

Titel: HLG dimmer aansturing met Arduino  
Auteur: Aad Slingerland  
Datum: mei 2018  
Versie : 1.1

## Inleiding

Het doel van dit project is het maken van een automatische regelaar voor een HLG dimmer. Deze regelaar moet op een bepaald tijdstip een zonsopgang nabootsen en op een later tijdstip een zonsondergang. De door Vincent gebruikte dimmer heeft drie manieren om de dim functie aan te sturen. De voor dit projectje gekozen manier is het gebruiken van een 10 volt PWM signaal.

Op advies van Radio Piet (Arnhem) heb ik in eerste instantie een Arduino Uno van het merk Velleman aangeschaft. Later is gebleken dat door de drie extra elektronische componenten deze Arduino net iets te klein bleek te zijn. Daarom heb ik later een Arduino Mega aangeschaft. Ook van het merk Velleman.

De wens van Vincent is om drie verschillende programma's te hebben. Programma één om de lampen constant aan te zetten. Programma twee met een zonsopkomst om 4:00 uur 's middags en een zonsondergang om 10:00 uur 's morgens (in totaal 16:00 uur licht). Programma drie met een zonsopkomst om 4:00 uur 's middags en een zonsondergang om 4:00 uur 's morgens (in totaal 12:00 uur licht). Daar heb ik zelf Programma nul aan toegevoegd waarbij de verlichting gewoon uit is.

## Hardware

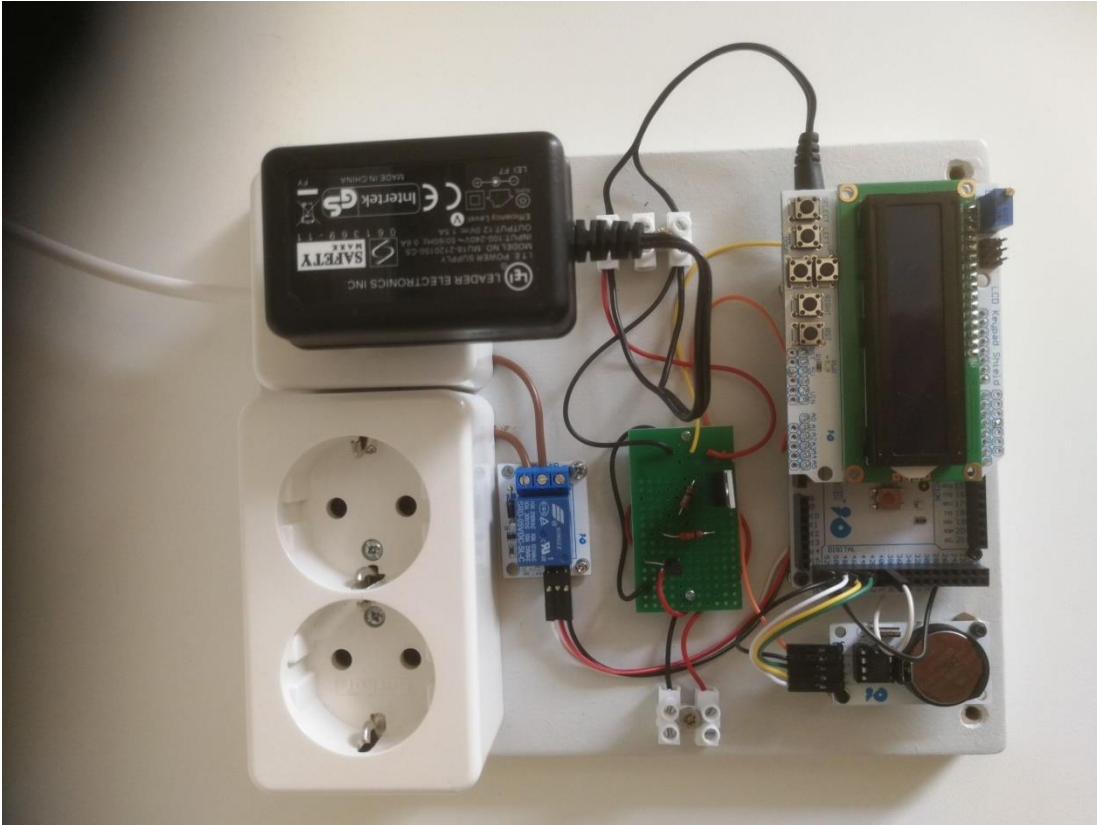
De gebruikte dimmer voor de LED verlichting is van de fabrikant Mean Well en het typenummer is HLG-185H-C1440B. Het gebruikte Arduino board is een VMA101. Deze is voorzien van een LCD keypad van het type VMA203. Verder is er een klokmodule nodig eveneens van de fabrikant Velleman. Het type is VMA301. Om de 230 volt AC voeding te kunnen schakelen wordt gebruik gemaakt van een relais van het type VMA405.

Verder bleek het nodig om het vijf volt PWM signaal (vande Arduino) op te waarderen naar 10 volt. Eén en ander volgens de specificaties van Mean Well. Op advies van Hureka (Zevenaar) heb ik hiervoor, met een handvol onderdeeljes, een PWM versterker gemaakt.

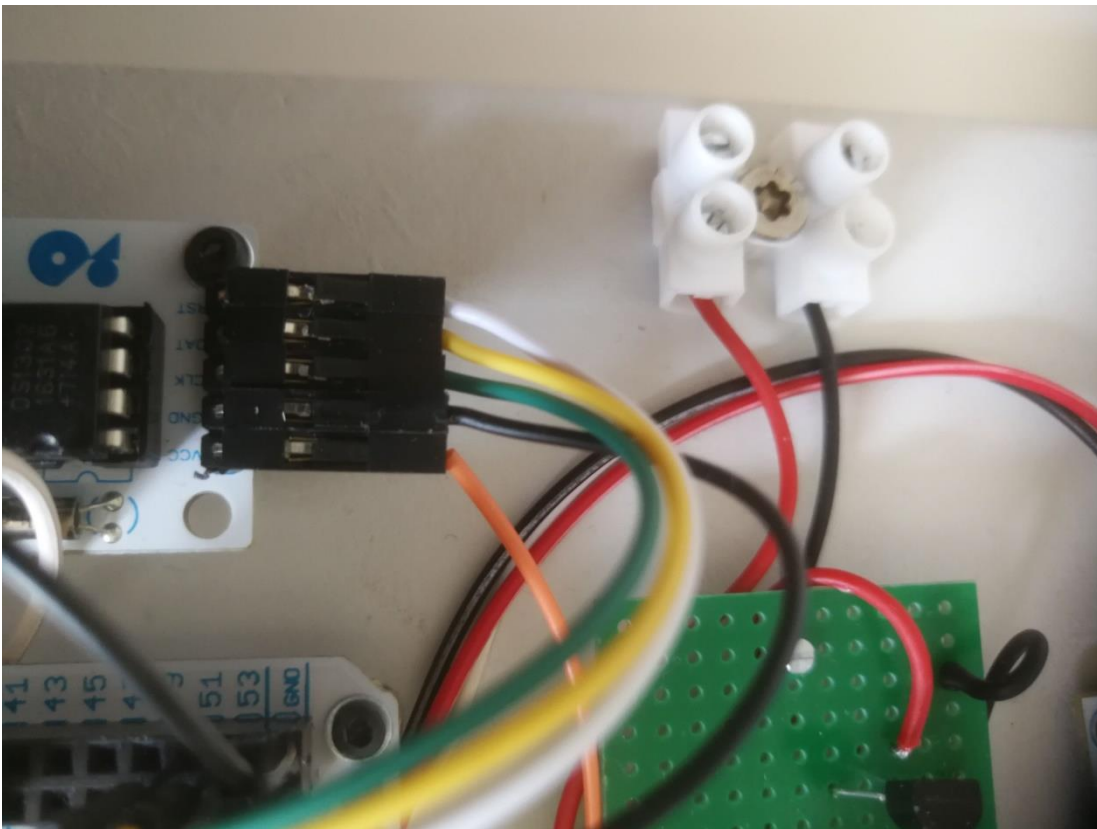
Voor het uiteindelijke resultaat is verder gebruik gemaakt van een transformator voor 12 volt gelijkstroom en twee wandcontactdozen voor opbouw. En een plankje MDF van 18 bij 22 centimeter.

