Arduino, ESP8266, ESP32 & Raspberry Pi stuff

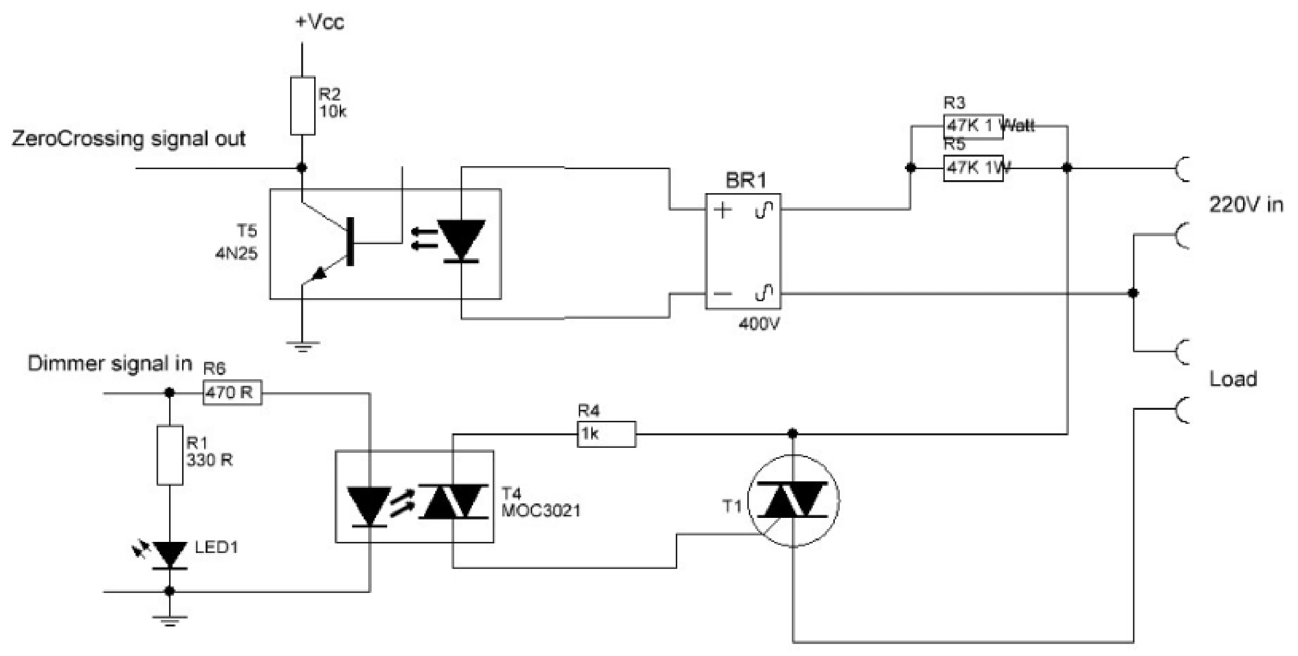
Arduino and related stuff (including Attiny and ESP8266) and the Raspberry Pi

Category: dimmer

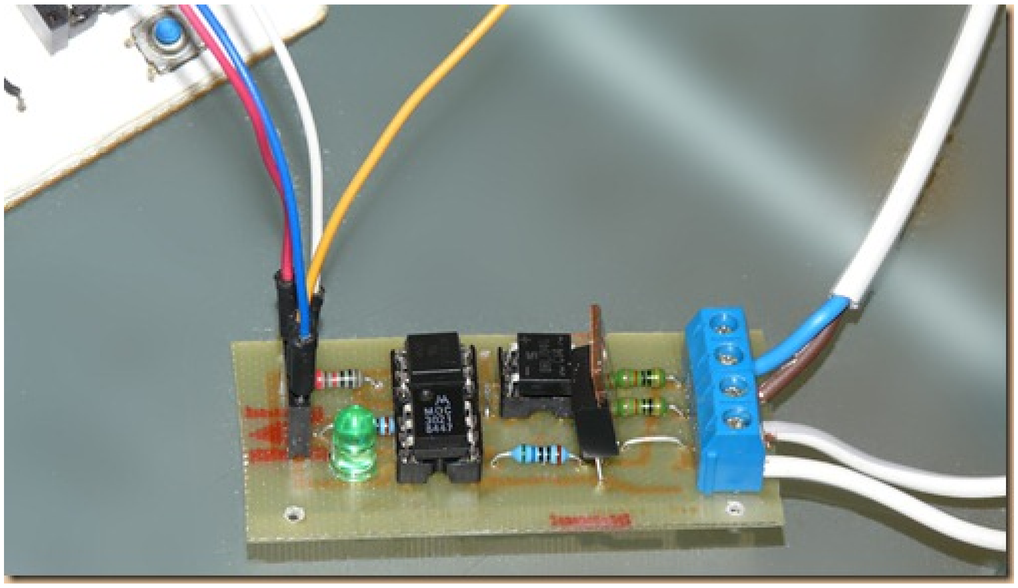
Dimming an AC lamp via MQTT

Years ago I published a TRIAC dimmer that could be controlled by a simple microcontroller such as an arduino. Times have changed and right now it is of more importance to be able to control that lamp from a home automation system such as OpenHab, Homematic or NodeRed.

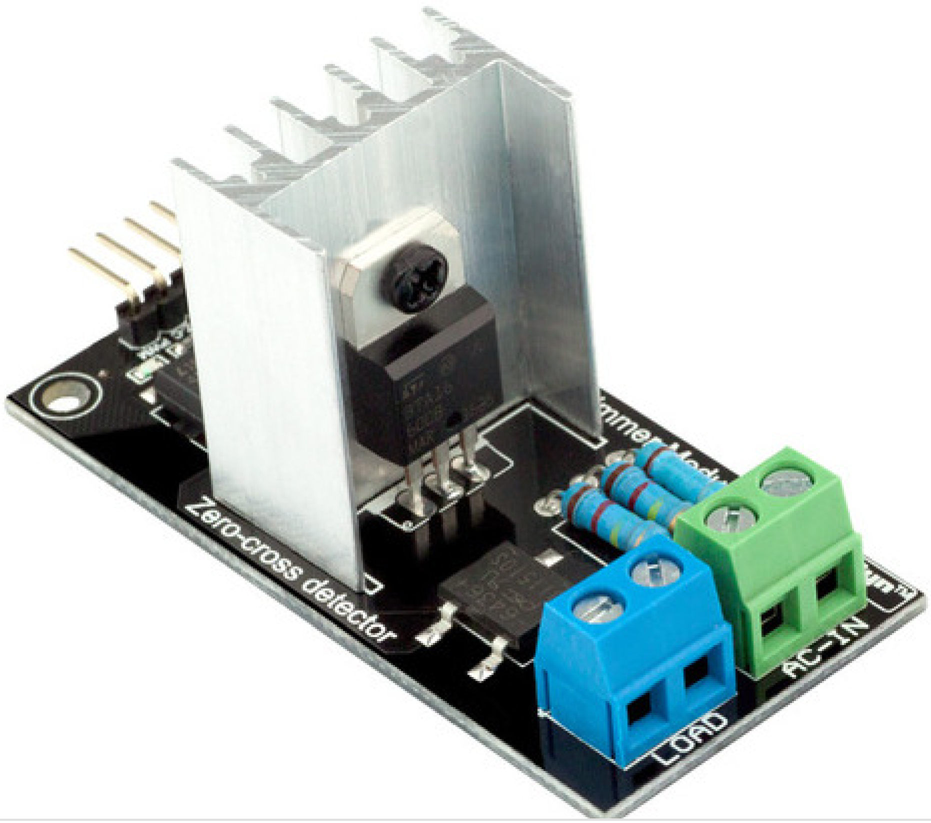
So, let's have a look at the circuit first:



