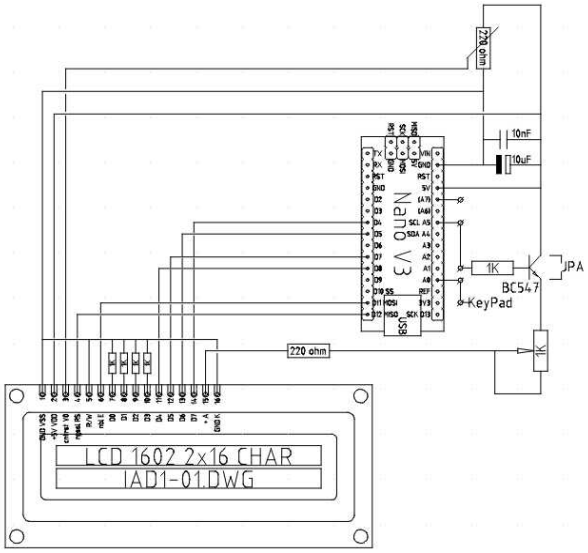
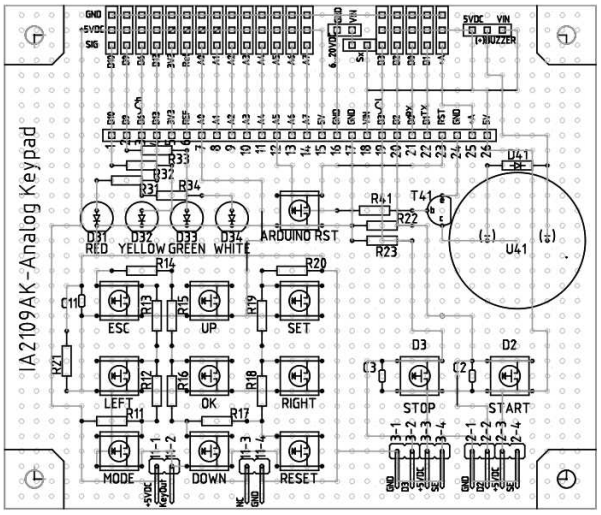
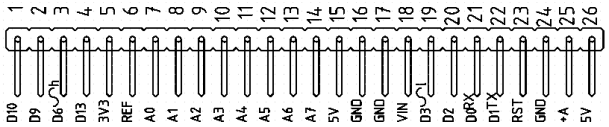
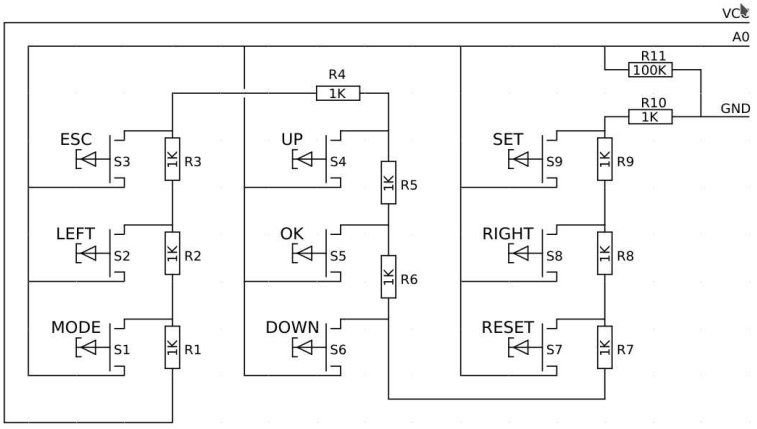
IA2109LCD-module In dit project wordt gebruik gemaakt van de IA2109 LCD-module. De opbouw en werking is in deel 1 behandeld.



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.

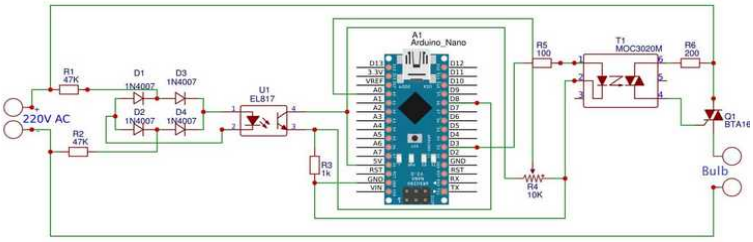






IA2109ACP, AC Powercontrol in procent

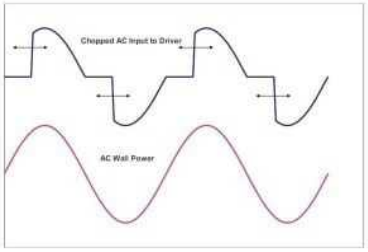
Inleiding. Wie 230 VAC wisselspanning wil gaan regelen, kan op internet onder ‘arduino 230 VAC control’ vele voorbeelden vinden.



