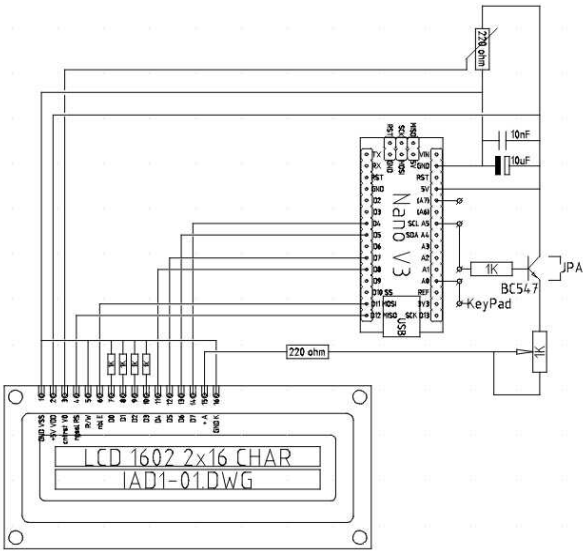
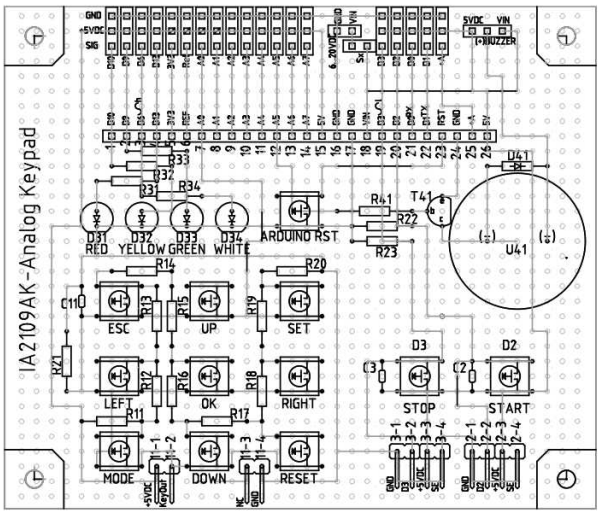
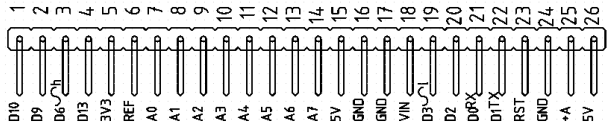
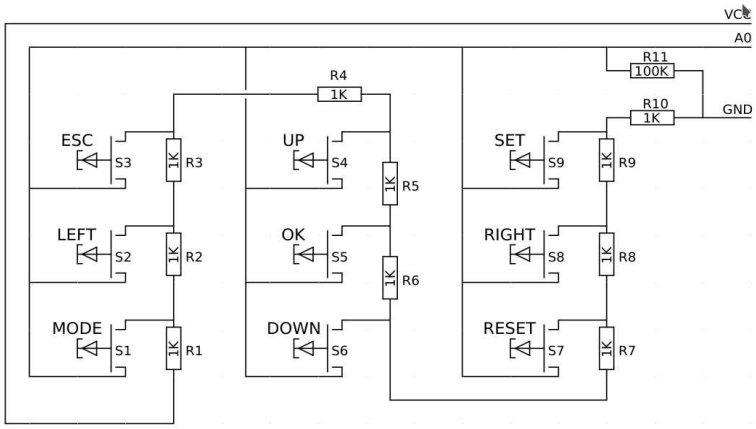
IA2109LCD-module In dit project wordt gebruik gemaakt van de IA2109 LCD-module. De opbouw en werking is in deel 1 behandeld.



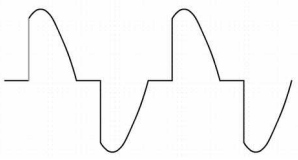
Er wordt gebruik gemaakt van de universele test- en experimenteerprint IA2109AK. De sketches geven voldoende informatie voor het direct correct aansluiten op een Nano of Uno of andere controller. Dus ook wanneer deze experimenteer-print niet gebruikt wordt.