microcontroller ignites the TRIAC with a time delay that determines the brightness of the attached Lamp. The full cycle is 10mS (at 50Hz)
A circuit like this is easy to make for very low cost (1-2 Euro).



However, if you shy away from soldering such a circuit, there are ready made modules available as well:



You should not need to pay more than 3-4 USD for such a module. Anything above that is robbery

Software to control this dimmer would look something like this:

```
#include <Ethernet.h>
#include <PubSubClient.h>
#include  <TimerOne.h>

#define CLIENT_ID       "Dimmer"
#define PUBLISH_DELAY   3000

String ip = "";
bool startsend = HIGH;
uint8_t mac[6] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x07};

volatile int i=0;               // Variable to use as a counter volatile as
it is in an interrupt
volatile boolean zero_cross=0;  // Boolean to store a "switch" to tell us if
we have crossed zero
int AC_pin1 = 4;// Output to Opto Triac
int dim1 = 0;// Dimming level (0-100)  0 = on, 100 = 0ff

int inc=1;          // counting up or down, 1=up, -1=down
int freqStep = 100; // make this 83 for 60Hz gridfrequency

EthernetClient ethClient;
PubSubClient mqttClient;
long previousMillis;

void zero_cross_detect() {
  zero_cross = true;               // set the boolean to true to tell our
dimming function that a zero cross has occured
  i=0;
  digitalWrite(AC_pin1, LOW);       // turn off TRIAC (and AC)
}

// Turn on the TRIAC at the appropriate time
void dim_check() {
      i=0;  // reset time step counter
      zero_cross = false; //reset zero cross detection
    }
    else {
      i++; // increment time step counter
    }
  }
}

void setup() {

  attachInterrupt(0, zero_cross_detect, RISING);    // Attach an Interupt to
Pin 2 (interupt 0) for Zero Cross Detection
  Timer1.initialize(freqStep);                      // Initialize TimerOne
library for the freq we need
  Timer1.attachInterrupt(dim_check, freqStep);
  pinMode(4, OUTPUT);

  // setup serial communication

  Serial.begin(9600);
  while (!Serial) {};
  Serial.println(F("dimmer"));
  Serial.println();

  // setup ethernet communication using DHCP
  if (Ethernet.begin(mac) == 0) {
    Serial.println(F("Unable to configure Ethernet using DHCP"));
    for (;;);
  }

  Serial.println(F("Ethernet configured via DHCP"));
  Serial.print("IP address: ");
  Serial.println(Ethernet.localIP());
  Serial.println();

  ip = String (Ethernet.localIP()[0]);
  ip = ip + ".";
  ip = ip + String (Ethernet.localIP()[1]);
  ip = ip + ".";
  ip = ip + String (Ethernet.localIP()[2]);
  ip = ip + ".";
  ip = ip + String (Ethernet.localIP()[3]);
  mqttClient.setClient(ethClient);
  mqttClient.setServer( "192.168.1.103", 1883); // <= put here the address
of YOUR MQTT server //Serial.println(F("MQTT client configured"));
mqttClient.setCallback(callback); Serial.println(); Serial.println(F("Ready
to send data")); previousMillis = millis();
mqttClient.publish("home/br/nb/ip", ip.c_str()); } void loop() { // it's
time to send new data? if (millis() - previousMillis > PUBLISH_DELAY) {
  sendData();
  previousMillis = millis();

  }

  mqttClient.loop();
  Serial.print("dim1 in loop = ");
  Serial.println(dim1);
}

void sendData() {
  char msgBuffer[20];
  if (mqttClient.connect(CLIENT_ID)) {
    mqttClient.subscribe("home/br/sb");
    if (startsend) {

      mqttClient.publish("home/br/nb/ip", ip.c_str());
      startsend = LOW;
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  char msgBuffer[20];

   payload[length] = '\0';            // terminate string with '0'
  String strPayload = String((char*)payload);  // convert to string
  Serial.print("strPayload =  ");
  Serial.println(strPayload); //can use this if using longer southbound
topics
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");//MQTT_BROKER
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
dim1=strPayload.toInt();

}
```

The value freqStep is usually set at 75 or 78 with this type of circuits, which allows for 128 steps of dimming at 50Hz gridfrequency. I have set the value here purposely to 100 allowing for only 100 steps, which is convenient for use in OpenHAB as the slider goes from 0-100%.
The calculation is as follows:

EDIT 2019 There is a library for the Arduino, the ESP8266 and the ESP32.

