

Manuals and Curriculum

Arduino StackExchange

Board Setup and Configuration

Development Tools

Arduino on other Chips

Interfacing With Hardware

- Output
- Input
- User Interface
- Storage
- Communication
- Power supplies
- General

Interfacing with Software

User Code Library

- Snippets and Sketches
- Libraries
- Tutorials

Suggestions & Bugs

Electronics Technique

Sources for Electronic Parts

Related Hardware and

Initiatives

Arduino People/Groups & Sites

Exhibition

Project Ideas

Languages

Participate

- Formatting guidelines
- All recent changes
- PmWiki
- WikiSandBox training
- Basic Editing
- Documentation index

AC Phase Control

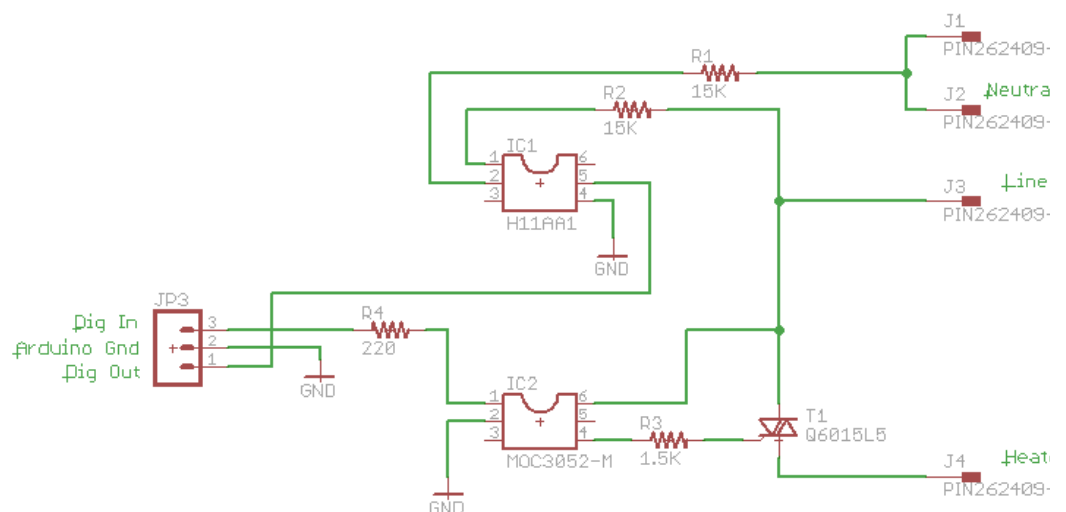
One method of controlling power to ac circuits uses a triac to turn the power on and off at precisely timed intervals that are synchronized with the ac signal. This method is called AC phase control. It is the method used in many light dimmer and heater power control circuits.

See Also

<http://playground.arduino.cc/Code/ACPhaseControl>

Circuit

Using an Arduino microcontroller with some simple circuitry, we can monitor the ac wave to determine the proper time to turn the power on and off with the triac (A 10kOhm Pull-up resistor is needed at the H11AA1 pin 5). The circuit I used to do this is shown below.

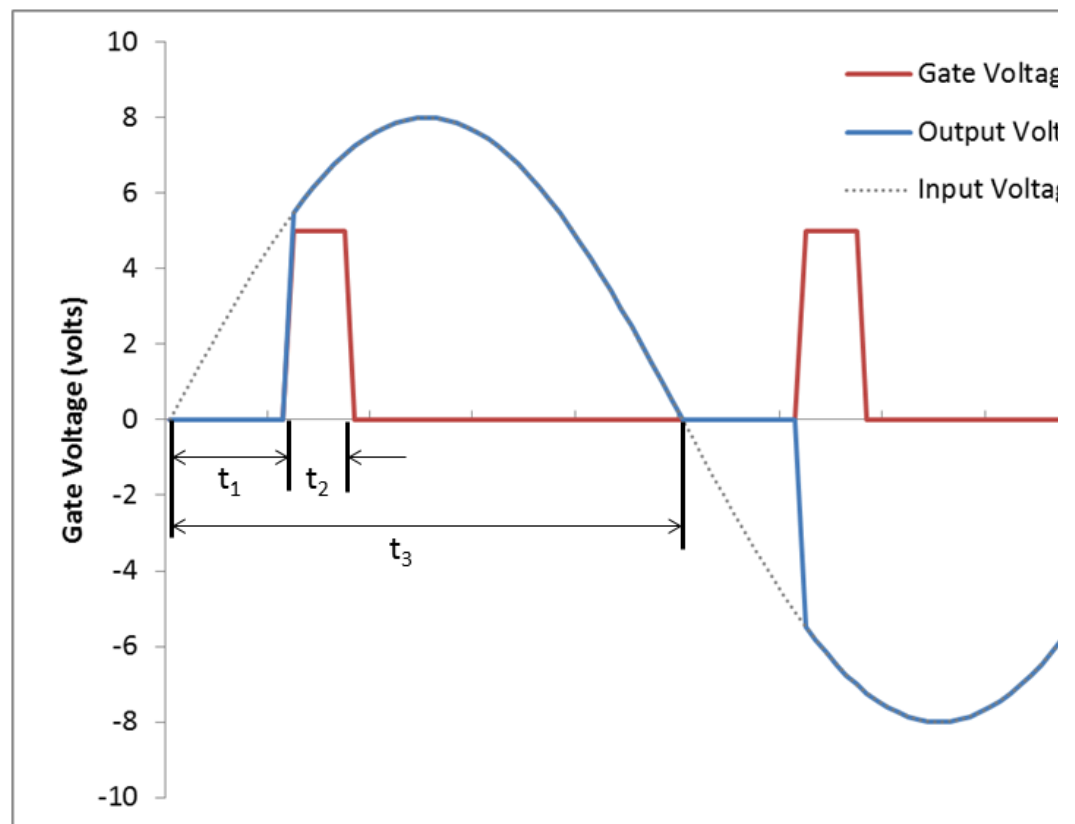


The circuit consists of an opto-isolated zero-crossing detector and a opto-isolated trigger circuit for the triac. The opto-isolators are necessary to keep the low voltage signal circuits away from the power circuits and provide an appropriate level of safety. As will all circuits involving mains voltage, make sure you know what you are doing.

