aansluitingen en een eventueel koelprofiel of andere componenten. Bij stromen groter dan 16 Ampere kan het beste een faston- ofwel vlaksteker-type-triac worden gebruikt. Boven de 25 ampere zijn de 'moer-en-bout'-typen een betere oplossing.

De vrije afstand geldt ook voor de ruimte tussen de netspanningsleidingen en de zwakstroomcircuits. Om kruipstromen (en overslag) over de print te voorkomen wordt onder de opto-couplers een luchtspleet aangebracht.

Maximaal stroom. Uiteraard moet de bedrading ('printkroonstenen', 'terminals' en koperbanen) voldoende dik en stevig zijn om de maximaal-stroom te kunnen verwerken. Bij de veel gebruikte koperdikte van 35 µm is een printkoperbaan voor 10 ampere 3 mm breed. Bij het ontwerp en constructie moet ook rekening gehouden worden met de mechanische duw-, trek-, uitzet- en krimp-spanningen die grote stromen kunnen veroorzaken. Een S-bocht in de triac-aansluitingen is dan een oplossing. Vertinnen van de koperbanen verhoogt de maximaalstroom (ca 10%), en een expoxy printplaat is warmtebestendiger en vormvaster dan pertinax.

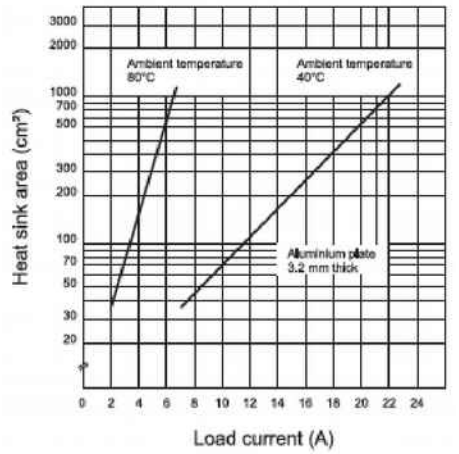
dV/dt-factor. Een gesperde triac kan zichzelf ontsteken (doorslaan) wanneer de spanningsverandering over de aansluitingen MT1 – MT2 te snel te groot wordt (de isolerende grenslagen in de triac-junctie kan zich niet snel genoeg aanpassen), zoals bij het uitschakelen van relais of motorsturing. In de datasheet wordt deze parameter vermeldt als de dV/dt-factor in volts per micro-seconde. Bij 230 VAC 50 Hz fase-aansnijding kan dat het geval zijn waardoor de triac vroegtijdig komt te overlijden. Bij periodensturing hoeven we hiermee geen probleem te verwachten, tenzij zware motoren geregeld gaan worden.

Ontstoring. Een triac in sper (met aangesloten leidingen, enz.) gedraagt zich ook als een condensator (en spoel) waarover afhankelijk van de spanning en frequentie een bepaalde lading wordt opgebouwd. Bij 230 VAC en 50 Hz zullen het eerder de harmonischen van PWM- of MCU-bloksignalen zijn die een triac kunnen ontregelen. De fabrikant heeft het nodige voorwerk al gedaan, en in de datasheet zijn vaak de waarden te vinden voor een RFI-RC-**dempingsnetwerkje** ***(snubber circuit)*** waarbij de condensator een waarde heeft van 10-100 nF en de weerstand een waarde van 10 ohm – 1K.

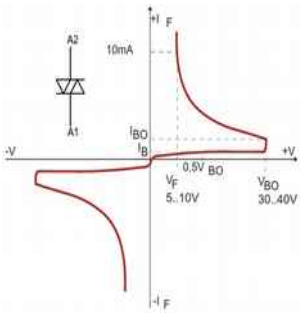


Koeling. Diac's, thyristoren en triac's zijn behoorlijk temperatuurgevoelig. Wanneer de omgevingstemperatuur te hoog is, zullen ze niet meer doven, en in doorlaat blijven. Zeker bij grote stromen (zoals 16A) moet de triac daarom voldoende gekoeld worden. De inwendige junctie-temperatuur kan maximaal ca. 125 °C zijn. Verlagen van de junctie-temperatuur voorkomt niet alleen thermische doorslag *(thermal runaway)*, maar verhoogt de bedrijfszekerheid, en verlengt de levensduur aanzienlijk. Een koel-vuistregel is dat per ampere 3 Watt aan warmte afgevoerd moet worden. Bij 16 ampere moet er dan 48 Watt aan warmte afgevoerd worden.

Zonder geforceerde ventilatie (ventilator) is een standaard koelvin *(heat sink)* meestal onvoldoende. Men kan een dure meter koelprofiel op maat maken, of uit aluminiumplaat zelf een koelprofiel maken.



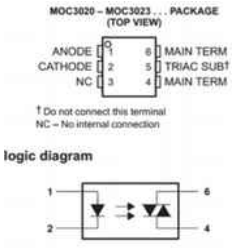
Diac. Bij een triac wordt meestal gebruik gemaakt van twee anti-parallel-geschakelde diodes in de gate-leiding als 'stroom-pulsgever' voor de ontsteking. De DIode Alternating Current (=diac) zijn twee anti-parallel geschakelde speciaal zenerdiodes in één behuizing. Het symbool geeft dit ook aan. De diac lijkt op de triac maar heeft geen gate.



Bij kleine/lage wisselspanning spert de diac. Boven een bepaalde spanning over de diac (zenerspanning fors groter dan de gebruikelijke kniespanning) zal de diac plotseling gaan geleiden. Dit is de **doorslagspanning** ***(Break Over Voltage; V_{BO})***. Zodra de diac gaat geleiden (dus net even boven de doorslagspanning) kan de triac-gate-stroom gaan lopen, en zal de triac gaan geleiden. Wanneer de wisselspanning over de diac weer terug zakt beneden de fors lagere **lawine-drempelspanning** ***(avalanche break down)*** komt, zal deze weer gaan sperren.

Voor het ontsteken/doorlaten van de diac, moet dus eerst een bepaalde (zener)spanning over de diac worden opgebouwd, voordat deze in doorlaat gaat. Een bijzondere manier van diac-ontsteken is om met een infra-rood-puls (=warmte!) de diac-drempel-spanning te verlagen. De opto-diac is géén mini-triac, maar een triggerbare diac, ontworpen om triac's aan te sturen.