50Hz
As we have two zerocrossings per sine wave, the frequency of zerocrossings is 100Hz. Therefore the period between two zerocrossings is 10mSec is 10000uS.
So if you want to have 100 levels of dimming the steps you need to take are 10000/100=100uS

60Hz
The frequency of zerocrossings is 120Hz, therefore the period between two zerocrossings is 8.3mS is 8300uS. So if you want to have 100 levels of dimming, the steps you need to take are 8300/100=83uS -> freqStep=83

January 23, 2018 / Arduino, dimmer, MQTT, openhab / 49 Comments

AC dimming with PWM and Arduino

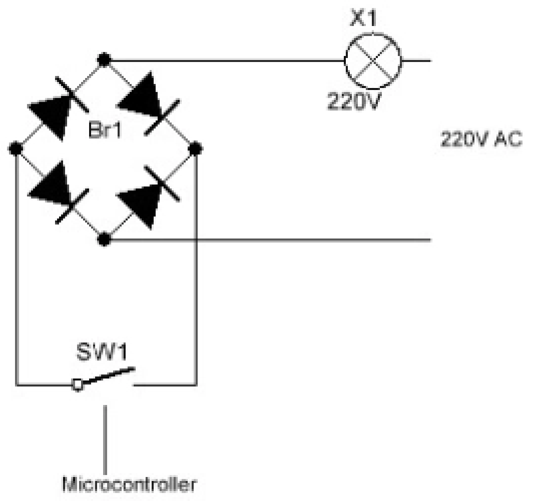
Almost 3 years ago, I published a simpel TRIAC AC dimmer for the arduino. That proved to be a very popular design. Yet in spite of the simplicity of the circuit the software needed was a bit complicated as it needed to keep track of the zero crossing of the AC signal, then keep track of the time and then finally open the TRIAC.
So to avoid letting the arduino just wait for most of the time, an interrupt and a timer were necessary.

So why can't we just use PWM, like with LED's? Well, I explained that in that instructable, but there are possibilities to do that. Someone looking for that would no doubt end up at design by Ton Giesberts/Elektor Magazine that can do PWM of an AC source.

That will work, but in spite of my admiration for Giesberts and Elektor, there is something fundamentally wrong with that circuit. I think it is necessary that I explain what is wrong before I come up with improvements.
If you are not interested in the technical details, just skip to the next step.

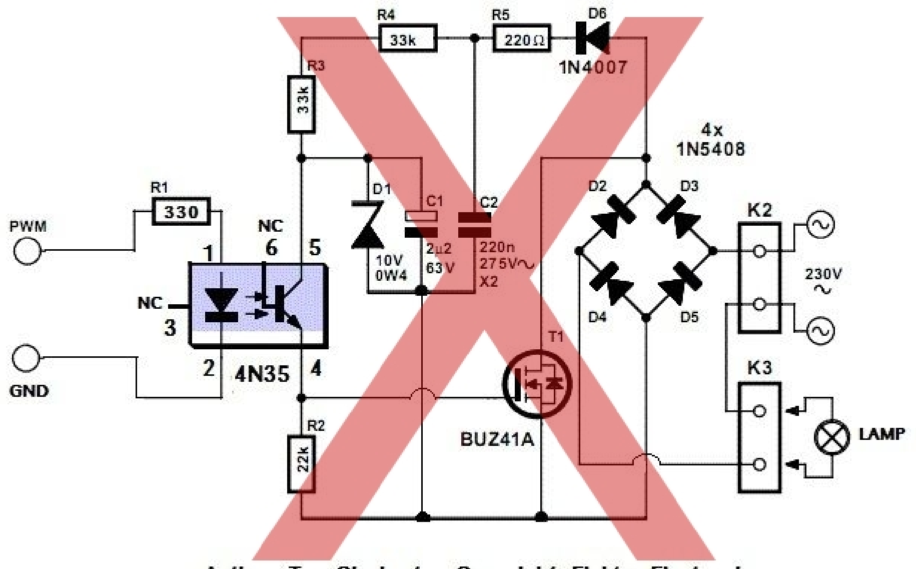
A lamp, and a switch, but as in fact the switching is done in DC rather than AC, it

becomes a lamp, a bridge rectifier and a switch.
That switch, which is in fact the MOSFET and the components around it is controlled by the Arduino (or PIC or whatever). So, switching that on and off in a certain duty cycle will switch the lamp on and off and if done fast enough the lamp won't be seen anymore as flickering, but as being dimmed, similar as we do with LED's and PWM.



So far so good. The theory behind the circuit is sound. However, the MOSFET needs a voltage on its gate to be switched on and as we cannot get that from an arduino for obvious reasons (it is only 5 Volt, which isnt enough AND you don't want your arduino to be connected to the mains grid), Giesberts uses an optocoupler. That optocoupler still needs a DC voltage and Giesberts is using the to DC rectified AC voltage for that.

And that is where the problems start, because he is feeding the gate from the MOSFET, with a voltage that is shorted by that same MOSFET. In other words, if teh MOSFET is fully opened the DC voltage coming from the rectifier is



Author: Ton Giesberts - Copyright: Elektor Electronics

This effect might not be so outspoken by a low dutycycle (= lamp on a low intensity),

because of the presence of C1, that will retain its charge for a while and will be receiving new charge thanks to the low dutycycle, but at 25-80% dutycycle the voltage on C1 just cannot be sustained anymore and the lamp may start to flicker. What's worse is that at moments that the voltage on the gate drops, for a while the MOSFET will be still conducting, but not be fully saturized: it will slowly go from its nominal 0.04 Ohm resistance to infinite resistance and the slower this goes, the higher the power that needs to be dissipated in the MOSFET. That means a lot of heat. MOSFETS are good switches but bad resistors. They need to be switched ON and OFF fast.
Currently the circuit heavily relies on D1 to keep the voltage on the gate of T1 at acceptable limits while the voltage is swinging between 0 Volt and Full peak
At peak the rectified voltage is 230×1.4=330V
The average rectified voltage is 230×0.9=207V

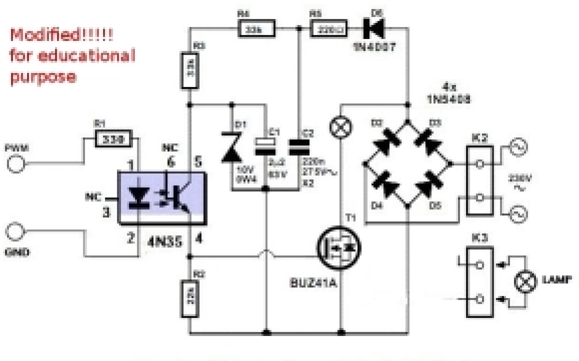
If we forget about the smoothing effect of the capacitor for a while and presume the optocoupler to be fully open the average voltage on the capacitor would be 22/88 * 207 =52 Volts and in peak 22/88 * 330= 83 Volts. That it is not is because of D1 and the fact that the MOSFET will short the Voltage.