Screenshots met de PIN aansluitingen:

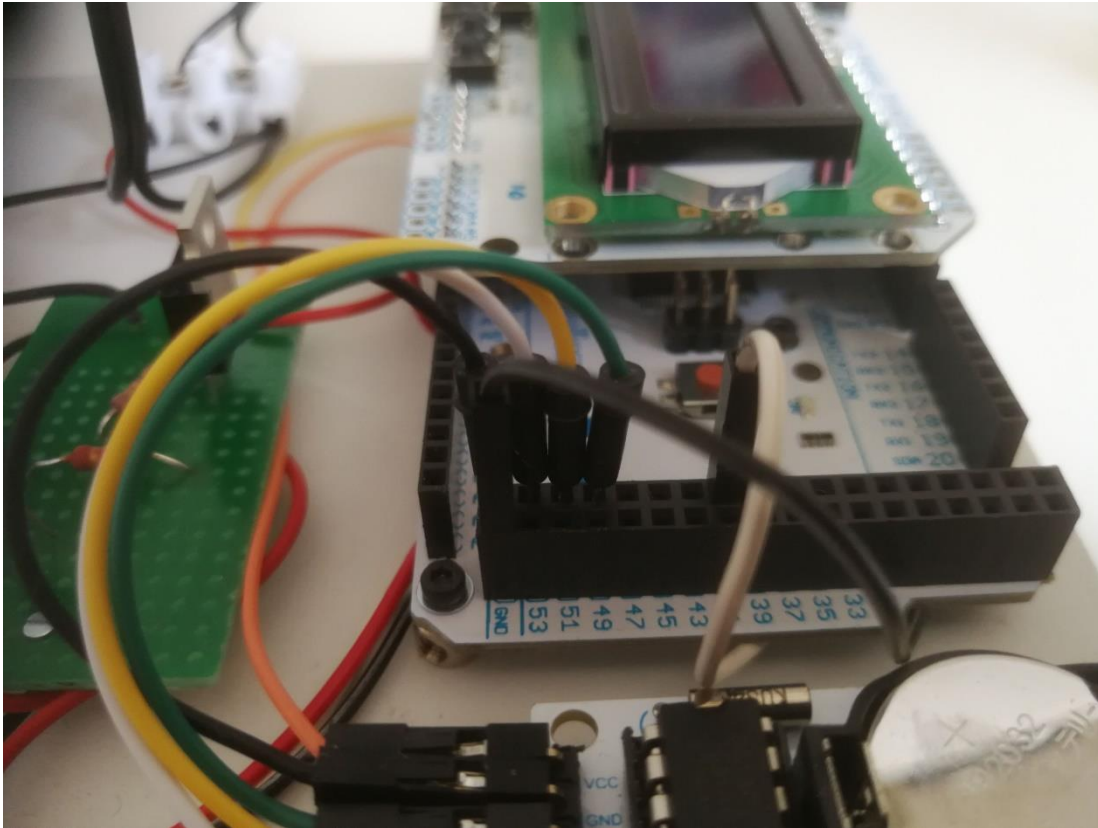
Overzicht



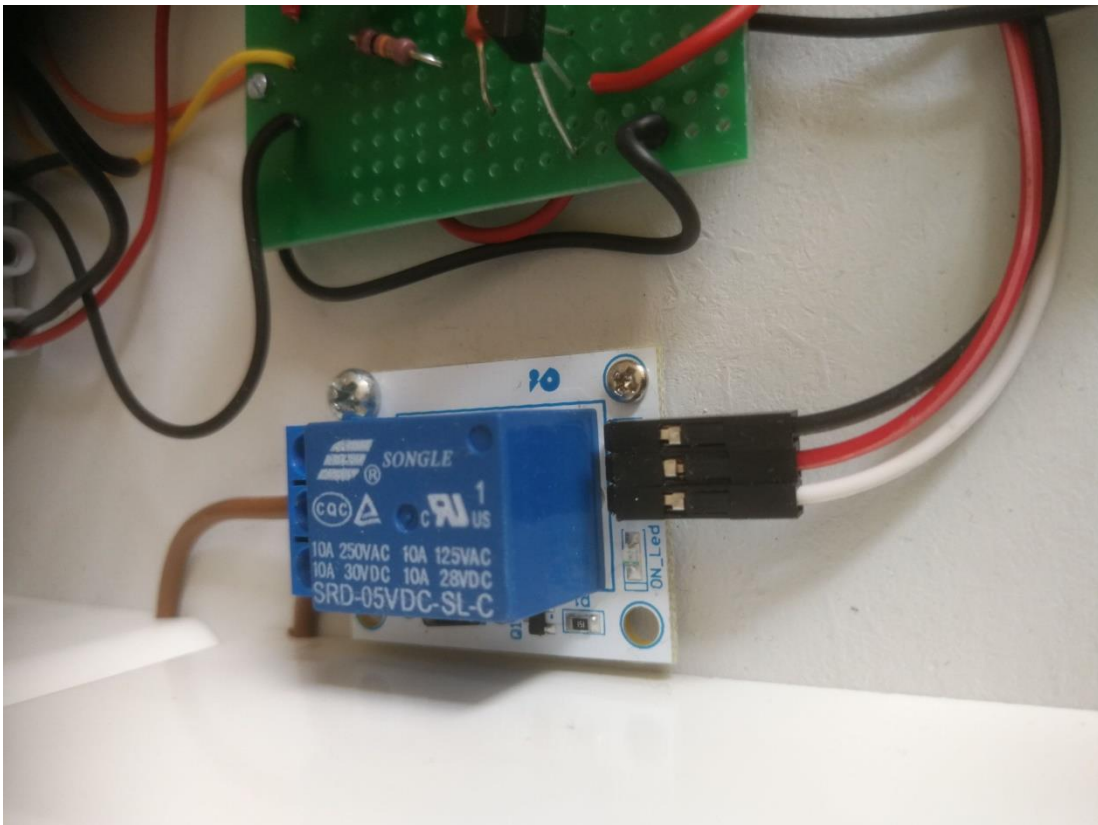
Clock aan ene kant



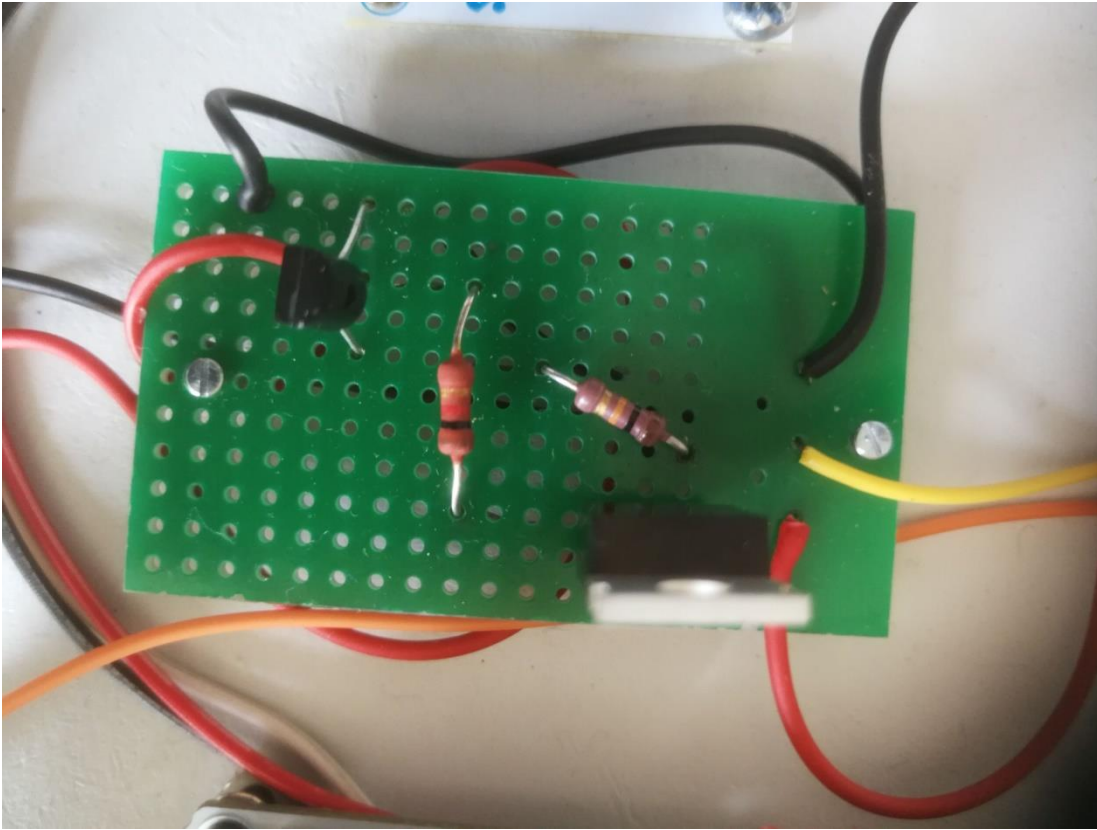
Clock aan de andere kant



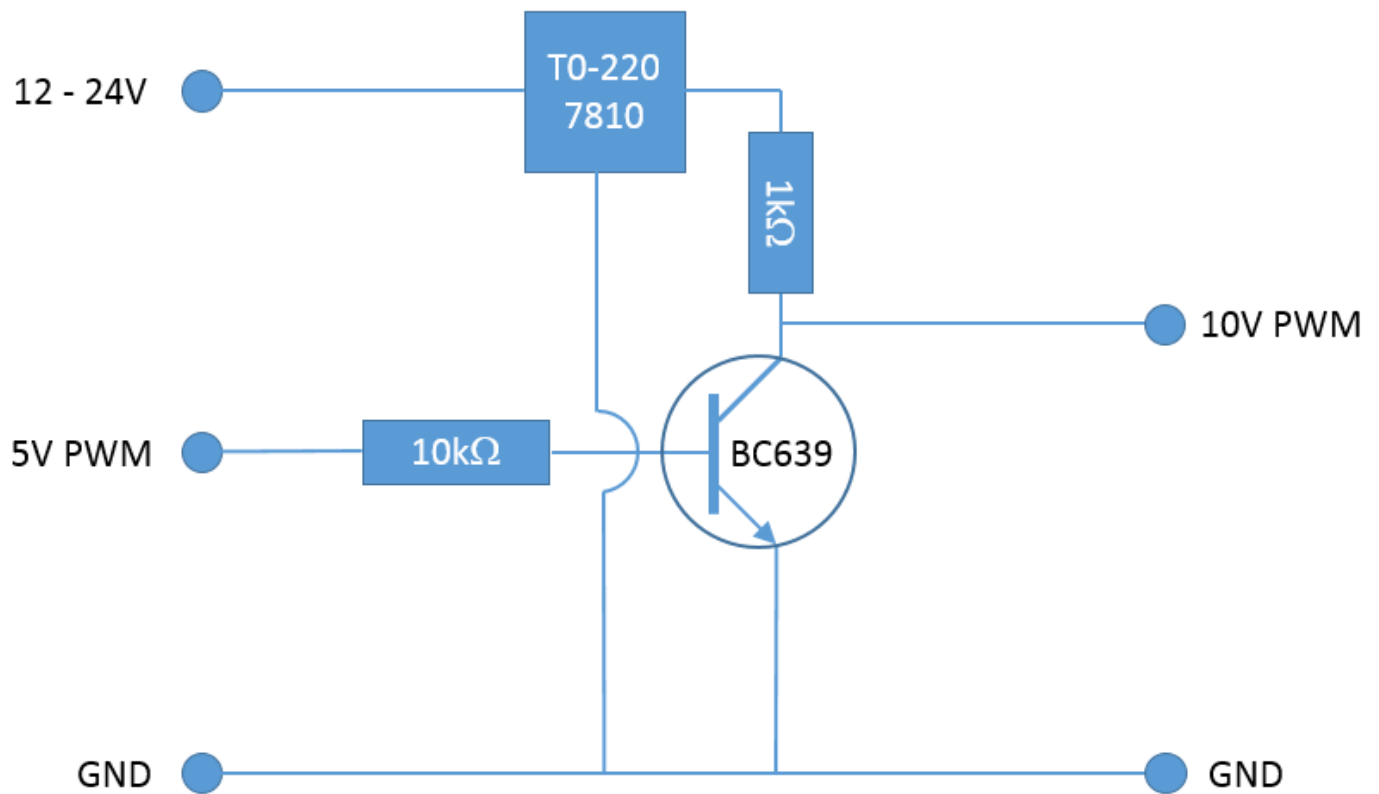
Relais



## Converter



Het schema van de converter is:



## Software

Voor het maken van het programma heb ik gebruik gemaakt van de Arduino programmeeromgeving. Zie ook één van de referenties. Voor dit project waren uiteindelijk twee verschillende programma's nodig. Het eerste programma is bedoeld om de Real Time Clock eenmalig op de juiste datum en tijd te zetten.