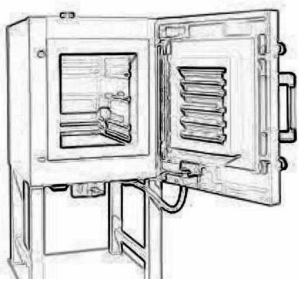




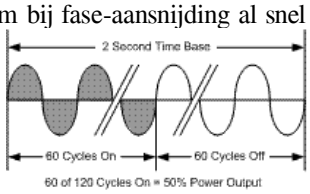
Perioden-regeling



Inleiding. Het grote nadeel van fase-aansnijding is dat deze techniek door de nogal hoge en zeer steile inschakelstromen, harmonische radio-stoorsignalen (Radio Frequency Interference; RFI) opwekken. 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. En zodra er een **zware last** geregeld moet worden, zal al snel een andere (industriële!) techniek moeten worden gebruikt.

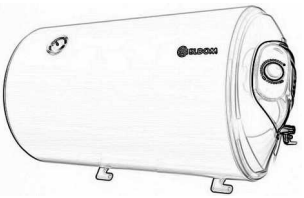


Perioden-regeling. Bij zware belastingen zoals verwarmings-elementen van boilers en keramiek-ovens in de industrie, vraagt de hoge inschakelstroom bij fase-aansnijding al snel om dure speciaal-oplossingen. De eenvoudigste manier van regelen, is Aan/Uit-regelen. Bij 10 seconden Aan gevolgd door 10 seconden 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 zal zijn.



Wanneer we afzien van relais-schakelen, kan een Aan/Uit-regeling gezien worden als een in lengte regelbare **periodentrein** *(period/cycle sequence; train)*. Wordt een triac aangestuurd met een 50% PWM-signaal van ca. 1000 Hz, is dat al voldoende om zo meerdere perioden 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. Samengevat; 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 **geschakelde PWM *(keyed PWM)***.

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



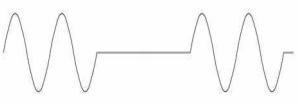
Wanneer de kWh's niet gesaldeerd kunnen worden, is de traditionele Aan/Uit-perioden-regeling een hele dure grappenmaker. Stel, neem een elektrische boiler van 4 kW, en de PV-installatie levert 2 kW aan het net. Wanneer de boiler Uit is, wordt de eigen duur opgewekte energie gratis aan de energieleverancier weggeven. Even later gaat de boiler Aan, wordt er 4 kW verbruikt, waarvan 2 kW weer ingekocht moeten 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 een triac-fase-aansnijding zodanig teruggeregeld worden, zodat alleen de eigen opgewekte zonne-stroom voor de boiler gebruikt wordt. Dit wordt dan **nul-op-de-meter** genoemd. Een nadeel is dat het opwarmen van een boiler dan langzamer gaat. Een bijzonder groot voordeel is dat de telwerken stil staan; de energie-leverancier hoeft niet meer betaald te worden voor de eigen energie.

Helaas is triac-fase-aansnijding 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 het 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 nuldoorgangs-detector *(zero cross)* wordt gebruikt om de 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', of '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. Wat is de invloed en zijn de effecten op het voedingsnet? Veroorzaakt de belasting een spannings-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 periodentrein-schakelaar? Wel of niet inschakelen op de nuldoorgang?

Hoewel SSR's op de nuldoorgang kunnen schakelen, hebben de meeste geen nuldoorgang-uitgangssignaal.

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	

Amplitude Modulatie. Een andere benadering van vermogensregeling is te stellen dat bij een spannings-amplitude-regeling het opgenomen vermogen ook geregeld kan worden. Ja, 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.

Dynamische Perioden-regeling. In de regeltechiek heeft een van nul tot honderd procent regelwaarde -vanwege het intuïtieve karakter- de voorkeur. Een kWh-meter **saldeert** en **sommeert** het energieverbruik **per seconde**. Ofwel de per seconde totaal ingaande energiestroom (in Watt) en totaal uitgaande energiestroom (in Watt) wordt eerst met elkaar verrekend (=salderen (Debet minus Credit)) en het verschil (Resultaat) wordt toegevoegd (cumulatief gesommeerd) aan het Import- of Export-telwerk. Op welke manier (chemisch, mechanisch, magnetisch of digitale processor) doet daarbij totaal niet ter zake.