If the optocoupler is not in saturation and its impedance therefore infinite, the capacitor C1 would charge up to full rectified voltage if not for D1. On average 3mA will flow through R3,R4 and R5 (207-10)/66k which equals a power consumption of 0.6 Watt in the resistors R3,R4, R5

Improvements
The problems mentioned with the Giesberts circuit can be remedied by putting the lamp somewhere else: remove it from the AC line and put it in the Drain of the MOSFET. For the lamp it doesnt really matter if it is receiving DC or AC. You could make more improvements, as now there is no need to cater for a a voltage swinging between 0 and 330 Volt

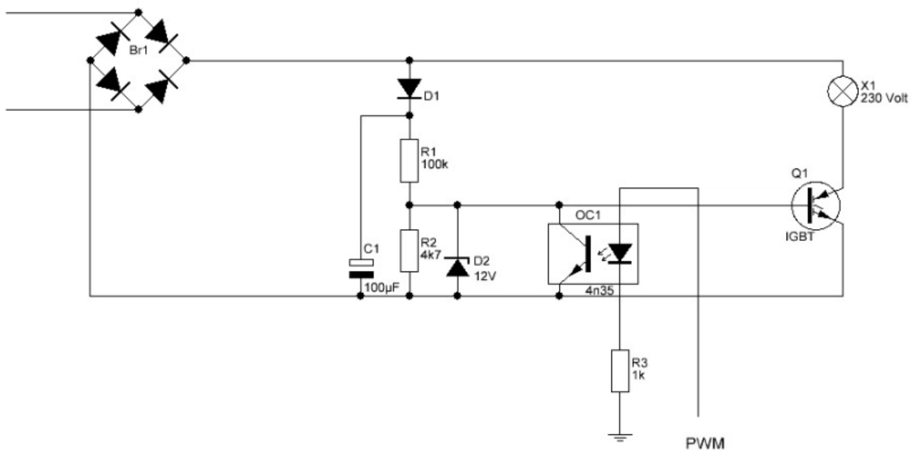
But as we are changing the design, we might as well take it a step further and use an

ideal switch. Thus we can come to the following circuit:



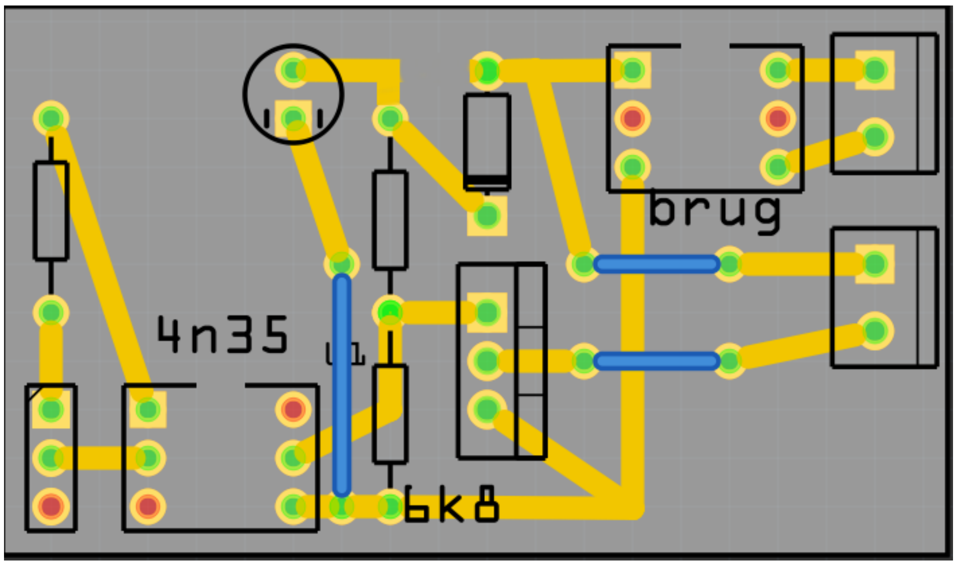
Author: Ton Giesberts - Copyright: Elektor Electronics

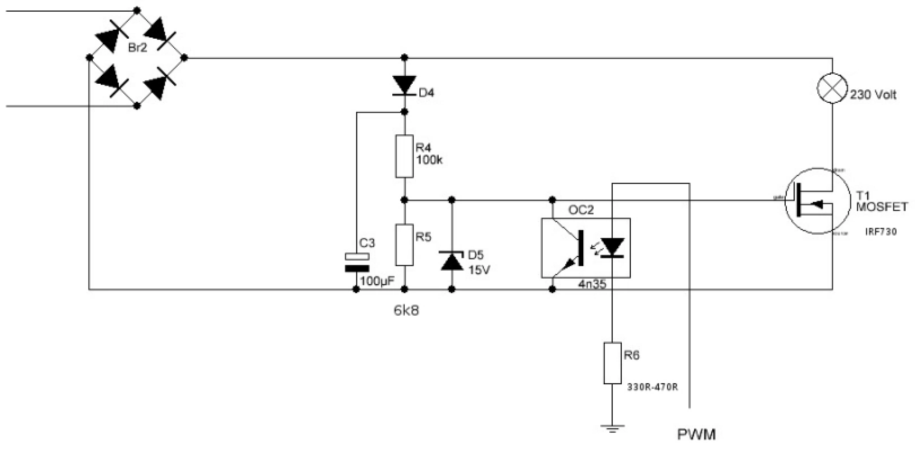
The IRBT acts as a fast switch that either switches the lamp on or off. It needs about 12 Volt on its gate to do that. The voltage divider R1/R2 should put about 13-15 volts* max on the Gate of the IGBT, switching the lamp fully ON. As there might be some fluctuations on the grid 4k7 is a safe value. If you want to be safe, make sure you have an IGBT with a Base Emmitter breakdown voltage of >= 20 Volt and put a zenerdiode of 15 V parrallel to R2. Possible IGBT's are IRGPC40W or IRG4PC30, but basically any will do provided they have a Base emmitter voltage rating of at least 20 Volts