Daarna zorgt de stroomvoorziening, en de ingebouwde batterij, er voor dat de klok gewoon door blijft lopen. De enige reden waarom dit programma ooit nog een keer nodig zou kunnen zijn is als de batterij van de Real Time Clock vervangen moet worden. De procedure om dit uit te voeren staat in het hoofdstuk `ingebruikname`.

Het tweede programma, het programma dat 24 uur per dag zeven dagen in de week actief is, is bedoeld om uit de verschillende hoofdprogramma's, zoals gewenst door Vincent, te kunnen kiezen. Een prettige bijeenkomst van het gebruikte LCD shield zijn de drukknopjes waardoor ik in de software bepaalde acties kan nemen.

Er zijn in totaal zes drukknopjes waarvan ik in de huidige versie van de software er drie gebruik. De Select button is bedoeld om tussen de verschillende hoofdprogramma's te schakelen. De Up en Down buttons doen ongeveer hetzelfde.

Zoals in de inleiding al aangegeven zijn er vier hoofdprogramma's:

P0     verlichting uit.

P1     verlichting aan.

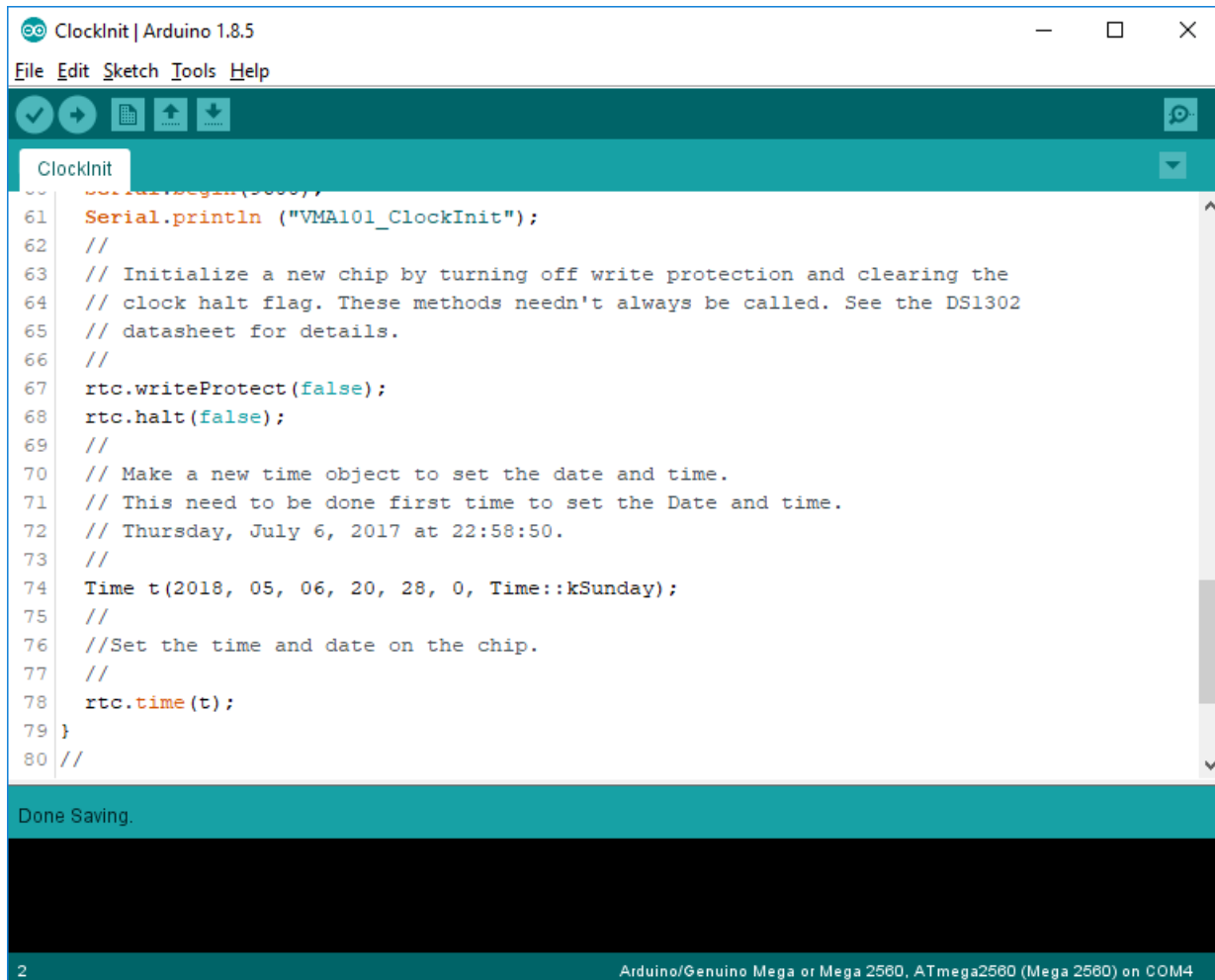
P2     verlichting met Sunrise om 16:00 uur en Sunset om 10:00 uur.

P3     verlichting met Sunrise om 16:00 uur en Sunset om 04:00 uur.

## Ingebruikname

Om een programma van een desktop computer of laptop in een Arduino te plaatsen heb je de Arduino programmeeromgeving nodig en een USB kabel tussen de laptop en de Arduino. De details van dit soort werk ga ik hier niet verder beschrijven.

Het eerste programma dat nodig is bij ingebruikname is ClockInit. Alvorens het programma uit te voeren op de Arduino moet in de setup functie de juiste datum en tijd ingesteld worden. Zie onderstaand screenshot:

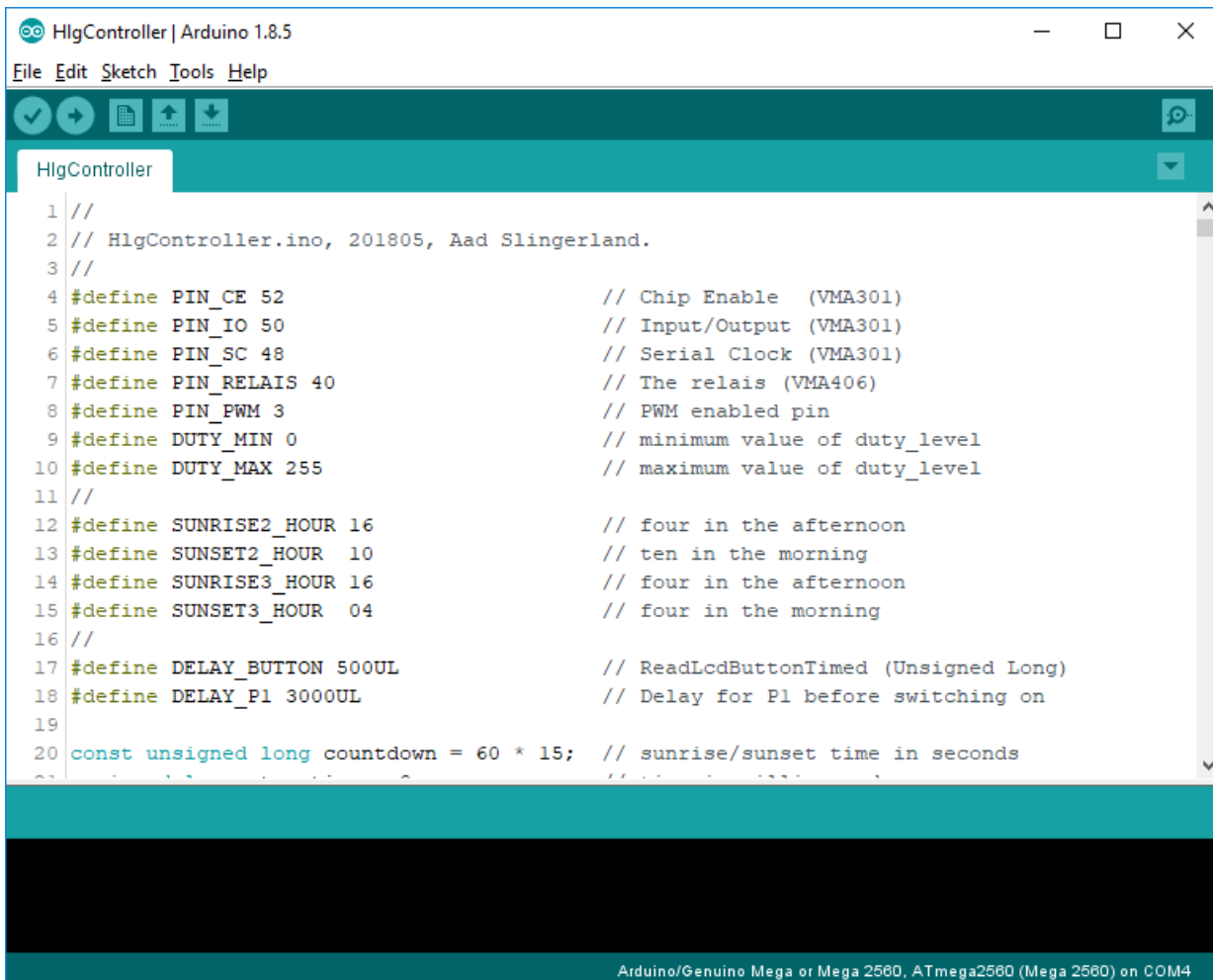
A screenshot of the Arduino IDE interface. The title bar reads 'ClockInit | Arduino 1.8.5'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for opening, saving, and running sketches. The main text area shows the 'ClockInit' sketch with the following code:

```
61 Serial.println ("VMA101_ClockInit");
62 //
63 // Initialize a new chip by turning off write protection and clearing the
64 // clock halt flag. These methods needn't always be called. See the DS1302
65 // datasheet for details.
66 //
67 rtc.writeProtect(false);
68 rtc.halt(false);
69 //
70 // Make a new time object to set the date and time.
71 // This need to be done first time to set the Date and time.
72 // Thursday, July 6, 2017 at 22:58:50.
73 //
74 Time t(2018, 05, 06, 20, 28, 0, Time::kSunday);
75 //
76 //Set the time and date on the chip.
77 //
78 rtc.time(t);
79 }
80 //
```

A status bar at the bottom indicates '2' and 'Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM4'. A teal banner at the bottom of the code editor says 'Done Saving.'

Het tweede programma, de aansturing van de Mean Well dimmer, kan daarna op de Arduino geplaatst worden. Eveneens weer met een laptop, of desktop computer, die met een USB-kabel aangesloten is aan de Arduino. Zie onderstaand screenshot:



The image shows a screenshot of the Arduino IDE interface. The title bar at the top reads "HlgController | Arduino 1.8.5". Below the title bar is a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Underneath the menu bar is a toolbar with icons for checking, running, saving, and other functions. The main area is a text editor displaying a C++ sketch named "HlgController". The sketch contains several preprocessor directives (#define) for pins and constants, and a const unsigned long variable named "countdown". The bottom status bar indicates the hardware is "Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM4".

```
1 //
2 // HlgController.ino, 201805, Aad Slingerland.
3 //
4 #define PIN_CE 52 // Chip Enable (VMA301)
5 #define PIN_IO 50 // Input/Output (VMA301)
6 #define PIN_SC 48 // Serial Clock (VMA301)
7 #define PIN_RELAIS 40 // The relais (VMA406)
8 #define PIN_PWM 3 // PWM enabled pin
9 #define DUTY_MIN 0 // minimum value of duty_level
10 #define DUTY_MAX 255 // maximum value of duty_level
11 //
12 #define SUNRISE2_HOUR 16 // four in the afternoon
13 #define SUNSET2_HOUR 10 // ten in the morning
14 #define SUNRISE3_HOUR 16 // four in the afternoon
15 #define SUNSET3_HOUR 04 // four in the morning
16 //
17 #define DELAY_BUTTON 500UL // ReadLcdButtonTimed (Unsigned Long)
18 #define DELAY_P1 3000UL // Delay for P1 before switching on
19
20 const unsigned long countdown = 60 * 15; // sunrise/sunset time in seconds
```

### Een stroomstoring

In geval van een stroomstoring is het van belang dat dit systeem zichzelf herstelt zodat de verlichting uit of aan is al naargelang het gekozen hoofdprogramma en het tijdstip van de stroomstoring. In de software is daarmee rekening gehouden omdat het gekozen hoofdprogramma (P0, P1, P2 of P3) vastgelegd wordt in een geheugenlocatie van de Real Time Clock.

Na een stroomstoring, of bij het resetten van de Arduino, wordt het juiste hoofdprogramma weer opgepakt. Daarna wordt, ingeval van de hoofdprogramma's P2 of P3, gekeken naar het tijdstip en wordt de verlichting eventueel weer ingeschakeld.