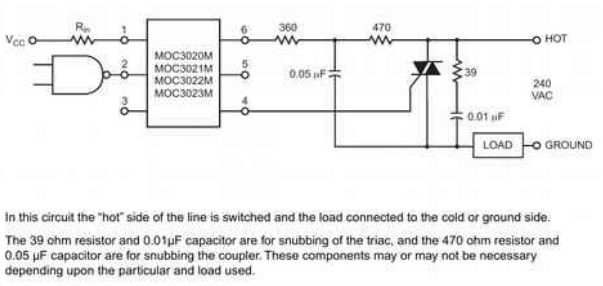
De MOC3021 is een 400 VAC opto-diac in een 6 pins DIL-behuizing, met een ingebouwde IR-LED-optocoupler, zodat een zwakspanning (3 VDC, 60 mA_{max}) voldoende is voor het ontsteken van de diac, die op zijn beurt dan de triac ontsteekt. De isolatiespanning is maximaal 7500 VAC. De MOC3021 is een universele **non-zero crossing opto-diac** voor aansturing door een MCU. Non-zero crossing wil zeggen dat de diac op elk tijdstip direct de ontsteekpuls aan de triac doorgeeft. Op deze manier kan met de triac ook fase-aansnijding gestuurd worden.

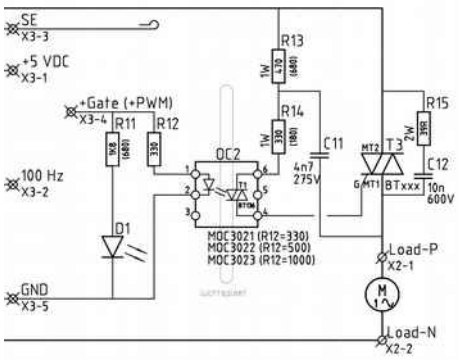


Er bestaan ook 'zero-crossing opto-diac's' (MOC304x-serie) die de ontsteekpuls direct na een nuldoorgang doorgeven. Sommige SSR's maken hier gebruik van.

MOC302x-test. Deze opto-diac kan getest worden door een ohm-meter te hangen over pin4 en pin6. Sluit een 330 ohm weerstand aan op pin1. Sluit pin2 aan op GND. Zet op de weerstand een spanning van 5VDC; de ohm-meter moet nu uitslaan in kilo-ohm-gebied (Een analoge multimeter is rustiger dan een digitale RMS-multi-meter.) Verlaag de spanning naar 3 VDC; de gemeten weerstand wijzigt mee.

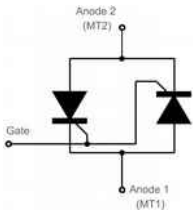
Triac circuit. Voor de circuit-ontwerpen is gebruik gemaakt van de diverse fabrikant-applicatie-voorbeelden.





Het printontwerp voorziet ook in het plaatsen van de **RC-ontstoorcircuits** (***snubber***-netwerken) voor de triac R15, C12 en voor de diac R13,C11, zodat de module ook voor licht inductieve lasten gebruikt kan worden. Het stuurcircuit kan gebruikt worden voor fase-aansnijding alsook voor halve perioden-schakelen, zodat de triac-module breed inzetbaar is.

Kwadranten. Twee antiparallel geschakelde **thyristoren** ***(Sillicon Controlled Rectifier; SCR)*** met een gemeenschapelijke gate-aansluiting in één behuizing wordt een **triac** genoemd. De triac en diac zijn ontwikkeld als wisselspanning-componenten en men zou verwachten dat het dus niets uitmaakt hoe deze aangesloten wordt. Nergens wordt immers een anode of kathode of plus of min aangegeven. Toch maakt de wijze van aansluiten heel veel uit op de correcte werking van de triac.

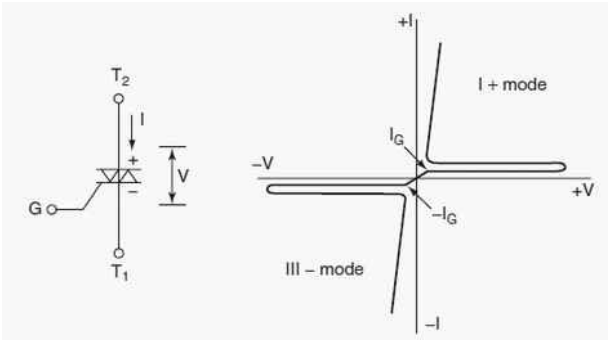


Voorbeeld; stel een wielrenner wil vanuit stilstand gaan fietsen (we willen een stroom door een thyristor laten stromen; =doorlaat). Men kan de wielrenner heel even opduwen, zodat deze op gang komt. Ofwel via de Gate worden elektronen in de sperlaag geïnjecteerd die de sperlaag van binnen af'breken. Wanneer de wielrenner tijdens de start wordt tegengeduwd, zal deze juist niet op gang kunnen komen. Ofwel, wanneer via de Gate elektronen uit de sperlaag worden getrokken, wordt de sperlaag juist nog dikker.

De combinatie van stroom**richting**en door de thyristor **èn** gate, maakt dus heel veel uit op de correcte werking. Bij een triac verdubbelt zich deze combinatie en spreekt men van schakel-**kwadranten** ***(quadrant)***.

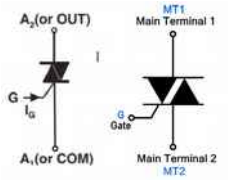
Symbol	Parameters	Quadrant		BTB16 C	BTB16 B	Unit
I_{GT} (1)	V_D = 12 V, R_L = 33 Ω	I - II - III	Max.	25	50	mA
		IV		50	100	
V_{GT}		All	Max.	1.3		V

Hiernaast het grafisch verband; in kwadrant I en III zijn de triac doorlaat- en gate-stroom-**richtingen** correct, zodat de triac zich gemakkelijk laat 'inschakelen'.



In de praktijk betekent dit dat de schema- en print-ontwerper rekening moet houden met de juiste aansluitvolgorde van de triac; het omwisselen van de aansluitingen geeft een niet-functionele schakeling.