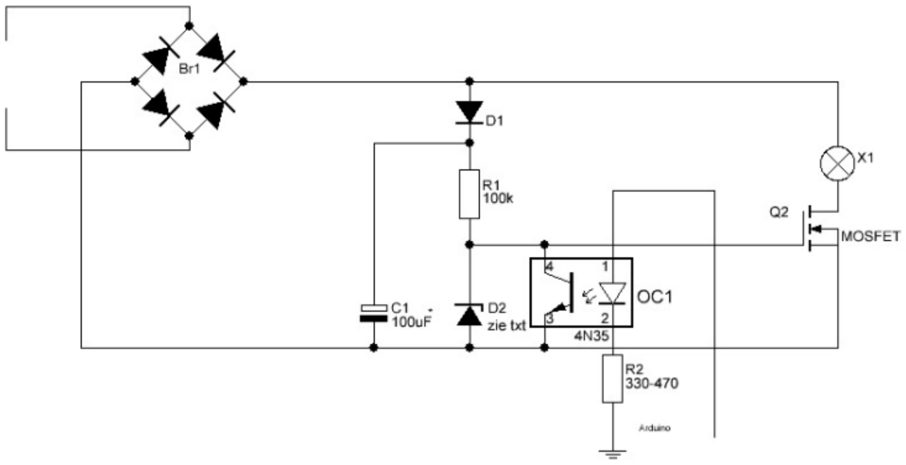
When the optocoupler receives a signal, it opens and pulls the voltage on R1/R2 to zero, effectively closing the IGBT. The PWM signal of an Arduino is faster than the 50Hz Frequency so you will basically see the PWM signal modulated on the 50Hz rectified sine wave, making the effective voltage lower.
This circuit is ONLY for incandescant bulbs. It is NOT for any inductive load as it is DC biased. With regard to the capacitor C1, I have tested it with 100uF but will probably work with lower capacity as well.

- Although the average voltage will be 230*0.9, C1 may eventually charge to 310-330 hence 4k7 is a safe value.





However, if you add the zener, you dont really need the 6k8 resistor anymore:



spectra. Using a MOSFET rather than an IGBT is therefore possible. MOSFETS are generally also cheaper than IGBT's. A tried and tested MOSFET is the STP10NK60Z (Thanks Pavel). This MOSFET has a gate-source breakdown voltage of 30 Volt and has clamping diodes protecting the the gate. MOSFETs usually need a bit of a higher voltage than IGBTs to switch so a 6k8 resistor should be fine. If you use a MOSFET without clamping diodes a zener of 15 Volt is advisable

Note: Peak and average voltages are calculated with the following formulas:

$$V_p = (V_{ac} * \sqrt{2}) - 1.4$$

$$V_{avg} = \frac{2V_p}{\pi}$$

Thus for 230 Volt V_p=323.86 Volt and V_{avg}=206 Volt

Video here

Heat development:
I tested this with a 60 Watt lamp at full brightness, without any heatsink: the temperature rose with 9 degrees above ambient after half an hour and an hour. Then I tried with continous dimming from zero to full and back again.: The temperature rose with 10 degrees above ambient after about 10-15 minutes and stayed like that for the hour I tried.

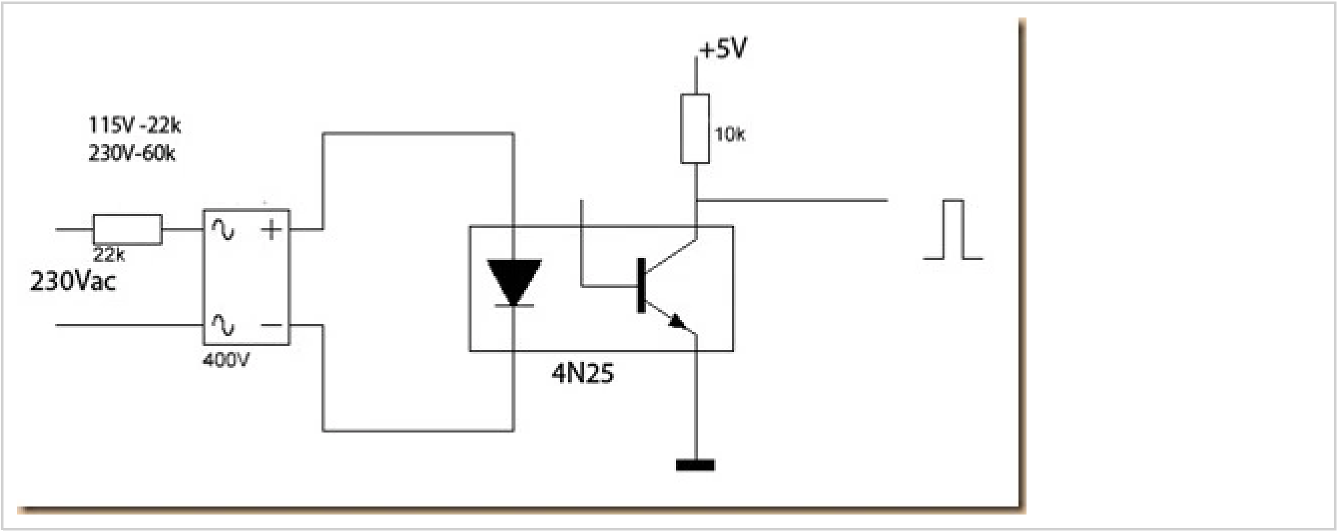
With a 150 Watt halogen spot the temperature went up 15 degrees. It reached max temperature after about 10 minutes and then stayed the same for the hour tested.

This was measured with a DHT11 sensor directly clamped to the MOSFET

July 25, 2015 / Arduino, dimmer / 128 Comments

Switching an AC load with an Arduino is rather simple: either a mechanical Relay or a solid state relay with an optically isolated Triac.
It becomes a bit more tricky if one wants to dim a mains AC lamp with an arduino: just limiting the current through a Triac is not really possible due to the large power the triac then will need to dissipate, resulting in much heat and it is also not efficient from an energy use point of view.

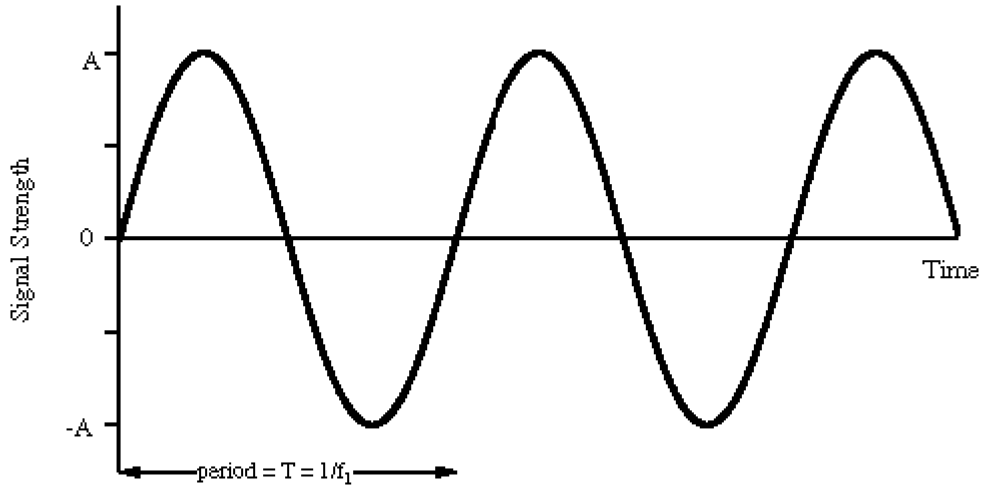
The proper way to do it is through phase control: the Triac then is fully opened, but only during a part of the sinoid AC wave.
One could an Arduino just let open the Triac for a number of microseconds, but that has the problem that it is unpredictable during what part of the sinuswave the triac opens and therefore the dimming level is unpredictable. One needs a reference point in the sinus wave.
For that a zero crossing detector is necessary. This is a circuit that tells the Arduino (or another microcontroller) when the sinus wave goes through zero and therefore gives a defined point on that sinus wave.
Opening the Triac for a number of microseconds after the zero crossing therefore gives a predictable level of dimming.