Wanneer bij een PV-installatie de eigen opgewekte energie zelf verbruikt of opgeslagen moet worden, moet een perioden-regeling effectief korter zijn dan één seconde, om ongewenst doortellen van de kWh-opname- en teruglever-telwerk te voorkomen. Ofwel de regeling moet zodanig functioneren, dat de kWh-meter de energiestromen niet gaat **tellen**, maar gaat **salderen**.

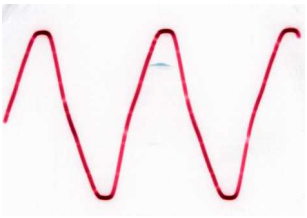
Het is niet eenvoudig om een procentuele perioden-regeling te realiseren, wanneer slechts 50 wisselstroom-perioden per seconde beschikbaar zijn. De **dynamische perioden-regeling** ***(Dynamic Period/Power Control; DPC)*** is een verbeterde variant van de seconden-schakelende periodentrein-regeling. De basis van de 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 regelcyclus 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 in het lagere regelbereik één volle gehele periode geschakeld wordt, kan het net -en de last- fors stotend belast worden. Ook de piek-inschakelstromen zullen dan aanzienlijk zijn. Wanneer de schakelperiode als twee halve perioden zo egaal mogelijk verspreid geschakeld worden, zal de belasting beter verdeeld worden en de piekinschakelstromen gehalveerd. Om RFI en EMI te voorkomen mag alleen op de nuldoorgang worden ingeschakeld.

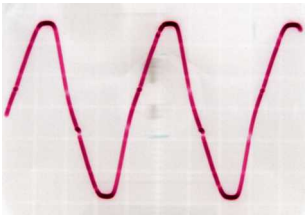
Bij een dynamische perioden-regeling wordt de belasting veel gelijkmatiger verdeeld. Een ander groot voordeel is dat bij een toenemende (regel)sturing de gelijkmatige belasting steeds beter wordt. Met discrete componten een dynamische perioden-regeling bouwen is een behoorlijk flinke klus, echter met 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.



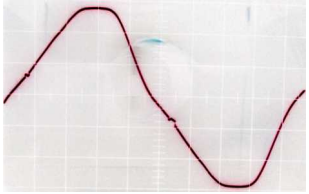
Hiernaast de **eerste nul-meting** van de schone netspanning van 230 VAC 50Hz, die tijdens deze meting enigszins uit balans is. Blijkbaar draaien er hier in de buurt nogal wat fase-aansnijding geregelde motoren (warmtepompen?) en zoveel schakelende vodingen die een versneld inzakken van de netspanning 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 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 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 vermogenperiode van 6 halve perioden.

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

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.

```
    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("");
  }
```

```
/dev/ttyUSB0

=====IA2109DPC mk1=====
1 fired
2
3
4 fired
5
6
1 fired
2
3
4 fired
5
6
1 fired
```

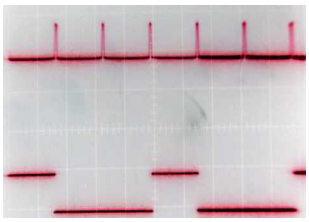
Als **vierde 'nul-meting'** wordt het daadwerkelijke schakelgedrag gecontroleerd.

```
// 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
```