Gate en A1 (MT1) moeten aan dezelfde zijde van het triac-symbool getekend worden. Bij schema's op het internet worden deze regelmatig onjuist gedocumenteerd; storingzoeken en printontwerpen wordt dan heel vervelend.



Nuldoorgangdetector. Naast het triac-stuurcircuit, is de module ook voorzien van een **nuldoorgangsdetector** ***(zero cross detector; ZCD)*** met een eigen uitgang. Deze uitgang kan dan via een MCU -of analoge schakeling- gebruikt worden om de triac aan te sturen. Om ongewenste faseverschuiving tussen de nuldoorgang en triac-aansturing door een trafo te voorkomen, wordt de AC netspanning gemonitord via een opto-coupler.

Wie op internet zoekt op 'zero cross detector' komt meestal uit op de 741 opamp of 4N25 opto-coupler; schema's, oplossingen, kant-en-klare module's te kust en te keur. Echter, ontwerpen met een 741 is best wel kritisch, en de 4N25 is ontwikkeld als 230 VAC één-richting (uni-directionele) **spanningdetector** met een doorslagspanning van maximaal 5000 Volt; **niet** als twee-richting (bi-directionele) **zero cross detector** voor het detecteren van nul-doorgangen.

Wanneer we de 4N25 beschouwen als een analoge transistor, verzorgt de ingangsled de basistroom. De overdrachtskarakteristiek is niet lineair. Bij een te grote basisstroom (led-stroom) wordt de foto-transistor in verzadiging gestuurd waardoor de stijg- en valtijden fors toenemen. Een héél groot nadeel bij verzadingsturing is de razendsnelle veroudering van de IR-led en fototransistor.

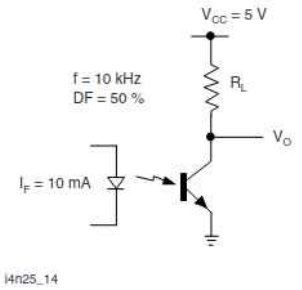


Fig. 14 - Switching Schematic

De opto-led-stroom moet dus zo klein mogelijk gehouden worden. De datasheet leert dat de optimale ledstroom ongeveer 10 mA is (levensduur is dan 10 jaar) en mag maximaal 60 mA zijn. Bij de meeste voorbeelden op het internet wordt de opto-led zwaar overstuurd, zodat de opto-transistor zich als digitale schakelaar gedraagt, dit scheelt in onderdelen en vraagt minder energie. Ja, het werkt, maar voor hoelang? Drie maand? Een half jaar?

De 4N35 is eigenlijk ontwikkeld voor scheiden DC-signaalniveau's, maar de nieuwere Vishay-versies voldoen prima als AC-alternatief. Wel zijn zowel de 4N35 stijg- als valtijd vijfmaal langer, ofwel de 4N25 is sneller.

Om een uni-directionele opto coupler als bi-directionele zero cross-detector te gebruiken moeten de nodige extra onderdelen worden toegevoegd, zoals een diodebrug. Als spanningsdetector moeten ook nog de nodige extra componenten worden toegevoegd, zoals zenerdiode, weerstanden, condensatoren, enz., zoals hiernaast weergegeven uit Application Note 02 van Vishay (83741.pdf). Het aantal onderdelen dat aan de netspanning hangt, vraagt om de nodige extra veiligheidsmarges en daarmee wordt het nog duurder.

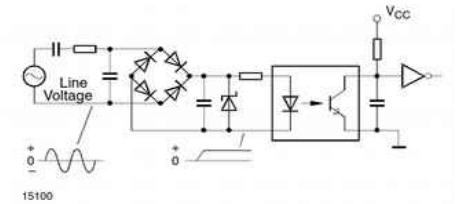


Fig. 4 - Conventional Optocoupler (One-Direction Input)
(Full-Wave Rectification by Means of Diode Bridge)

Veel eenvoudiger, goedkoper, veiliger en handiger is om dan gebruik te maken van bi-directionele AC-opto coupler typen zoals de SFH620A, PC817, TCET1600, K814P, H11AA1. Omdat nu alleen nog maar een lijn-serie-weerstand nodig is, scheelt dat aardig in onderdelen.

LET OP; het opto-led-ingangcircuit is rechtstreeks met de netspanning verbonden!!

Omdat het ingangscircuit rechtstreeks met de netspanning verbonden is, geldt ook hier de afstandseis van 5 mm. 1/8 Watt weerstanden zijn simpelweg te klein en te kwetsbaar voor eventuele spanningspieken, door- of overslag, zodat voor alle weerstanden minimaal voldoende grote (lengte!) 1 Watt typen gebruikt moet worden.

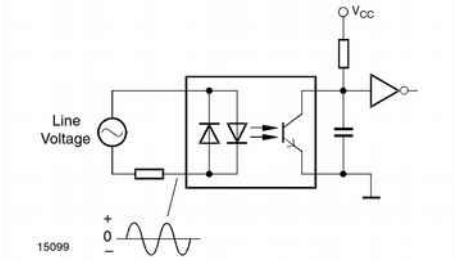
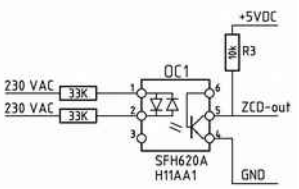


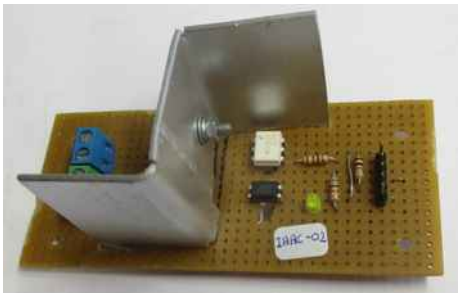
Fig. 3 - AC-Input-Compatible Optocoupler (Bi-Directional Input)