Voor het regelen van een wissel-spanning en -stroom wordt meestal gebruik gemaakt van fase-aansnijding, met een triac-schakeling. Op het internet zijn onder zoekterm ‘arduino triac’ de nodige voorbeelden en kant-en-klare module’s te vinden.

De meeste regelingen gebruiken de RBDdimmer.h -bibliotheek. De sketch zelf is dan een kwestie van Copy&Paste, en voila! We hebben als na-aapje een dimmer gemaakt. Pas lastig wordt het wanneer we van de voorbeelden de bediening willen veranderen, vooral omdat RBDdimmer.h gebruik maakt van interrupts en delayMicroseconds().

Een groot nadeel van **positieve fase-aansnijding *(leading edge phase cutting)*** is het ongewenst opwekken van R.F.-stoorpulsen, die om de nodige extra ontstoor-onderdelen vragen. Zonder extra voorzieningen is fase-aansnijding eigenlijk alleen geschikt voor ‘low power’ ohmse lasten, want vanwege de forse inschakelstromen krijgt de triac het flink voor z’n kiezen; een kapotte triac is niets opzienbarends.



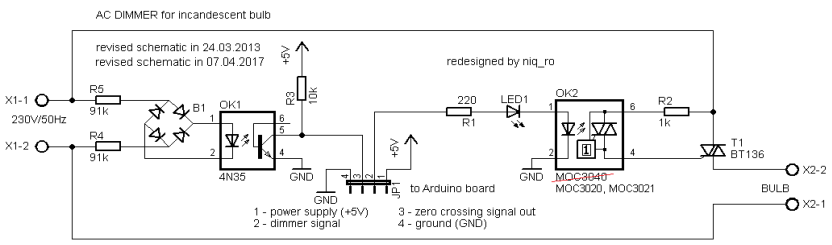
Als laatste, maar niet als minste, het fase-aansnijding-regelgedrag is beslist niet altijd optimaal te noemen, vooral bij het naderen van de grenswaarden van 0 en 100% wordt het elektronisch regelen een lastige aangelegenheid.

Bij de stijgende energieprijzen, toenemend aantal solar-installaties en afbouw salderingsregeling, wordt het interessant om bij een overschot aan opgewekte energie, deze te gebruiken voor het extra opwarmen van water in een eenvoudige standaard elektrische boiler Een standaard 230 VAC 16A elektrische boiler vraagt om een 3680 VA-regeling, en bij 25A om zelf een 5750 VA-regeling. Dat zijn forse vermogens die om speciaal oplossingen vragen. Omdat er een wezenlijk verschillen bestaan tussen de ‘lichte’ fase-aansnijding- en een ‘zware’ perioden-regeling, verdiepen we ons eerst in de fase-aansnijding.

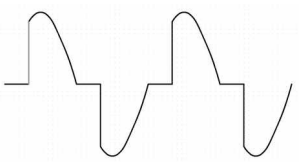
Benodige elektronica. Uiteraard moet de triac-stuur-schakeling voldoende ruim bemeten voor de bedoelde toepassing. De BT-series zijn algemene triac-typen, waarbij de BTA- en BTB-series-xW ook inductieve lasten kunnen schakelen. De BTA16 kan inductief 16 ampere schakelen bij 230 VAC.

TO-220AB Insulated (BTA16)

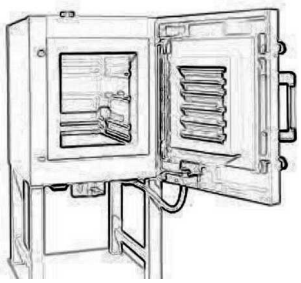
In dit voorbeeld is het helemaal gemakkelijk, want er wordt gebruik gemaakt van een kant-en-klare AC-dimmer-module met de BTA16. Wel vervelend dat de print maximaal 5 Ampere mag schakelen.



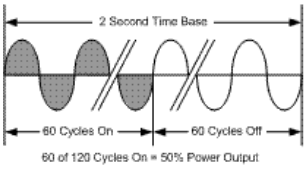
Dynamische Perioden-regeling (DPC)



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



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



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

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

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

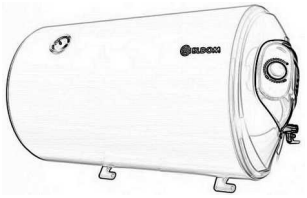
De kWh-meter heeft als **meetsensor** een **Watt-meter**, waarbij het netto aantal getelde **energiepakketjes (joules) per seconde** wordt gemeten in Watt.

