HOTBED OVERCLOCKING

Lazaros Economou

V1.0 - July 2017

Everyone who is involved in 3D printing knows very well what a tedious and time consuming process the preparation of the hotbed can be. One thing everyone would all like is faster heat up times. The basic idea – already implemented by the bravest of all ☺ - is to use a bigger and separate PSU. More volts means more Amps, more power, more energy and faster heat up times.

There are many different hotbed types and suppliers, all with varying fabrication qualities. The main concern by a lot people is whether the extra power could or would damage their hotbed..

Wouldn't it be nice if someone could use and test intermediate power values and gradually settle down to a power setting he feels comfortable with?

The following two methods are commonly used for power control.

1) PWM Modulation. Some people still won't feel comfortable claiming that this is energy control not power. Well….strictly speaking they are right, but it seems to do the trick in most cases.
2) Current Limiting circuits like the ones used in diode lasers and steppers.
3) A combination of 1 and 2. Obviously this is an overkill from an engineering point of view.

Some safety precautions:

1) Use wires that are rated for the current you plan to draw. Using smaller diameter wires not only is a serious fire hazard but it may also lead to unpredictable results due to heat and power dissipation on the wire itself.
2) YOU are the sole responsible for your safety !!!! Do not proceed unless you are 100% sure of what you are doing.

From my experience:

-Get a decent Aluminum hotbed. Normal hotbeds usually have big temperature differences between the center (which is usually the thermistor measurement point) and the corners. Aluminum is an excellent heat conductor, therefore you get an even heat distribution.

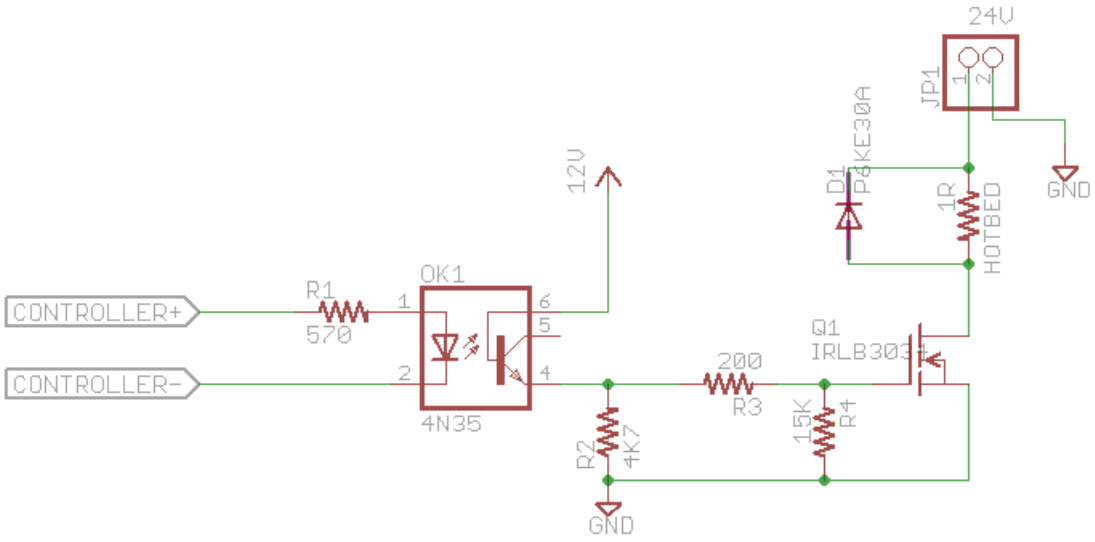
My intention is not a crash course in electronics, I assume you know what Ohms law is, basic and fundamental transistor principles and how to hold a soldering iron ☺

Finally, yes, I know you can PWM the whole thing from the firmware….but…it is not fun…not at all..

So let's get started.

Basic Hotbed Driving – No Current Control

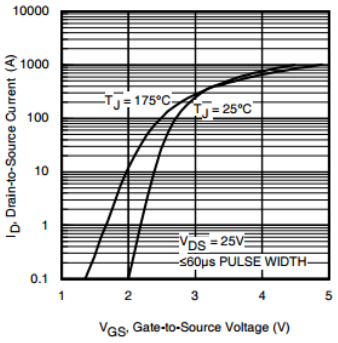
If no current control is required then something similar to the one below could be used.



Many things can go wrong when building and/or testing high current circuits. The optocoupler is there to isolate and protect the precious controller board from the bed driving circuitry. 4N35 is a very common choice but anything similar could be used.

When the input from the controller is OFF (0V) pin4 of the optocoupler is pulled to ground via R2. When it is on, it is around 12V (in fact 12V-Vce~12V). 12V is more than enough to turn on the Mosfet.