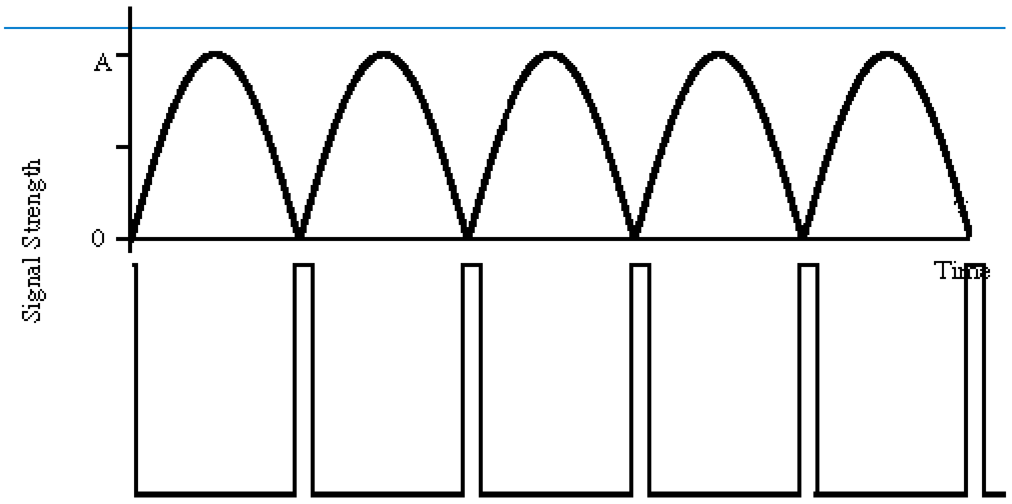


An example of a zero cross detection circuit

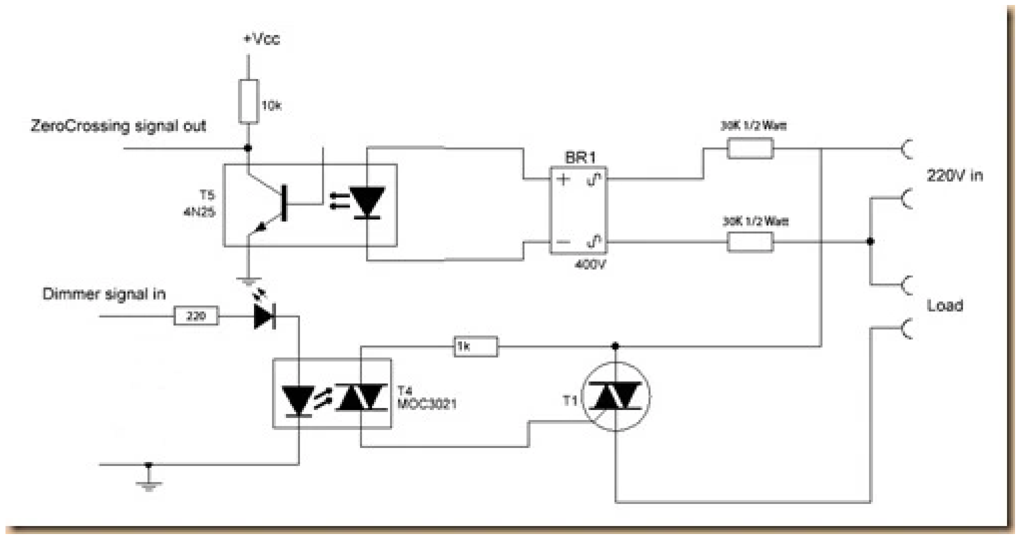
Such a circuit is easy to build: The zero crossing is directly derived from the rectified mains AC lines – via an optocoupler of course- and gives a signal every time the

the sinus wave goes up through zero or down through zero. This signal then can be used to steer an interrupt in the Arduino.





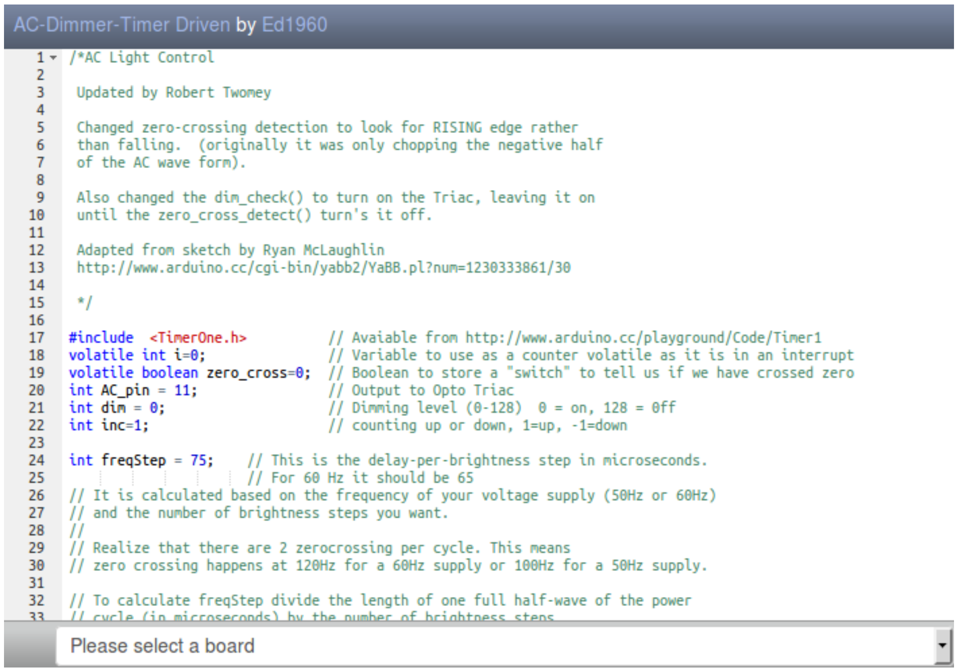
Opening the Triac is done via a tried and tested opto coupled Triac circuit. For the Triac a TIC206 can be used, but a BT136 (a bit cheaper) would probably work as well. With regard to the MOC3021: that has a forward voltage of about 1.2-1.4 Volts and a trigger current of 8-15 mA. An LED has a forward voltage of about 2Volts. Presuming a 5V digital steering signal that gives a resistor value between (5-3.2)/0.015= 120 Ohm and (5-3.2)/0.008=225 Ohm, making 220 a decent choice. In reality an LED may have a slightly lower forward voltage and a 330 Ohm resistor was found to work as well. It is important NOT to chose an opto-Triac here with an inbuilt zero-crossing filter. The MOC3041 or the MOC3061 for that matter therefore is not usable in this circuit. The MOC 3021 is. NOTE: some people have experienced flicker as a resistor to 330 or 470 Ohm With regard to the 30k resistors: The 1/2 Watt should do it as the resistors dissipate some 400mW. If you think that is too much, replace the 4N25 with a 4N33 and the two 30k resistors with two 100 k resistors. Be careful though in using sensitive optocouplers as the 4N32 as they will easily be fully in saturation.