## Referenties

Dit project op github:

<https://github.com/aadslingerland>

De firma Velleman:

<https://www.velleman.eu>

Het Velleman support forum:

<https://forum.vellemanprojects.eu>

Arduino development software:

<https://www.arduino.cc/en/Main/Software>

Software voor de Real Time Clock aansturing (VMA301)

<http://digitalab.org/2017/08/real-time-clock-ds1302>

De 5V naar 10 V PWM converter:

<http://birota.azurewebsites.net/0-5v-to-0-10v-pwm-converter-for-arduino>

PWM informatie:

<https://provideyourown.com/2011/analogwrite-convert-pwm-to-voltage>



## Appendix A (ClockInit)

```
//
// ClockInit.ino, 201805, Aad Slingerland.
// http://digitalab.org/2017/08/real-time-clock-ds1302/
// Example sketch for interfacing with the DS1302 timekeeping chip.
//
#include <stdio.h>
#include <DS1302.h>
namespace
{
    //
    // Set the appropriate digital I/O pin connections. These are the pin
    // assignments for the Arduino as well for as the DS1302 chip.
    //
    const int CePin    = 52; // Chip Enable
    const int IoPin    = 50; // Input/Output
    const int SclkPin = 48; // Serial Clock
    //
    // Create a DS1302 object.
    //
    DS1302 rtc (CePin, IoPin, SclkPin);
    String dayAsString (const Time::Day day)
    {
        switch (day)
        {
            case Time::kSunday:    return "Sunday";
            case Time::kMonday:    return "Monday";
            case Time::kTuesday:   return "Tuesday";
            case Time::kWednesday: return "Wednesday";
            case Time::kThursday:  return "Thursday";
            case Time::kFriday:    return "Friday";
            case Time::kSaturday:  return "Saturday";
        }
        return ("unknown day");
    }

    void printTime()
    {
        //
        // Get the current time and date from the chip.
        //
        Time t = rtc.time ();
        //
        // Name the day of the week.
        //
        const String day = dayAsString (t.day);
        //
        // Format the time and date and insert into the temporary buffer.
        //
        char buf[50];
        snprintf (buf, sizeof(buf), "%s %04d-%02d-%02d %02d:%02d:%02d",
            day.c_str (), t.yr, t.mon, t.date, t.hr, t.min, t.sec);
        //
        // Print the formatted string to serial so we can see the time.
        //
        Serial.println (buf);
    }
}
```

```

    }
}

void setup() {
  Serial.begin(9600);
  Serial.println ("VMA101_ClockInit");
  //
  // Initialize a new chip by turning off write protection and clearing the
  // clock halt flag. These methods needn't always be called. See the DS1302
  // datasheet for details.
  //
  rtc.writeProtect(false);
  rtc.halt(false);
  //
  // Make a new time object to set the date and time.
  // This need to be done first time to set the Date and time.
  // Thursday, July 6, 2017 at 22:58:50.
  //
  Time t(2018, 05, 06, 20, 28, 0, Time::kSunday);
  //
  //Set the time and date on the chip.
  //
  rtc.time(t);
}
//
// Loop and print the time every second.
//
void loop ()
{
  printTime();
  delay(1000);
}

```

## Appendix B (HlgController)

```
////
// HlgController.ino, 201805, Aad Slingerland.
//
#define PIN_CE 52 // Chip Enable (VMA301)
#define PIN_IO 50 // Input/Output (VMA301)
#define PIN_SC 48 // Serial Clock (VMA301)
#define PIN_RELAIS 40 // The relais (VMA406)
#define PIN_PWM 3 // PWM enabled pin
#define DUTY_MIN 0 // minimum value of duty_level
#define DUTY_MAX 255 // maximum value of duty_level
//
#define SUNRISE2_HOUR 16 // four in the afternoon
#define SUNSET2_HOUR 10 // ten in the morning
#define SUNRISE3_HOUR 16 // four in the afternoon
#define SUNSET3_HOUR 04 // four in the morning
//
#define DELAY_BUTTON 500UL // ReadLcdButtonTimed (Unsigned Long)
#define DELAY_P1 3000UL // Delay for P1 before switching on
//
const unsigned long countdown = 60 * 15; // sunrise/sunset time in seconds
unsigned long step_time = 0; // time in milliseconds
byte duty_level = DUTY_MIN; // PWM duty level value
const String APP = "HlgController";
//
// For this DS1302 library see also:
// http://digitalab.org/2017/08/real-time-clock-ds1302 for the DS1302 library.
//
#include <DS1302.h>
DS1302 rtc (PIN_CE, PIN_IO, PIN_SC);
//
// Make an LCD object with the standard pins used on the VMA203.
//
#include <LiquidCrystal.h>
LiquidCrystal lcd (8, 9, 4, 5, 6, 7);
//
// Buttons defined.
//
enum Buttons : byte
{
    BTN_NONE,
    BTN_SELECT,
    BTN_LEFT,
    BTN_DOWN,
    BTN_UP,
    BTN_RIGHT
};
//
// Analogous values for buttons (VMA101 + VMA203).
//
enum AnalogousValues : int
{
    BTN_NONE_VAL = 1023,
    BTN_SELECT_VAL = 640,
    BTN_LEFT_VAL = 409,
    BTN_DOWN_VAL = 254,
```

```

    BTN_UP_VAL      = 98,
    BTN_RIGHT_VAL   = 0,
    BTN_RANGE       = 50
};
//
// Determine which button has been pressed.
//
int ReadLcdButton ()
{
    int btn_val = analogRead (0);
    //
    // Serial.print    ("ReadLcdButton: btn_val=");
    // Serial.println (btn_val);
    //
    if (btn_val > (BTN_NONE_VAL - BTN_RANGE)) return BTN_NONE;
    if (btn_val > (BTN_SELECT_VAL - BTN_RANGE)) return BTN_SELECT;
    if (btn_val > (BTN_LEFT_VAL - BTN_RANGE)) return BTN_LEFT;
    if (btn_val > (BTN_DOWN_VAL - BTN_RANGE)) return BTN_DOWN;
    if (btn_val > (BTN_UP_VAL - BTN_RANGE)) return BTN_UP;
    if (btn_val > (BTN_RIGHT_VAL - BTN_RANGE)) return BTN_RIGHT;
    //
    Serial.println ("LcdReadButton: this line should not occur.");
    //
    return BTN_NONE;
}
//
// This method prevents button presses to overflow the state machine
// by keeping the last button pressed noted for a short while.
//
byte ReadLcdButtonTimed ()
{
    static byte last_button = 0;
    static unsigned long last_button_time = 0;
    //
    byte button = ReadLcdButton ();
    //
    Serial.print    ("ReadLcdButtonTimed: button=");
    Serial.println (button);
    //
    // Only perform the timed delay for buttons other then BTN_NONE.
    //
    if (button != BTN_NONE)
    {
        if (button != last_button)
        {
            last_button = button;
            last_button_time = millis ();
            //
            Serial.print    ("ReadLcdButtonTimed: last_button_time=");
            Serial.println (last_button_time);
        }
        else
        {
            unsigned long timed = DELAY_BUTTON - (millis () - last_button_time);
            //
            Serial.print    ("ReadLcdButtonTimed: delay=");
            Serial.println (timed);
        }
    }
}