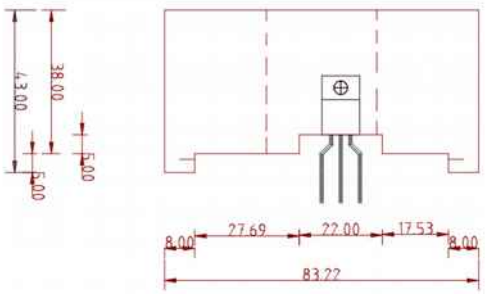
Ook voor de bi-directionele opto-coupler geldt vaak een normaal led-stroom van ca. 10 mA. Bij 230 VAC vraagt dit om een voorschakelweerstand van (230/0.01=) 23K van (230*.01=) 2,3 Watt. Bij twee seriegeschakelde 33K voorschakelweerstanden is de ledstroom (230/66000=) 3,48 mA, en zijn minimaal ((230/2) * 0.00348=) ½ Watt weerstanden nodig, zodat 1 Watt weerstanden ruimschoots zullen voldoen. De maximaal led-stroom is dan ((230 * 1,414) / 66000=) ca 5 mA, zodat de levensduur-verwachting dan ca. 20 jaar is.
(Verwachten is iets anders dan voorspellen).

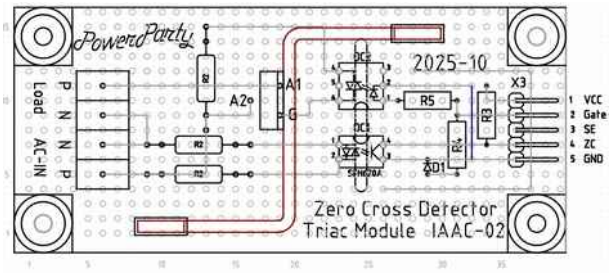


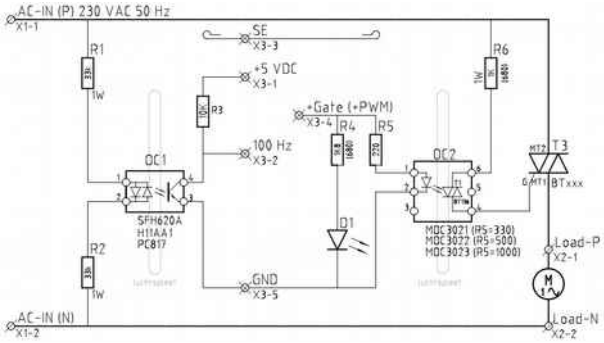
Zonder oversturing verbruikt de opto coupler-transistor iets meer energie, maar leeft ook aanmerkelijk langer. De foto-transistor-uitgang levert nu het nuldoorgangssignaal als een 100 Hz 'sinus-toppuls' van 4,2 Volt hoog en ca 0,6 ms breed. De nuldoorgang ligt exact op de top van deze puls. Aangesloten op een digitale ingang zal het signaal ingelezen worden als een digitaal signaal, en door de HoogLaag-drempelspanningen (hysterese) met een iets kortere pulsbreedte van ca 0,4 ms.

Een experimentele basis-module van ZCD & Triac voor kleine ohmse lasten tot ca 100 Watt, kan snel een eenvoudig gebouwd worden. Hieronder een voorbeeld.

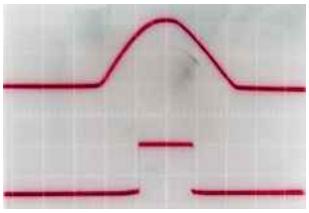








Nuldoorgang-pulsbreedte. Hoe belangrijk is de nuldoorgang-timing en -pulsbreedte?



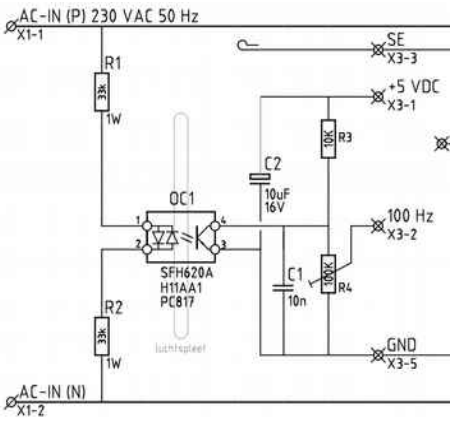
Om een triac te doven/sperren moet het gatesignaal vòòr de nuldoorgang (de top van de sinuspuls) weer op tijd nul zijn. Bij fase-aansnijding bepaalt de tijdsduur van de **nuldoorgang-voorlooptijd** de minimum regelwaarde. Het is juist de **nuldoorgang-nalooptijd** die inschakel-stroompieken en RFI veroorzaken. Bij fase-aansnijding bepaalt de nalooptijd de maximaal regelwaarde. De nalooptijd moet bij perioden-sturing zo kort mogelijk zijn.

Bij 50 Hz duurt een periode 1/50 sec = 20 ms, ofwel een halve periode duurt 10 ms. Wanneer de nuldoorgangspuls 0,6 ms duurt, kan (theoretisch) 6% van de 100% regeltijd niet gebruikt worden. Sturing is dan beperkt van 3 tot 97% (praktisch 5...95%).

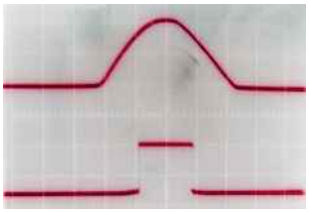
Aan de ander kant, de triac heeft zelf ook een stijg-en daal-tijd. Daar bovenop; de diac-optocoupler zal zijn eigen stijg-schakel- en val-tijden aan de triac-schakel-tijden toevoegen. Bovendien heeft ook de MCU en diens programmering de nodige (executie)tijd nodig om de nuldoorgang-schakelflanken te kunnen verwerken.

Het probleem van de nuldoorgangs-timing kan omzeild worden door gebruik te maken van een analoge ingang, en een 'detecteer maxwaarde'-routine. Het 'schakel-timing-probleem' van de 'sinus-toppuls' wordt daarmee niet opgelost, maar verplaatst naar de programmering en heeft dan invloed op de executieverloop. In veel MCU-regelbare-dimmer/ vermogens-regelingen wordt het timing-probleem softwarematig opgelost, door het tijdstip van de LaagHoog-nuldoorgang te gebruiken als 'timing-referentiepunt'. In de software moeten dan de nodige extra timing-routine's geprogrammeerd worden die de executietijd vergroten. Menig programmeur grijpt dan naar het 'interrupt-onderbreek-schakelen', of naar directe register-manipulatie om maar snel genoeg te kunnen reageren.