This is the complete circuit of the dimmer

All one needs then is the software. Below a piece of demo software. Due to use of ‘delay()’ it does not work efficiently meaning that the microcontroller can’t do much else, but if you just want to dim a light that is fine.

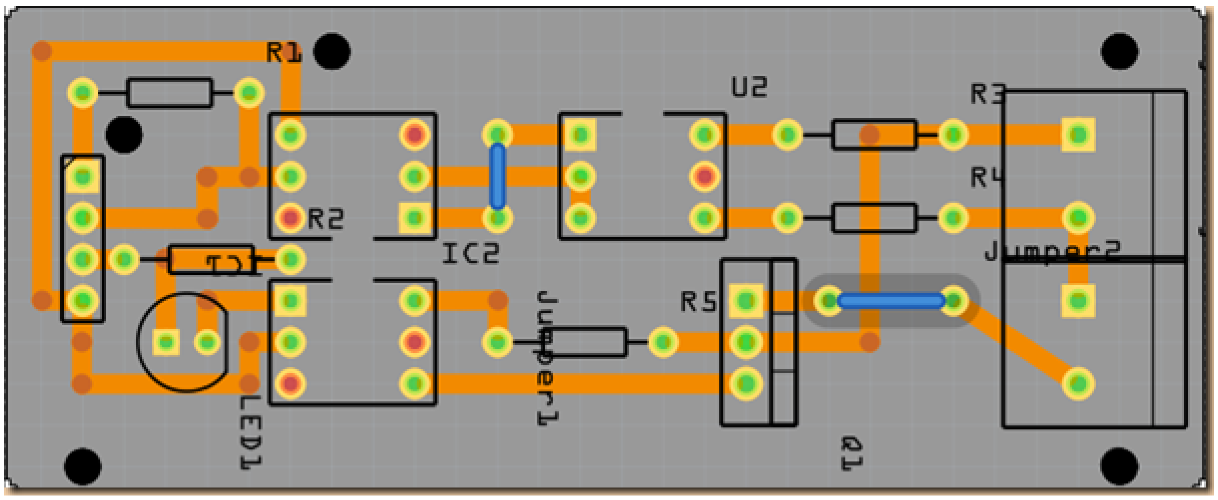
```
/*
AC Voltage dimmer with Zero cross detection
Author: Charith Fernanado http://www.inmojo.com
License: Creative Commons Attribution Share-Alike 3.0 License.

Attach the Zero cross pin of the module to Arduino External Interrupt pin
Select the correct Interrupt # from the below table:
(the Pin numbers are digital pins, NOT physical pins:
digital pin 2 [INT0]=physical pin 4
and digital pin 3 [INT1]= physical pin 5)

 Pin    |  Interrrupt # | Arduino Platform

 18     |  5            |  Arduino Mega Only
 19     |  4            |  Arduino Mega Only
 20     |  3            |  Arduino Mega Only
 21     |  2            |  Arduino Mega Only

In the program pin 2 is chosen
 */

int AC_LOAD = 3;    // Output to Opto Triac pin
int dimming = 128;  // Dimming level (0-128)  0 = ON, 128 = OFF
/* Due to timing problems, the use of ‘0’ can sometimes make the circuit
flicker. It is safer to use a value slightly higher than ‘0’
*/
void setup()
{
  pinMode(AC_LOAD, OUTPUT);// Set AC Load pin as output
  attachInterrupt(0, zero_crosss_int, RISING);
// Chooses '0' as interrupt for the zero-crossing
}
// the interrupt function must take no parameters and return nothing
void zero_crosss_int()
// function to be fired at the zero crossing to dim the light
{
  // Firing angle calculation : 1 full 50Hz wave =1/50=20ms
  // Every zerocrossing thus: (50Hz)-> 10ms (1/2 Cycle) For 60Hz => 8.33ms

  // 10ms=10000us
  // (10000us - 10us) / 128 = 75 (Approx) For 60Hz =>65
  int dimtime = (75*dimming);    // For 60Hz =>65
  delayMicroseconds(dimtime);    // Off cycle
  digitalWrite(AC_LOAD, HIGH);   // triac firing
  delayMicroseconds(10);         // triac On propogation delay
                                 //(for 60Hz use 8.33)
  digitalWrite(AC_LOAD, LOW);    // triac Off
}
void loop()  {
 for (int i=5; i <= 128; i++)
{
 dimming=i;
 delay(10);
 }
 }
```