De Watt-meter als meet-sensor moet dus de **per seconde** totaal ingaande energiestroom (import in Joules) en totaal uitgaande energiestroom (export in Joules) bij elkaar optellen (ofwel met elkaar verrekenen), en het resultaat dan toevoegen aan het import- of export-telwerk.

Bedenk wel, de meetmethode van energie-meting **per seconde** ligt vast in de wet; daar kan geen meter-fabrikant, netbeheerder of energieleverancier ook maar iets aan veranderen. Op welke manier (chemisch, mechanisch, magnetisch of digitale processor) de kWh-meter dat doet, doet daarbij totaal niet ter zake.

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

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



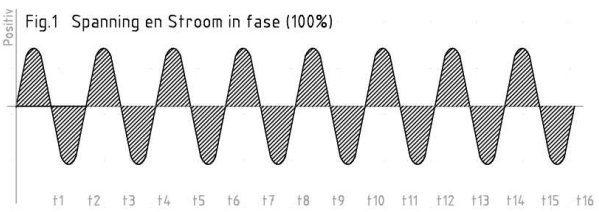
Wanneer de zonne-energie-kWh's niet meer gesaldeerd kunnen worden, is de traditionele perioden-regeling een hele dure grappenmaker. Stel, neem een elektrische boiler van 4 kW, en de PV-installatie levert 2 kW. Wanneer de boiler Uit is, wordt de eigen duur opgewekte energie gratis (-of met boete-) aan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

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

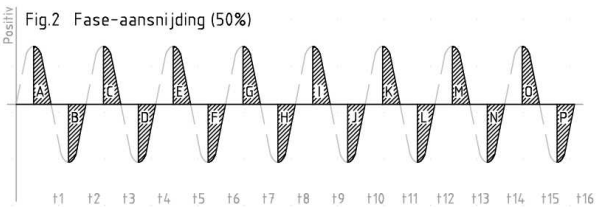
Helaas is triac-fase-aansnijding -vooral vanwege de piek-inschakelstromen- niet zo geschikt voor het aansturen van zware lasten, zoals een boiler, olieradiatorkachel, infra-rood paneel, of …

Introductie dynamische periodenregeling (DPC).

De totaal afwijkende manier van opbouw en sturen met een dynamische periodensturing (DPC), is niet even gemakkelijk te doorgronden. De wiskundige onderbouwing en verhandeling laten we voor wat het is. Hier wordt de grafische weergave gebruikt wordt om het principe en werking van de dynamische periodenregeling toe te lichten.

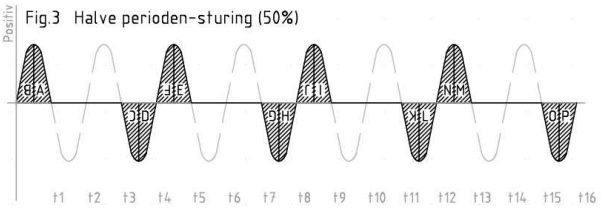


Figuur 1 is de 100% weergave van een wisselspanning en stroom in fase. Het vermogen (energie) is dan som van de momentele spanning maal de momentele stroom, ofwel P = U x I. Bij een weerstandslast beweegt de stroom netjes mee met de wisselspanning.

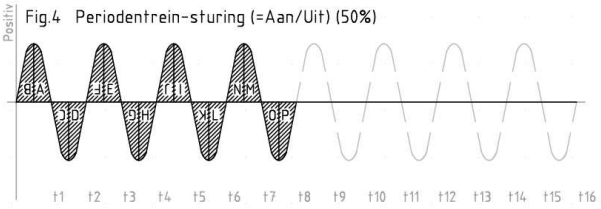


In figuur 2 wordt 50% fase-aansnijding gebruikt voor een energieregeling. Het totaal aan gearceerde vlakken is dan de grafische weergave van de energie-doorgifte naar de last. Door de periodieke inschakeling bij fase-aansnijding ontstaan er extra grote inschakelstromen en dus ook harmonische stoorsignalen.

In figuur 3 is een halve-perioden-sturing gemaakt, door om-en-om de fase-aansnijding van figuur 2 te roteren. De energie-doorgifte van 50% is nog steeds exact hetzelfde aan die van de 50% fase-aansnijding van figuur 2. Een groot verschil is wel dat er nu geen grote inschakelpiekstromen en harmonische stoorsignalen worden opgewekt. Wanneer met fase-aansnijding nul-op-de-meter geregeld kan worden, kan dat dus ook met de halve-perioden-sturing van figuur 3.