```

```

        //
        delay (timed);
        last_button = BTN_NONE;
        button = BTN_NONE;
    }
}
else
{
    last_button = BTN_NONE;
}
return button;
}
//
// A small class to encapsulate the operation of a VMA405 relay.
//
class Relay
{
public:
    Relay (byte pin)
    {
        _pin = pin;
    }
    void On ()
    {
        pinMode (_pin, OUTPUT);
        digitalWrite (_pin, HIGH);
    }
    void Off ()
    {
        pinMode (_pin, OUTPUT);
        digitalWrite (_pin, LOW);
    }
private:
    byte _pin;
};
Relay rel (PIN_RELAIS);
//
enum Main_states : byte
{
    STATE_P0 = 0,
    STATE_P1 = 1,
    STATE_P2 = 2,
    STATE_P3 = 3
};
byte current_main_state;
//
enum P1_states : byte
{
    P1_WAITING = 0,
    P1_ON      = 1
};
byte current_p1_state = P1_WAITING;
//
enum P2_states : byte
{
    P2_OFF      = 0,
    P2_SUNRISE = 1,

```

```

P2_ON      = 2,
P2_SUNSET  = 3
};
byte current_p2_state = P2_OFF;
boolean dirty_p2_state = true;
//
void setup ()
{
  Serial.begin (9600);
  Serial.println (" ");
  Serial.println (APP);
  //
  lcd.begin (16, 2);
  lcd.clear ();
  //
  // The previous current_main_state is saved in clock module. Get it from there.
  //
  byte b0 = rtc.readRam (0);
  if (b0 >= STATE_P0 && b0 <= STATE_P3)
  {
    current_main_state = b0;
  }
  else
  {
    current_main_state = STATE_P0;
  }
  //
  Serial.print ("setup: restored current_main_state=");
  Serial.println (current_main_state);
  //
  // How long takes each step (255 in total) to switch from fully on
  // (duty_level = 255) to fully off (duty_level = 0) or vice versa.
  //
  step_time = (countdown * 1000 / DUTY_MAX);
  //
  Serial.print ("setup: step_time=");
  Serial.println (step_time);
}
//
void loop ()
{
  //
  // Small delay for debugging in serial monitor.
  // Comment this out before running in production.
  //
  // delay (100);
  //
  // The date and time on the first row.
  //
  lcd.setCursor (0, 0);
  Time t = rtc.time ();
  char str[17];
  snprintf (str, sizeof(str), "%04d%02d%02d %02d%02d%02d",
            t.yr, t.mon, t.date, t.hr, t.min, t.sec);
  lcd.print (str);
  //
  Serial.print ("loop: date_and_time=");

```

```

Serial.println (str);
//
// The current main state (program) on the second row.
//
lcd.setCursor (7, 1);
lcd.print ("P");
lcd.print (current_main_state);
//
// The current sub-state (if applicable) just after that.
//
if (current_main_state == STATE_P1)
{
    lcd.print (current_p1_state);
}
else if (current_main_state == STATE_P2 || current_main_state == STATE_P3)
{
    lcd.print (current_p2_state);
}
else
{
    lcd.print (" ");
}
//
// The duty_level expressed as a percentage on the second row.
// Note that the map function is used to inverse the duty_level
// values because the PWM signal amplifier does an inversion.
//
int p_num = map (duty_level, 0, 255, 100, 0);
char p_str[6];
snprintf (p_str, sizeof(p_str), "%3d%%", p_num);
lcd.setCursor (12, 1);
lcd.print (p_str);
//
// Set the cursor for the button text.
//
lcd.setCursor (0, 1);
//
byte button = ReadLcdButtonTimed ();
//
switch (button)
{
    case BTN_NONE:
    {
        lcd.print ("NONE ");
        break;
    }
    case BTN_SELECT:
    {
        lcd.print ("SELECT");
        set_state_select ();
        break;
    }
    case BTN_LEFT:
    {
        lcd.print ("LEFT ");
        break;
    }
}

```



```

    case BTN_DOWN:
    {
        lcd.print ("DOWN ");
        set_state_down ();
        break;
    }
    case BTN_UP:
    {
        lcd.print ("UP ");
        set_state_up ();
        break;
    }
    case BTN_RIGHT:
    {
        lcd.print ("RIGHT ");
        break;
    }
}
//
run_main ();
}
//
// Switch to a next main state (program) when the SELECT button is pressed.
//
void set_state_select ()
{
    switch (current_main_state)
    {
        case STATE_P0:
            current_main_state = STATE_P1;
            break;

        case STATE_P1:
            current_main_state = STATE_P2;
            break;

        case STATE_P2:
            current_main_state = STATE_P3;
            break;

        case STATE_P3:
            current_main_state = STATE_P0;
            break;
    }
    //
    // Save current main state for Setup ().
    //
    rtc.writeRam (0, current_main_state);
    //
    // Indicate that an initial substate for P2 P3 should be done.
    //
    dirty_p2_state = true;
}
//
// Switch to a higher main state (program) when the UP button is pressed.
//
void set_state_up ()

```

```

{
switch (current_main_state)
{
case STATE_P0:
break;

case STATE_P1:
current_main_state = STATE_P0;
break;

case STATE_P2:
current_main_state = STATE_P1;
break;

case STATE_P3:
current_main_state = STATE_P2;
break;
}
//
// Save current main state for Setup ().
//
rtc.writeRam (0, current_main_state);
//
// Indicate that an initial substate for P2 P3 should be done.
//
dirty_p2_state = true;
}
//
// Switch to a lower main state (program) when the DOWN button is pressed.
//
void set_state_down ()
{
switch (current_main_state)
{
case STATE_P0:
current_main_state = STATE_P1;
break;

case STATE_P1:
current_main_state = STATE_P2;
break;

case STATE_P2:
current_main_state = STATE_P3;
break;

case STATE_P3:
break;
}
//
// Save current main state for Setup ().
//
rtc.writeRam (0, current_main_state);
//
// Indicate that an initial substate for P2 P3 should be done.
//
dirty_p2_state = true;

```

```

}
//
// Run one of the main programs (state).
//
void run_main ()
{
    //
    // First the value of current_p1_state is set to P1_WAITING because the
    // run_P1 () function has no way to find out when to do so.
    //
    if (current_main_state != STATE_P1)
    {
        current_p1_state = P1_WAITING;
    }
    //
    switch (current_main_state)
    {
        case STATE_P0:
            run_P0 ();
            break;

        case STATE_P1:
            run_P1 ();
            break;

        case STATE_P2:
            run_P2 (SUNRISE2_HOUR, SUNSET2_HOUR);
            break;

        case STATE_P3:
            run_P2 (SUNRISE3_HOUR, SUNSET3_HOUR);
            break;
    }
}
//
// Program 0. Switch off immediate.
//
void run_P0 ()
{
    Serial.println ("run_P0");
    //
    rel.On ();
    duty_level = DUTY_MAX;
    analogWrite (PIN_PWM, duty_level);
}

//
// Program 1. Switch on but with a small delay.
//
void run_P1 ()
{
    Serial.println ("run_P1");
    //
    static unsigned long entry_time = 0;
    boolean rb = false;
    //
    switch (current_p1_state)