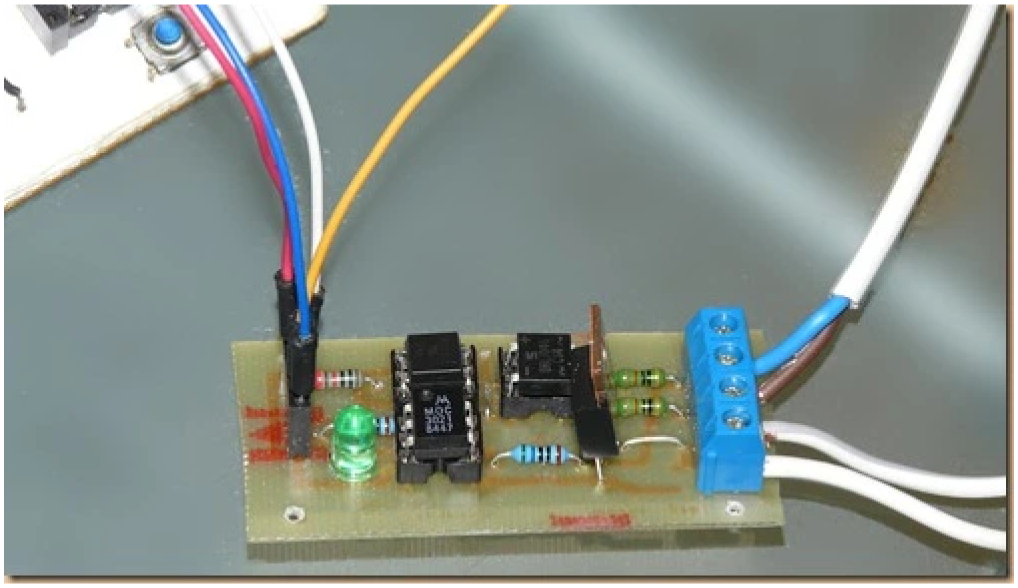
AC-Dimmer-Timer Driven by Ed1960

```
/*AC Light Control

 Updated by Robert Twomey

 Changed zero-crossing detection to look for RISING edge rather
 than falling.  (originally it was only chopping the negative half
 of the AC wave form).

 Also changed the dim_check() to turn on the Triac, leaving it on
 until the zero_cross_detect() turn's it off.

 Adapted from sketch by Ryan McLaughlin
 http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1230333861/30

 */

#include  <TimerOne.h>          // Avaiable from http://www.arduino.cc/playground/Code/Timer1
volatile int i=0;               // Variable to use as a counter volatile as it is in an interrupt
volatile boolean zero_cross=0;  // Boolean to store a "switch" to tell us if we have crossed zero
int AC_pin = 11;                // Output to Opto Triac
int dim = 0;                    // Dimming level (0-128)  0 = on, 128 = 0ff
int inc=1;                      // counting up or down, 1=up, -1=down

int freqStep = 75;    // This is the delay-per-brightness step in microseconds.
                      // For 60 Hz it should be 65
// It is calculated based on the frequency of your voltage supply (50Hz or 60Hz)
// and the number of brightness steps you want.
//
// Realize that there are 2 zerocrossing per cycle. This means
// zero crossing happens at 120Hz for a 60Hz supply or 100Hz for a 50Hz supply.

// To calculate freqStep divide the length of one full half-wave of the power
// cycle (in microseconds) by the number of brightness steps.
```

For an MQTT version see here

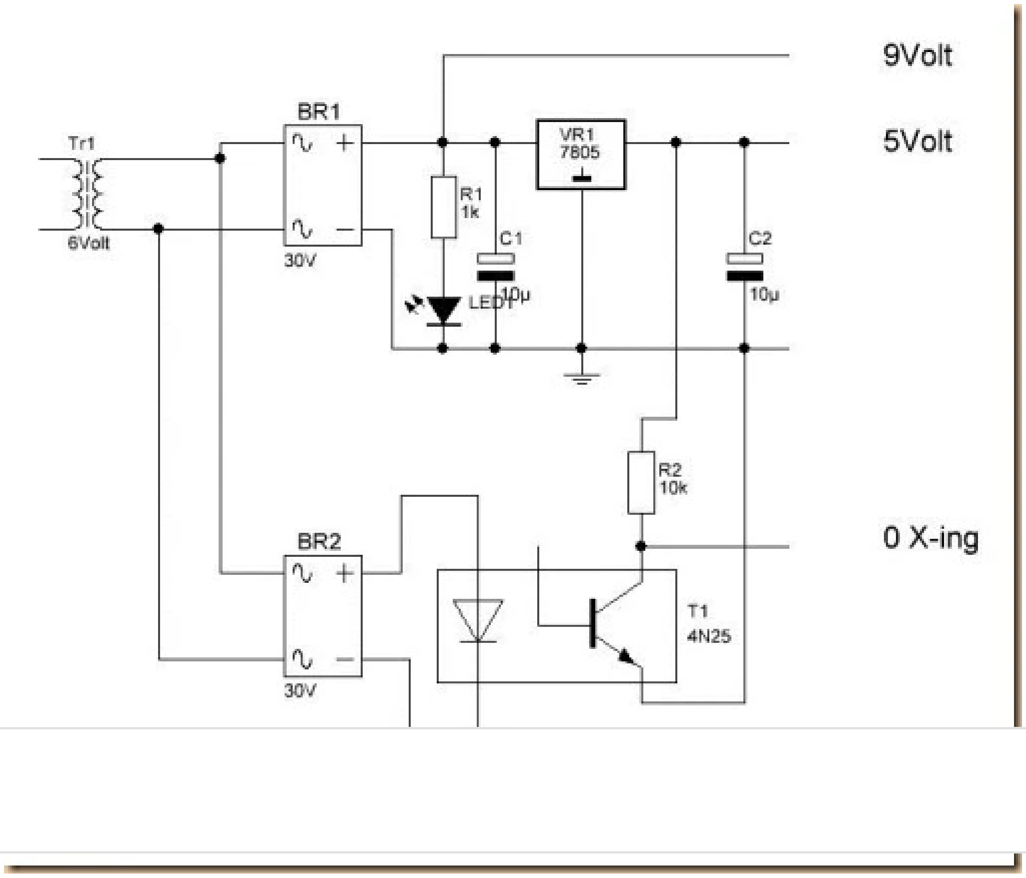


Print design. Mirrored.



On thing to consider is that if you are building a system in which you are not using a ready made PSU but rather supply your own PSU, built from a transformer and a rectifier, you could consider picking up the 0-crossing from the secondary side of the transformer.

A possible circuit could look like this:



Using the dimmer with a fan
A question I often get is if this circuit can be used with a (ceiling)fan. Well a Triac is not particularly good at regulating inductive loads and the speed of many AC-motors in fact set by the frequency of the grid and the number of poles. (rmp=120*f/p). However, the Shaded-pole motor can be regulated with a Triac and the shaded pole motor is often (but not always as the PSC motor is common too) found in fans. Check here.

October 19, 2012 / dimmer / 213 Comments

Arduino, ESP8266, ESP32 & Raspberry Pi stuff / Create a free website or blog at WordPress.com.