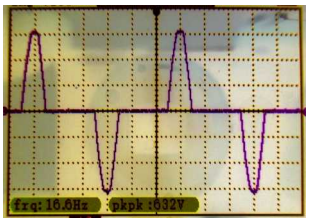
```
        ZeroFireFree = false;                       //
        digitalWrite(TriacGate, LOW);               // switch triac off (just in case ..)
        digitalWrite(TriacMonitor, LOW);            //
        //digitalWrite(ZeroMonitor, LOW);           //
        ZeroCrossCounter++;                         // 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 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, zou een kWh-meter bij gebruik van zonnestroom de energiestromen moeten salderen. ”Zou”! Of dat ook daadwerkelijk gebeurt moet in de praktijk nog getest worden (niet alle kWh-meters werken hetzelfde).

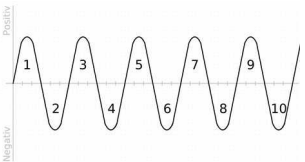
IA2109DPC DynamicPeriodControl mk2

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

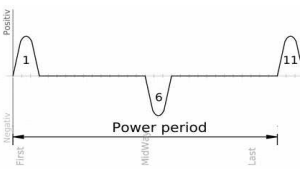
Voor de besturing wordt voorlopig gebruik gemaakt van de bij de AC-fase-aansnijding gebruikte elektronica (Arduino Nano, analog keypad, LCD, op IA2109AK-experimenteerprint) en de AC-triac-(dimmer)module met zero-cross detector.

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 ‘gefabriceerd’ wordt. ‘Gefabriceerd’ 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). Als oplossing wordt daarom gebruik gemaakt van reeksen en (golf)patronen.



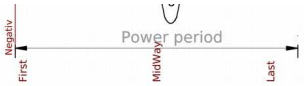
Wanneer grafisch meerdere perioden achter elkaar worden weergegeven, waarbij de eerste halve periode positief is, kunnen de opeenvolgende halve perioden genummerd worden als 1, 2, 3, 4, 5, 6, … enz. Merk op dat alle positieve halve perioden **oneven** genummerd zijn, en alle negatieve perioden **even**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

De dynamische perioden-regeling begint in het lage regelbereik met het **interval**-schakelen van **halve-perioden**, waarmee een **vermogenperiode** 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 vermogenperiode altijd een **even** aantal perioden groot is. Dit vereenvoudigt de programmering, en vermindert ook een eventuele '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.

VermogenPeriode. 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 vermogenperiodehelft is dan automatisch afwisselend 'positief en negatief'.



De vermogen-periode is in principe dan gelijk aan tweemaal de intervaltijd, en dus altijd een even getal (E, 'Last'). Ofwel, bij de 'fabricatie' wordt de 1 seconde-regeltijd compleet genegeerd. De focus ligt helemaal op het aanmaken van een vermogen-periode (E).

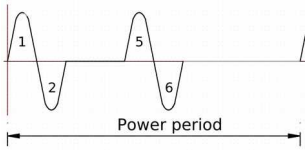
Voorwaarde bij 'één-periodehelft-schakelen' ('TrainLength=1') is dat een (oneven) positieve periodehelft gevolgd moet worden door een (even) negatieve periodehelft, en andersom, en dat na de vermogensperiodetijd (E, 'Last'), de vermogenperiode eindigt (resetten halve periodenteller).

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 vermogenperiodeduur 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).




```
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
```

De eerste vermogen-periode bestaat dan uit het treintje van de halve perioden 1 en 2, en het volgende treintje bijvoorbeeld perioden 5 en 6, of een veelvoud daar van. Door afronding van de laatste halve-periodehelft op een even nummer kan de vermogen-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 vermogenperiode (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.

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 doorgerekend, 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 voor het echte 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]);
   if ( ControlPercent > 50 )
      { TrainLength = 1;
        MidWay = 2;
        Last = 4;
      }
  }
```