Vaak zal de opto-coupler-nuldoorgangspuls van ca 0,6 ms voldoende zijn. Bij een universele heavy-duty triac-module voor een periodenregeling kan een smallere nuldoorgangs-puls handig zijn. De sinus-toppuls kan met een Schmitt-trigger verbetert worden naar een nuldoorgangspuls tot ca 0,2 ms breed. Uiteraard kan daarvoor een Opamp, digitaal IC, of transistoren worden gebruikt.

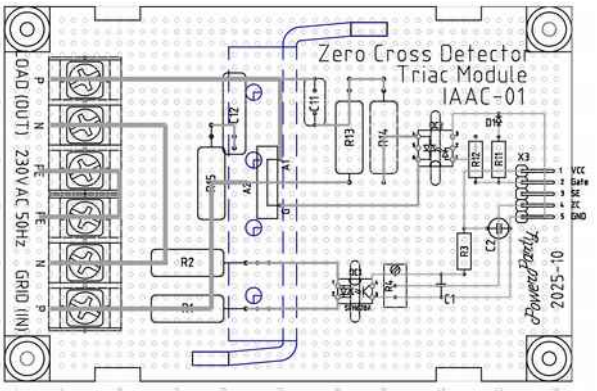


Een veel simpelere oplossing is gebruik te maken van de digitale-ingang-schakeldrempel-spanningen als hysterese. Het nuldoorgangs-signaal na de opto-coupler hoeft daarvoor alleen maar instelbaar te worden gemaakt met een instelpotmeter van 100k. Met de potmeter kan de nuldoorgangs-puls eenvoudig versmald worden van 0,4 ms naar ca 0,2 ms.



Module bouwen. Een ontwikkel-exemplaar (prototype) is gebouwd op eilanden-printplaat. Het ontwerp is bedoeld voor maximaal 25 ampere triac-toepassing en de print is onderdeel van de constructie. Extra aandacht is besteedt aan voldoende vrije afstand tussen alle netspanning-voerende componenten, koeling en natuurlijke ventilatie. Bij het kiezen van de 'netspannings-weerstanden' is rekening gehouden met eventuele extra opwarming door de triac, zodat er zelfs ruimte is voor 3 watt weerstands-typen. In de regel zullen 1 watt weerstanden al voldoen.

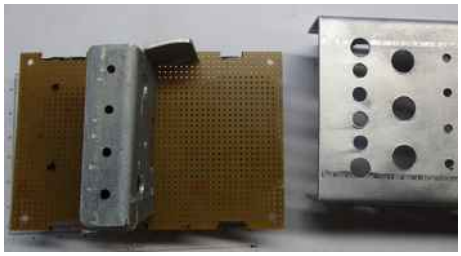
DISCLAIMER. Levensgevaarlijk. De hier gepresenteerde triac-module is slechts een studie-model en niet bedoeld voor continu of commercieel gebruik. Pas op; er wordt gebruik gemaakt van schakelingen welke direct met het lichtnet verbonden zijn. De ontwerper is niet aansprakelijke voor welke schade dan ook.



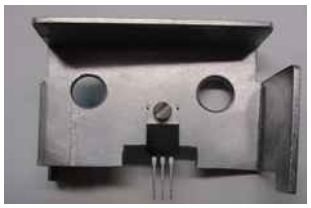
De 2 mm koelplaat bestaat uit twee delen; namelijk koelplaat1 op de print waarop de triac wordt gemonteerd.

Wanneer koelplaat1 onvoldoende is, kan een 'overkappende' koelplaat2 worden toegevoegd die tegelijkertijd de netspanningvoerende delen enigszins afschermt. Voorlopig wordt getest op maximaal 16 ampere.

De constructie begint met het nodige zaag-, vijl- en boorwerk. Maak de printplaat op maat en boor de vier hoek-montage-gaten. Figuurzaag de koelplaat-uitsparingen en zaag de luchtspleet-openingen onder de opto-couplers.



Uit een stukje 2 mm aluminiumplaat wordt koelplaat1 gezaagd, waarop de triac gemonteerd wordt. Uit een ander aluminiumplaatje wordt koelplaat2 gezaagd. Monteer de triac op koelplaat1 en controleer of de aansluitingen voldoende afstand houden ten opzichte van de koelplaat. Markeer op de koelplaat de binnen-buigrichtingen.



Begin met het omzetten in de juiste richting van de bovenkant van koelplaat1 in de bankschroef. Vervolgens kunnen de 'zijflappen' in de juiste richting omgezet worden. Na het omzetten kan met een hamer voorzichtig het 'verzet' aangebracht worden.

Nadat ook koelplaat2 in de in de bankschroef is omgezet, kan met een 'droge montage' gecontroleerd worden of alles goed past. Markeer de 3 mm gaten in de bovenkant van koelplaat1, en boor deze op maat.

Demonteer de triac en koelplaten.

Vanwege de grote stromen en hoge spanningen moeten geschikte printkroonstenen worden gebruikt. De print is daarom ontworpen voor een 600 Volt 30 Ampere print-scheidings-schot-schroef-klemmenstrook *(pcb barrier terminal)* met een steekafstand *(pitch)* van 10 mm en M3,5 kruisschroefklem. Vanwege de dubbele soldeeraansluiting is gekozen voor de driepolige MT213-10 van EuroClamp.

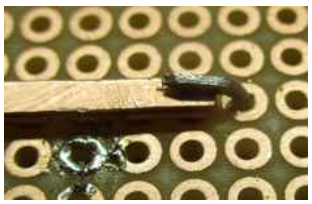
Bij een kleinere ohmse last zal een dergelijk zware aansluitklem niet nodig zijn, en kunnen deze vervangen worden door 6,35 x 0,8mm Faston printvlakstekers.

Bij een nog lichtere last kan een 5 mm printkroonsteen aangepast worden als 10 mm aansluitklem.