```

```

{
  case P1_WAITING:
    if (entry_time == 0)
    {
      Serial.println ("run_P1: switching off.....");
      //
      rel.On ();
      duty_level = DUTY_MAX;
      analogWrite (PIN_PWM, duty_level);
      //
      entry_time = millis ();
      //
      Serial.print  ("run_P1: entry_time=");
      Serial.println (entry_time);
    }
    rb = HasTimedWaitPassed (entry_time, DELAY_P1);
    if (rb == true)
    {
      current_p1_state = P1_ON;
      entry_time = 0;                      // reset this for the next time
      //
      Serial.println ("run_P1: switching to P1_ON state");
    }
    break;

  case P1_ON:
    entry_time = 0;                      // reset this for the next time
    //
    rel.Off ();
    duty_level = DUTY_MIN;
    analogWrite (PIN_PWM, duty_level);
    break;
}
}
//
// Program 2. Switch on the LEDS for a number of hours per day depending on
// the parameters passed as SUNRISE and SUNSET times. This routine is used
// for both P2 and P3 but with different parameter values.
//
void run_P2 (byte this_sunrise, byte this_sunset)
{
  Serial.print  ("run_P2: this_sunrise=");
  Serial.print  (this_sunrise);
  Serial.print  ("", this_sunset=");
  Serial.println (this_sunset);
  //
  boolean rb;
  Time this_time = rtc.time();
  //
  Serial.print  ("run_P2: millis=");
  Serial.println (millis ());
  //
  // Are we in the first (half) second that this program is running?
  // Or, was the main program state changed from 2 to 3 or vice versa?
  // If so, proceed with adjusting the initial current_p2_state.
  // Adjusting the initial state is important for the very first time this
  // program runs and, more important, in the case of a power outage.

```

```

//
if (dirty_p2_state == true)
{
    dirty_p2_state = false;
    current_p2_state = P2_OFF;           // assume this, unless proven otherwise
    //
    if (this_time.hr == this_sunrise)    // inside the sunrise hour?
    {
        if (this_time.min < (countdown / 60)) // inside the countdown?
        {
            current_p2_state = P2_SUNRISE;    // a new awakening
        }
        else
        {
            current_p2_state = P2_ON;          // sunrise already done
        }
    }
    else if (this_time.hr > this_sunrise)    // already between sunrise and midnight?
    {
        current_p2_state = P2_ON;
    }
    else if (this_time.hr < this_sunset)    // between midnight and sunset?
    {
        current_p2_state = P2_ON;
    }
    //
    Serial.print ("run_P2: calculated initial state=");
    Serial.println (current_p2_state);
    //
}
//
switch (current_p2_state)
{
    case P2_OFF:
        rel.On ();
        duty_level = DUTY_MAX;
        analogWrite (PIN_PWM, duty_level);
        //
        if (this_time.hr == this_sunrise)
        {
            current_p2_state = P2_SUNRISE;
            //
            Serial.println ("run_P2: switching to P2_SUNRISE state");
        }
        break;

    case P2_SUNRISE:
        rel.Off ();
        rb = SunRise ();
        if (rb == true)
        {
            current_p2_state = P2_ON;
            //
            Serial.println ("run_P2: switching to P2_ON state");
        }
        break;
}

```

```

case P2_ON:
    rel.Off ();
    duty_level = DUTY_MIN;
    analogWrite (PIN_PWM, duty_level);
    //
    if (this_time.hr == this_sunset)
    {
        current_p2_state = P2_SUNSET;
        //
        Serial.println ("run_P2: switching to P2_SUNSET state");
    }
    break;

case P2_SUNSET:
    rb = SunSet ();
    if (rb == true)
    {
        current_p2_state = P2_OFF;
        //
        Serial.println ("run_P2: switching to P2_OFF state");
    }
    break;
}
}
//
// Initiate or continue a Sunrise event.
// Returns false when not yet done, true when done.
//
boolean Sunrise ()
{
    Serial.println ("SunRise");
    //
    static unsigned long entry_time = 0;
    boolean rb = false;
    //
    Serial.print  ("SunRise: step_time=");
    Serial.print  (step_time);
    Serial.print  (". entry_time=");
    Serial.println (entry_time);
    //
    if (entry_time == 0)
    {
        entry_time = millis ();
    }
    //
    rb = HasTimedWaitPassed (entry_time, step_time);
    if (rb == true)
    {
        if (duty_level > DUTY_MIN)
        {
            duty_level--;
            analogWrite (PIN_PWM, duty_level);
            //
            Serial.print  ("SunRise: duty_level=");
            Serial.println (duty_level);
            //
            entry_time = millis ();

```

```

    //
    Serial.print  ("SunRise: new entry_time=");
    Serial.println (entry_time);
    //
    rb = false;                                // SunRise still in progress
}
else
{
    rb = true;                                // SunRise has just completed
}
}
//
Serial.print  ("SunRise: return=");
Serial.println (rb);
Serial.println ("-----");
//
return rb;
}
//
// Initiate or continue a SunSet event.
// Returns false when not yet done, true when done.
//
boolean SunSet ()
{
    Serial.println ("SunSet");
    //
    static unsigned long entry_time = 0;
    boolean rb = false;
    //
    Serial.print  ("SunSet: step_time=");
    Serial.print  (step_time);
    Serial.print  (". entry_time=");
    Serial.println (entry_time);
    //
    if (entry_time == 0)
    {
        entry_time = millis ();
    }
    //
    rb = HasTimedWaitPassed (entry_time, step_time);
    if (rb == true)
    {
        if (duty_level < DUTY_MAX)
        {
            duty_level++;
            analogWrite (PIN_PWM, duty_level);
            //
            Serial.print  ("SunSet: duty_level=");
            Serial.println (duty_level);
            //
            entry_time = millis ();
            //
            Serial.print  ("SunSet: new entry_time=");
            Serial.println (entry_time);
            //
            rb = false;                                // SunSet still in progress
        }
    }
}

```



```

    else
    {
        rb = true;                                // SunSet has just completed
    }
}
//
Serial.print  ("SunSet: return=");
Serial.println (rb);
Serial.println ("-----");
//
return rb;
}
//
// Function to determine if a certain amount of milliseconds have passed since
// a specified entry_time. In case of an overflow of the result of the millis ()
// function (approximate each fifty days) this function will not fail but will
// return true so the program using this method will not stall.
//
boolean HasTimedWaitPassed (unsigned long entry_time, unsigned int wait)
{
    boolean rb = false;                            // assume not yet passed
    unsigned long this_time = millis ();
    //
    Serial.print  ("HasTimedWaitPassed: this_time=");
    Serial.println (this_time);
    //
    if (this_time >= entry_time)
    {
        if (this_time > (entry_time + wait))
        {
            rb = true;                            // wait has passed
        }
    }
    else
    {
        Serial.println ("HasTimedWaitPassed: overflow detected.");
        rb = true;                                // wait has not passed but...
    }
    //
    Serial.print  ("HasTimedWaitPassed: rb=");
    Serial.println (rb);
    //
    return rb;
}

```