Theory of Operation

The zero-crossing detection circuit provides a 5v pulse every time the ac signal crosses

zero volts. We detect this with the Arduino and leverage interrupts to time the trigger circuit precisely in synchronization with these zero-crossing events. The method for power control is shown in the diagram below.



Once a zero crossing is detected, the triac remains off for a controlled amount of time (t_1). The longer this time is, the less power the ac circuit receives. Once the "off-time", t_1 has elapsed, the microcontroller turns on the triac by applying a voltage to the gate (shown in red). Once turned on, the triac will remain on even after the gate voltage has been removed. It will turn off if the gate voltage is zero the next time the ac wave crosses zero. Because of this, we do not need to take care to turn the triac off when the ac signal crosses zero again. All we need to do is to ensure that the triac gets turned off inside of the period of $\frac{1}{2}$ wave (t_3). The duration of the gate pulse (t_2) is determined by a minimum requirement of the triac. If this pulse is too short, the triac will not fire. Once the second zero crossing occurs, since there is no voltage on the gate, the triac remains off until triggered again in the next $\frac{1}{2}$ cycle. The net result here is that we "chop" parts of the wave out resulting in lower average power. This is essentially how one accomplishes "PWM" control of an AC wave.

We will be using interrupts and the arduino timer to precisely control the timing of the triac gate. To get a feel for the time intervals, we need to look at the AC signal and the Arduino clock.

The AC signal (in the US anyway) is 60 Hz. What this means is that the AC signal crosses

zero, reaches peak positive voltage, crosses zero, reaches peak negative voltage and returns to zero 60 times each second. The period (length of time this takes) is $1/60$ or 0.01667 seconds (16.67 milliseconds). A half cycle (the time between two zero-crossings) occurs in 8.33 milliseconds. This is t_3 in the figure above.

The Arduino clock runs at 16 MHz, which is 16,000,000 cycles per second: one clock cycle takes 0.0625 microseconds. A single half cycle of the 60 Hz AC signal contains 133,333 clock cycles. This is important because we will be determining the time intervals by clock counts in the Arduino code, not by seconds.

There is quite a bit of good information on use of interrupts with the Arduino out on the web so I won't cover that in much detail here. Basically the way an interrupt works is that when some event happens (either internal or external to the microprocessor), the microprocessor immediately stops what it is doing to "service" the interrupt. This allows the microprocessor to handle very time sensitive events such as the AC Phase control task here.

Other Resources

Printed circuit board available here: <https://batchpcb.com/pcbs/93955>

HeaterControl.pde

```

// AC Control V1.1
//
// This arduino sketch is for use with the heater
// control circuit board which includes a zero
// crossing detect function and an opto-isolated triac.
//
// AC Phase control is accomplished using the internal
// hardware timer1 in the arduino
//
// Timing Sequence
// * timer is set up but disabled
// * zero crossing detected on pin 2
// * timer starts counting from zero
// * comparator set to "delay to on" value
// * counter reaches comparator value
// * comparator ISR turns on triac gate
// * counter set to overflow - pulse width
// * counter reaches overflow
// * overflow ISR turns off triac gate
// * triac stops conducting at next zero cross

// The hardware timer runs at 16MHz. Using a
// divide by 256 on the counter each count is
// 16 microseconds. 1/2 wave of a 60Hz AC signal
// is about 520 counts (8,333 microseconds).

#include <avr/io.h>
#include <avr/interrupt.h>

#define DETECT 2 //zero cross detect
#define GATE 9 //triac gate
#define PULSE 4 //trigger pulse width (counts)
int i=483;

void setup(){

    // set up pins
    pinMode(DETECT, INPUT); //zero cross detect
    digitalWrite(DETECT, HIGH); //enable pull-up resistor
    pinMode(GATE, OUTPUT); //triac gate control

    // set up Timer1
    //(see ATMEGA 328 data sheet pg 134 for more details)
    OCR1A = 100; //initialize the comparator
    TMSK1 = 0x03; //enable comparator A and overflow interrupts
    TCCR1A = 0x00; //timer control registers set for
    TCCR1B = 0x00; //normal operation, timer disabled

    // set up zero crossing interrupt
    attachInterrupt(0,zeroCrossingInterrupt, RISING);
    //IRQ0 is pin 2. Call zeroCrossingInterrupt
    //on rising signal

}

//Interrupt Service Routines

void zeroCrossingInterrupt(){ //zero cross detect
    TCCR1B=0x04; //start timer with divide by 256 input
    TCNT1 = 0; //reset timer - count from zero
}

ISR(TIMER1_COMPA_vect){ //comparator match
    digitalWrite(GATE,HIGH); //set triac gate to high
    TCNT1 = 65536-PULSE; //trigger pulse width
}

ISR(TIMER1_OVF_vect){ //timer1 overflow
    digitalWrite(GATE,LOW); //turn off triac gate
    TCCR1B = 0x00; //disable timer stopd unintended triggers
}

void loop(){ // sample code to exercise the circuit

    i--;
    OCR1A = i; //set the compare register brightness desired.
    if (i<65){i=483;}
    delay(15);

}

```

[\[Get Code\]](#)

Share



NEWSLETTER

Enter your email to sign up



©2016 Arduino Copyright Notice Contact us