Begin met het monteren van de 230 VAC-weerstanden en de opto-couplers. monteer daarna de zwakspanningcircuits. Pas wanneer de zwak-spanningcircuit's gemonteerd zijn, wordt de triac op de koelplaat1 gemonteerd, en wordt deze op de print geplaatst. Verbuig de doorstekende koelplaatnokken, zodat de koelplaat stevig op de print wordt vastgezet. Nu pas kunnen de triac-aansluitingen afgewerkt worden.

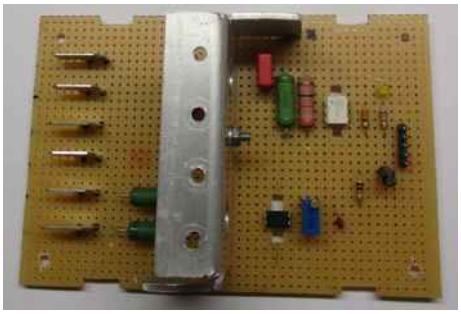
Als laatste worden de 'printbanen' naar en van de triac afgewerkt. Een print-koperlaag is absoluut ongeschikt voor het verwerken van 16/25 ampere, zodat de 230 VAC triac-printbanen 'opgedikt' moeten worden. In dit geval wordt een blank stukje 2,5 VD draad gebruikt. Voor een betere soldeer-verbinding en elektrische geleiding wordt de draad eerst met zachte 'strijkende' hamerslagen plat en glad gemaakt.

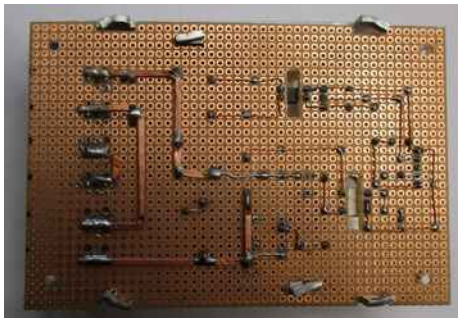
De 'opdikking' wordt vooraf op maat gebogen en 'pas' gemaakt.



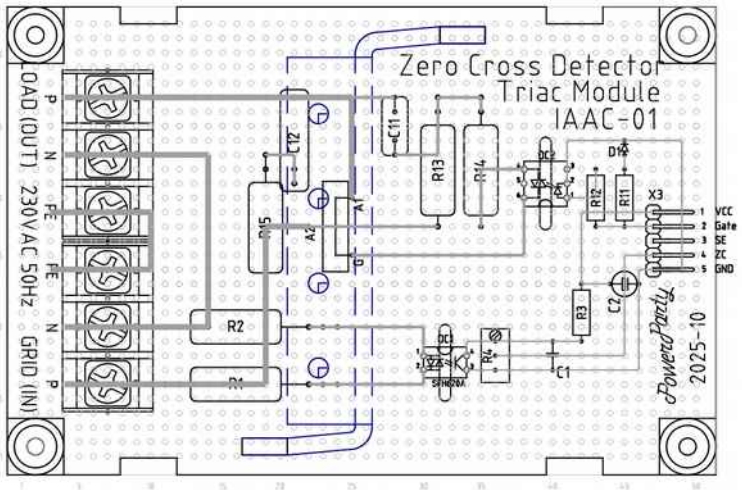
Door de triac-aansluiting over de opdikking te leggen wordt het soldeercontactvlak vergroot, zodat een betere elektrische verbinding ontstaat.

Al met al is de module meer koel- en print-plaat dan elektronica.