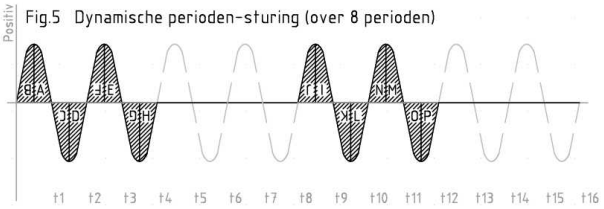


In figuur 4 worden alle halve perioden aan elkaar gerijgd naar een periodentrein. Binnen de tijdsduur van deze voorbeelden (van 16 halve perioden), maakt voor de energie-doorgifte de positie van de halve perioden rekenkundig totaal niets uit. Deze periodentrein zal evenveel energie doorgeven dan die van de halve perioden alsook van de 50% fase-aansnijding.



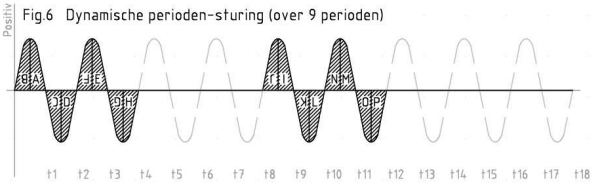
Maar, periodentrein-sturing is feite een Aan/Uit-sturing, en zolang de Aan- en Uit-tijd **samen** korter is dan 1 seconde, zal de **Watt-meter** van de kWh-meter netjes naar rato salderen. Bij een traditionele periodenregeling van bijvoorbeeld 10 seconden Aan en 10 seconden Uit (=50%) zal de wattmeter uiteraard nog steeds salderen, maar worden ook de kWh-telwerken gebruikt zodat de energieleverancier een factuur kan opmaken. Voor hergebruik van de gratis (-of met boete-) weggegeven duur opgewekte eigen energie moet dan extra betaald worden. Kortom, bij periodentreinen gelijk of langer dan 1 seconde is nul-op-de-meter regelen niet mogelijk; er moet dan betaald worden voor het gebruik van het energienet als accu.

Bij de dynamische periodensturing (figuur 5) vormen twee periodentreintjes **èn** hun wachttijden **samen** één **'energie-periode'**. Van de 16 halve-perioden die (deze) energie-periode duurt, worden 8 halve perioden gebruikt om energie door te laten, zodat de energie-doorgifte ((8/16) x 100=) 50% is. Bij 50 Hz duurt de energie-periode dan (16/100=) 0,16 seconde, zodat de kWh-meter-**wattmeter** de energiestroom keurig zal salderen. Met een dynamische periodensturing kan er wèl nul-op-de-meter geregeld worden.

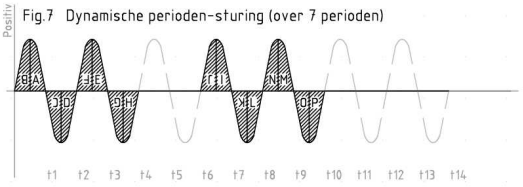


Zo gemakkelijk een fase-aansnijding zich laat aansturen, zo lastig is dat juist bij een dynamische periodensturing. Deze is immers ***veranderlijk*** in **periodentreinlengte**, **èn** een veranderlijke **wachtijdlengte na de eerste periodentrein**, **èn** een veranderlijke **wachttijdlengte na de tweede periodentrein**. Er zijn dus drie 'regelknoppen' voor de opbouw van één energie-periode. Omdat al deze factoren veranderlijk zijn, heet de sturing dan ook **dynamisch**.

In figuur 6 is de energieperiode verlengd van 16 naar 18 halve perioden. De energieperiode duurt dan (18/100=) 0,18 seconde. De energie-doorgifte is dan ((8/18) x 100=) 44,4 %.



In figuur 7 is de energieperiode verkort naar 14 halve perioden. De energieperiode duurt dan (14/100=) 0,14 seconde. De energie-doorgifte is dan ((8/14) x 100=) 57,1 %.



Zoals bij een vingerzetbank verschillende vinger-breedtes gerijgd kunnen worden naar een gewenste binnen-zetmaat, zo gebruikt de dynamische periodenregeling het rijgen van de eerste periodentrein, eerste wachttijd, tweede periodentrein en tweede wachttijd **samen**, om een energie-periode samen te stellen, die korter duurt dan 1 seconde, waardoor er met DPC nul-op-de-meter geregeld kan worden.