Voorlopig wordt de regeling begrenst op 50% aansturing.

```
// 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
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%)
                                                //
//float ControlOutMin = 5.0;                    // minimal limit in %
//float ControlOutMax = 95.0;                   // maximal limit in %
                                                //
int HalfPeriodBase = 100;                       // default start value
int MidWay = 0;                                 // rounded even number of half periods
int HalfPeriodCounter = 1;                      // half period counter
int Last = 0;                                   // total numbers of half periods
bool HalfPeriodOn = true;                       // true  = fire period
                                                // false = do not fire
int TrainLength = 1;                            // number of half periods to fire
int TrainWagon = 1;                             // current half period counter to 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;                              //
//int LookUpCol = 0;                            //
//int LookUp[m][n] =                            // define a 2D array; m-rows, n-columns
const int LookUp[51][3] PROGMEM =               // define a 2D array; 51 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, 20, 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
    { 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
  };                                                //
                                                    //
// ------------------------------------------------ //
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();                        //
        SyncLcd = true;                           //
      }                                           //
  }                                               //
                                                  //
void IntervalControl()                            // fabricate half period interval
  { LookUpRow = ControlPercent;                   //
    TrainLength = pgm_read_word(&                 //
      LookUp[LookUpRow][0]);                      //
    MidWay = pgm_read_word(&                      //
      LookUp[LookUpRow][1]);                      //
    Last = pgm_read_word(&                        //
      LookUp[LookUpRow][2]);                      //
   if ( ControlPercent > 50 )                     //
      { TrainLength = 1;                          //
        MidWay = 2;                               //
        Last = 4;                                 //
      }                                           //
    Serial.print("LookUp: ");                     // echo...
    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 mk2  ");             // 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 mk2=====");  //
    delay(2000);                                //
  }                                             //
                                                //
void loop()                                     //
  { ReadKeyVal();                               // read analog input
    ReadKeyPad();                               // read pressed key
    KeyHandle();                                // execute key
    LcdMessage();                               // update LCD
    ReadZeroCross();                            // read zero cross input
                                                //
    if (ControlStatus == 1)                     // if switched On
      { //Serial.print(HalfPeriodCounter);       //
        if (HalfPeriodCounter ==1)              //
          { 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 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 halve perioden-regeling 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. Uiteindelijk lijkt de gekozen oplossing het meest op de werking van een schuiftrompet.

De belangrijkste conclusie is dat de dynamische periodenbesturing precies dat doet waarvoor ze ontworpen is, namelijk het 'salderen' van de energiestromen. Maar …, als gevolg daarvan 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 zal salderen, maar in hoeverre een digitale kWh-meter dat ook gaat doen, zal de praktijk moeten leren.

IA2109DPC DynamicPeriodControl mk3 In versie mk3 is de lookup table uitgebreid naar 100%, is de seriele 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
```

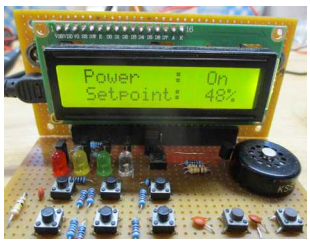
```
           //Serial.println("Up percent");         //
         }                                        //
       if (KeyKey == DefKeyDown)                  // if Down-key pressed
         { ControlPercent--;                      // lower percentage
           if (ControlPercent < 1)                // 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.

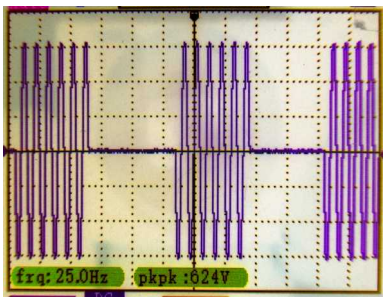
Praktijktest. Helaas ontbreekt de mogelijkheid de dynamische periodenregeling uit te testen op allerlei verschillende kWh-meters. Daarvoor ontbreken simpelweg de nodige meet-instrumenten (en alle in nederland gebruikte diverse typen kWh-meters).

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 wordt ook gebruikt 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 als last.

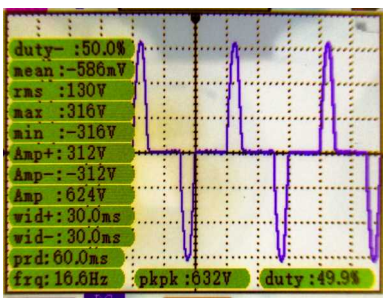


Hiernaast de bij 48% uitsturing gebruikte periodentreintjes. Merk op dat nu een frequentie van ‘25 Hz’ gemeten wordt voor de gehele ‘vermogen-periode’.

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. Het lijkt er op dat de dynamische perioden-regeling 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 gemeten waarden bij 33% sturing.