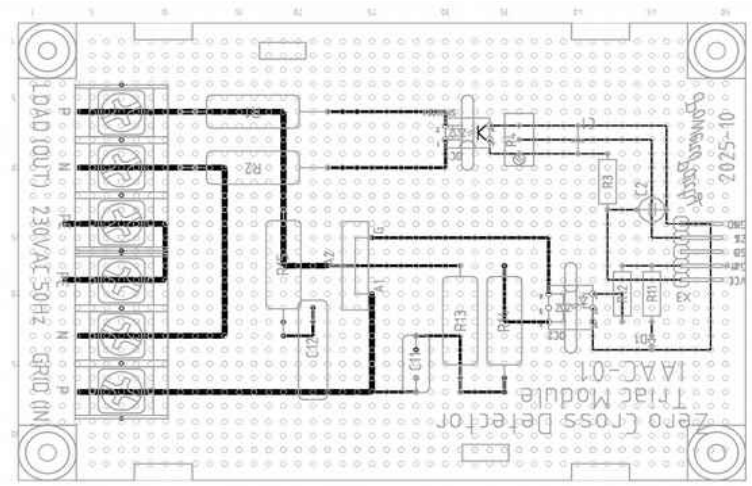




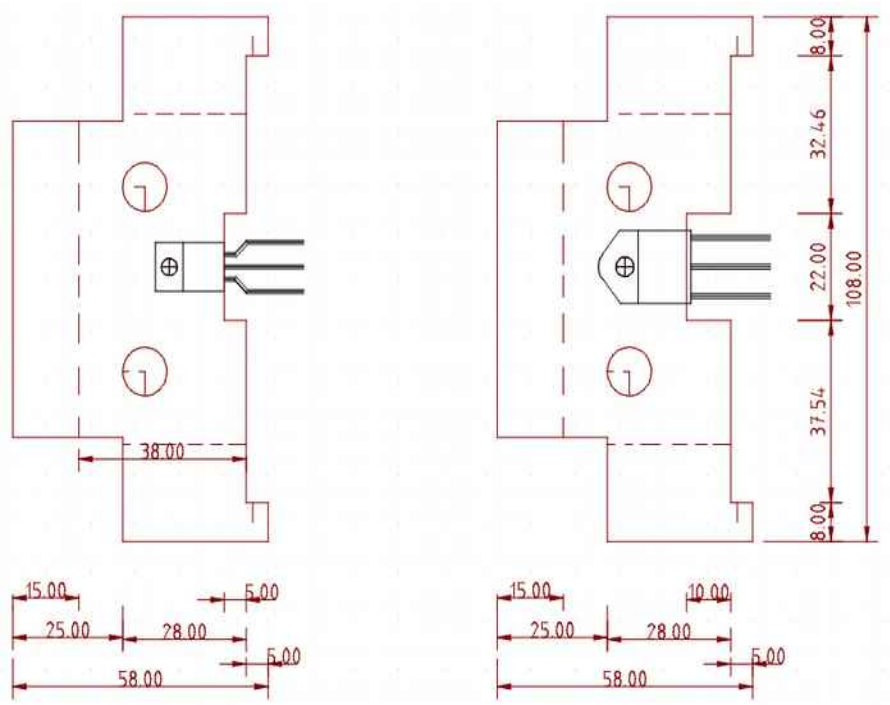
Componentzijde



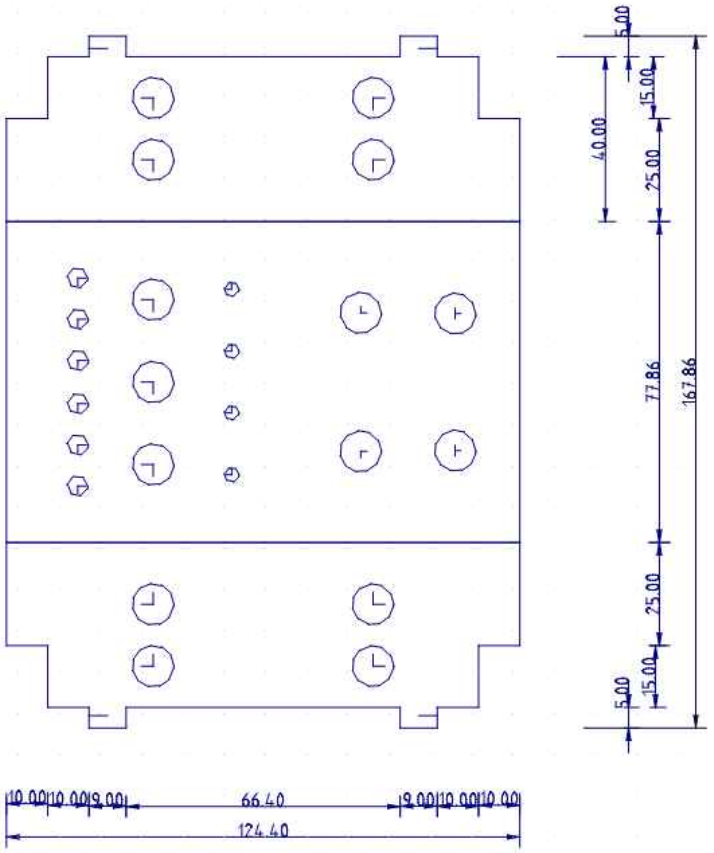
Koper(printbaan)zijde



Koelplaat1



Koelplaat2



Onderdelenlijst

~~pcb~~	~~1 stuk~~	~~Experimenteerprint, europrint (10x16 cm)~~
pcb	1 stuk	Volgens ontwerp
koel1	1 stuk	aluminium, 2 mm
koel2	1 stuk	aluminium, 2 mm
X1, X2	2 stuks	print-kroonsteen, 3 polig, MT213 EuroClamp
~~-~~	~~6 stuks~~	~~Vlaksteker 6,3-0,8mm Printconnector Haaks~~
OC1	1 stuk	SFH620A-3 opto coupler
OC2	1 stuk	MOC3021 diac opto coupler
T3	1 stuk	BTA41-800B 40A 800V
~~T3~~	~~1 stuk~~	~~BTA140-800 25A 800V~~
~~T3~~	~~1 stuk~~	~~BTA139-800 16A 800V~~
D1	1 stuk	led, 3 mm
R1, R2	2 stuks	33K, metaaloxide 3 watt
R4	1 stuk	10K, 0,25 watt
R5	1 stuk	instelpotmeter 100K, 3x2.54 steek
R11	1 stuk	1K8, 0,25 watt
R12	1 stuk	330, 0,25 watt
R13	1 stuk	330 ohm, metaaloxide 3 watt
		~~180 ohm, metaaloxide 3 watt~~
R14	1 stuk	470 ohm, metaaloxide 3 watt
		~~680 ohm, metaaloxide 3 watt~~
R15	1 stuks	39 ohm, metaaloxide 3 watt
C1	1 stuk	10 nF, keramisch
C2	1 stuk	10 uF, 16 Volt
C11	1 stuk	MKP10-4n7-1000V
C12	1 stuk	MKP10-10nF-1000V

Instellen nuldoorgang-puls. Met een osciloscoop laat zich de nuldoorgang-pulsbreedte eenvoudig met de 10K potmeter instellen.

Wie geen scoop heeft, kan de in de broncode met //>>// gemarkeerde programmaregels activeren door de voorloop // -tekens weg te halen.

Het gaat om programmaregels in de variabelen-declaratie,

```
// unsigned long ZeroTimeStart = 0;
// unsigned long ZeroTimeCum = 0;
// unsigned long ZeroTimeAvg = 0;
// unsigned long ZeroCount = 0;
```

in `ReadZeroCros()`,

```
// ------------------------------------------------ //
void ReadZeroCross()                                // Detec
  { ZeroStatus = digitalRead(ZeroCrossIn);          // read
    if ((ZeroStatus == HIGH) &&                     // on ra
        (ZeroPrev == LOW))                          // (end
      { ZeroPrev = HIGH;                            // (re)s
        ZeroFireFree = false;                       //
        // ZeroTimeStart = micros();                //>>//
        digitalWrite(TriacGate, LOW);               // switc
        //digitalWrite(TriacMonitor, LOW);          //
        ZeroCrossCounter++;                         // raise
        HalfPeriodCounter++;                        //
        if (ZeroCrossCounter > ZeroFrequency)       // after
          { ZeroCrossCounter = 1;                   // reset
            //digitalWrite(ZeroSecond, HIGH);       // switc
            //ZeroSeconds++;                        // raise
          }                                         //
      }                                             //
    if ((ZeroStatus == LOW) &&                      // on fa
        (ZeroPrev == HIGH))                         // (star
      { ZeroPrev = LOW;                             // reset
        ZeroFireFree = true;                        // triac
        // ZeroTimeCum = ZeroTimeCum +              //>>//
        //   micros() - ZeroTimeStart;              //>>//
        // ZeroTimeCount++;                         //>>//
        // if (ZeroTimeCount > 50)                  //>>//
        //  { ZeroTimeAvg = ZeroTimeCum / 50;       //>>//
        //    SyncLcd=true;                         //>>//
        //    ZeroTimeCum = 0;                      //>>//
        //    ZeroTimeCount = 0;                    //>>//
        //  }                                       //>>//
```