Stuur- en Regel-gedrag. *Een nauwelijks onderkend verschijnsel van fase-aansnijding, is de invloed van de niet-lineaire stuur-karakteristiek. Processen wil men vanwege het intuïtieve karakter graag procentueel regelen van 0 ...100% bij een setpoint vanaf 50%. Bij fase-aansnijding is de energie-stapgrootte van 0 naar 2% vele vele vele malen kleiner dan de energie-stapgrootte van 49 naar 51 %. Grafisch heeft de stuurlijn een S-vorm, die in het middenbereik het steiltst (grootst) is.*

Wordt in een proportioneel regelsysteem fase-aansnijding gebruikt als 'actuator', dan zal een correctie in het midden van het regelbereik door de fase-aansnijding-stuurlijn extra versterkt worden. Met wat pech ontstaat er dan een onrustige regeling, die in het ergste geval zelfs kan gaan pendelen. Aan de andere kant, wanneer men rekening houdt met deze S-vormige stuurlijn, en de procesregeling wat trager kan laten reageren, kan het juist een voordeel zijn voor een veel stabieler regelproces, en kan vastlopen (clipping) juist voorkomen worden.

De dynamische periodensturing is lineair. Een verstorende invloed van de stuurlijn op de regelaar is daarom niet te verwachten.

Conclusies;
– Met fase-aansnijding kan eenvoudig een vermogensregeling worden gemaakt.
– Voor zware belastingen is fase-aansnijding niet zo geschikt.
– Wanneer alle vermogensregelingen fase-aansnijding gaan gebruiken,
(wat bij toenemende elektrificering zeer waarschijnlijk is),
wordt het net te sterk eenzijdig belast en zal in de toekomst de netspanning ontregeld raken (wat ze soms al is).

– Periodentrein-sturing veroorzaakt minder hoge inschakelstromen en stoorsignalen.
– Voor zware belastingen zeer geschikt, maar voor verlichting totaal ongeschikt.
– Bij periodentreinen langer of gelijk aan 1 seconde is nul-op-de-meter niet mogelijk

– De dynamische periodensturing (DPC) combineert de periodentrein-schakeltechniek,
en 'snelheid van de de fase-aannijding', om een binnen één seconde een last te sturen.
– Met DPC kan daardoor ook bij zware lasten nul-op-de-meter gerealiseerd worden.

De theorie van de dynamische periodenregeling *(Dynamic Period Control)* is behoorlijk eenvoudig.

De praktische uitwerking naar een bruikbare schakeling is heel iets anders!

De volgende alinea's is een compleet verslag van het DPC-ontwikkelproces.

IA2109DPC DynamicPeriodControl mk3

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

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

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

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

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

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

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

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

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

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

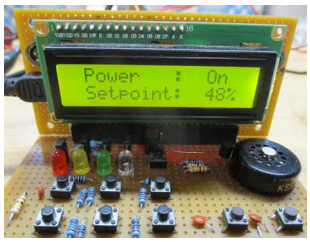
```
          ZeroFireFree = false;                 // block firing
          HalfPeriodOn = false;                 //
          TrainWagon--;                         // lower counter
          if (TrainWagon > 0 )                  // if train not passed by yet
            { HalfPeriodOn = true; }  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

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

Praktijktest.

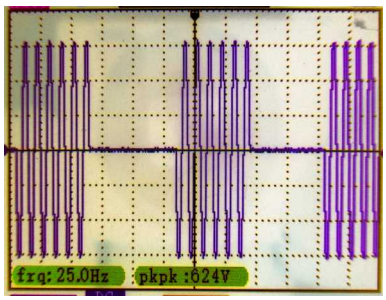
Helaas ontbreekt de mogelijkheid de dynamische periodensturing uit te testen op allerlei verschillende kWh-meters. Daarvoor ontbreken simpelweg de nodige meet-instrumenten (en alle in nederland gebruikte diverse typen kWh-meters).

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



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

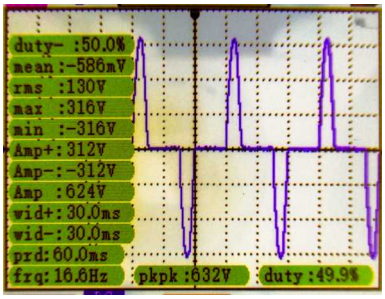
De huis-installatie is voorzien van een schakelbare PV-installatie, en een real-time energie-monitor (de IA2109EM mk3 Energy Monitor), zodat op een zonnige dag handmatig de dynamische periodenregeling getest is om 'nul op de meter' te krijgen, met een elektrische olieradiatorkachel van 1500 Watt als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



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

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

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



Voor grote vermogens moet een geschikte zwaardere triac-module worden gebruikt.

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

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