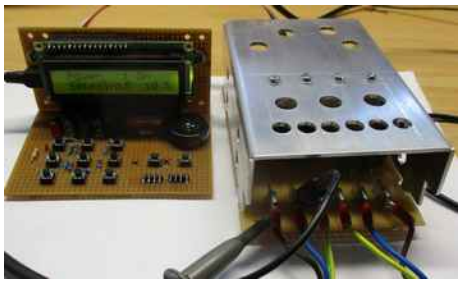
en in `LcdMessage()`.

```
void LcdMessage()                                   //
  { if (SyncLcd == true)                            // whene
      { SyncLcd = false;                            //
        lcd.clear();                                //
        lcd.setCursor(0,0);                         //
        lcd.print(" Power    : ");                  //
        if (ControlStatus == 0)                     //
          { lcd.print("Off  "); }                   //
          else                                      //
          { lcd.print("On   "); }                   //
        lcd.setCursor(0,1);                         //
        lcd.print(" Setpoint: ");                   //
        if (ControlPercent < 10)                    //
          { lcd.print(" "); }                       //
        if (ControlPercent < 100)                   //
          { lcd.print(" "); }                       //
        lcd.print(ControlPercent,0);                //
        lcd.print("%      ");                       //
        // lcd.setCursor(0,0);                      //>>//
        // lcd.print("ZCD pulse:");                 //>>//
        // lcd.print(ZeroTimeAvg);                  //>>//
        // lcd.print(" us         ");               //>>//
```

Na het uploaden van het programma wordt de huidige pulsbreedte op het lcd-display weergegeven. Om de pulsbreedte in te stellen, moet de triac-module eerst ingeschakeld worden. Het display laat de pulsbreedte zien in micro-seconden. Hoewel de meetwaarde variëert, is ze redelijk ‘constant’ ergens tussen 400 µs tot 150 µS. Versmal de pulsbreedte totdat de meetwaarde niet meer variëert. De dan ingestelde waarde (ca 125 µS) is te klein om de digitale ingangspoort te kunnen schakelen. Draai de instelpotmeter nu zover terug totdat een min-of-meer stabiele pulsbreedte gemeten wordt (ca 200 µS).

Vergeet niet in de broncode de vrijgegeven programmaregels weer uit te schakelen; deze verstoren namelijk de normale werking van het programma.

Module praktijktest. De zelfbouw module is getest met als last een 1500 Watt radiatorkachel plus een 2000 watt gloeispiraalkachel; samen 3500 Watt. Deze last is opgesteld in de buitenlucht bij ca 7 °C, zodat de last-opwarming geen enkele invloed op de triac-module heeft, en er ca 15,2 ampere wordt gestuurd, hetgeen aardig overeenkomt met een forse elektrische boiler.



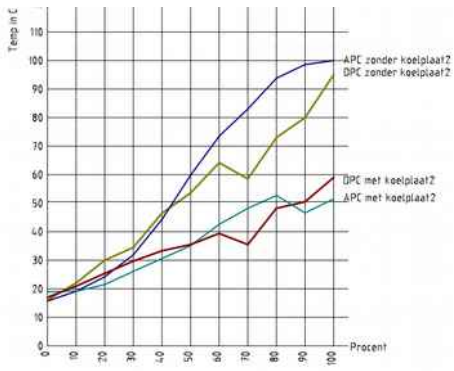
Het koelprofiel is gemaakt uit 2 mm aluminium plaat en bestaat uit twee delen. Het eerste deel (koelplaat1) dient voor montage van de triac (TO220 of TOP3) en fungeert als ‘basis-koeling’.

Wanneer deze koeling onvoldoende is -warm mag, heet worden niet-, kan daarop met vier boutjes en koelpasta een extra koelplaat2 worden gemonteerd. Uieraard moet de verwarmde lucht wel afgevoerd kunnen worden.

Hiernaast zijn de meetgegevens weergegeven van de temperatuur-ontwikkeling van fase-aansnijding (APC) met koelplaat1 en met koelplaat2 bij een last van 3500 Watt.

Dezelfde metingen zijn ook uitgevoerd bij een dynamische periodensturing (DPC).

Voor de temperatuurmetingen is gebruik gemaakt van een IR-temperatuur-meter en gemeten is op koelplaat1, 5 mm boven de triac.



Conclusies heavy duty triac module.

Opmerkelijk is de warmte-ontwikkeling in de gebruikte verlengsnoeren, stekers en contra-stekers; dit is echt wel heavy-duty. Bij een vaste installatie moet minimaal massief VD 2,5 gebruikt worden.

Ook opmerkelijk is dat de stralingswarmte van de triac de andere componenten behoorlijk mee opwarmt.

De triac-temperatuurmetingen laten een dip zien zo rond de 70%. Onbekend of dit exemplarisch is (alleen voor deze triac) of dat dit voor alle triac's geldt.

De triac-module kan gebruikt worden voor fase-aansnijding, maar ook voor (dynamische) periodensturing.

Hoewel de print ruim oogt en de module niet klein is, moet vastgesteld worden dat ze voldoet aan de eis van **heavy duty** en in staat is om 16 ampere te verwerken, mits beide koelplaten gemonteerd zijn.

Printplaat. Op basis van opgedane ervaringen en metingen met het prototype is een printplaat ontworpen. Ook bij de printplaat moeten de 230 VAC-triac-koperbanen met VD opgedikt worden.

De MCU-aansluiting kan worden gemaakt met een 2,54 mm pinheader (al of niet haaks), of een mini-printschroefklem, of met een vijfpolige JST-PH-connector, al naar gelang de gewenste connector.

Hetzelfde geldt voor de 230 VAC -aansluitingen. Men heeft de keus uit schroefklemmenstrook, faston-stekers of direct de bedrading op de print solderen.