IRLB3034 is my "favorite" power Mosfet. It can carry as much as 195A (datasheet specs). Its Vgs vs Id graph is shown below, it can be turned on with as little as 3V which makes it suitable for use even with 3.3V microcontrollers. This is not the only choice for this case; most power Mosfets out in the market would happily cope with this job.



R3, R4 and D1 are optional components, however they are considered by many people as good circuit design practice.

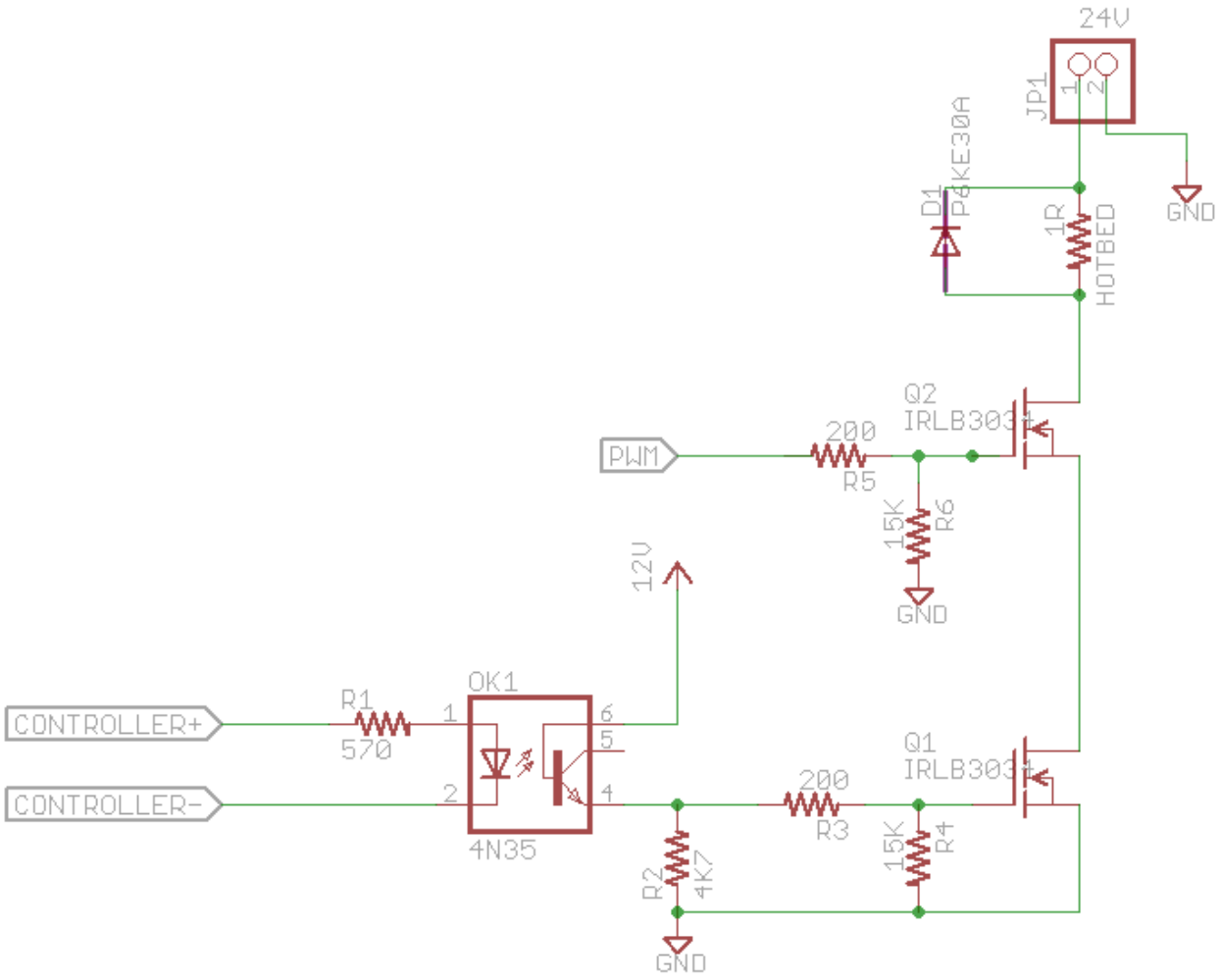
R3 is there to protect sensitive microcontroller output circuitry. Most Mosfets have a high input capacitance. This can cause sharp current transients during OFF->ON transitions (and unwanted oscillations in some cases) and damage the sensitive electronics that drive the transistor. Values of 100-200 Ohms would be just fine.

R4 is a pull down resistor. Mosfets (as opposed to BJTs) have a tendency to “float”. R4 is there to assume a predefined input state until the input to the Mosfet is stabilized (for example during the boot up time of a microcontroller). Since it also forms a voltage divider with R3, a rather large value is needed to ensure adequate Vgs voltage (depends on application and Mosfet type). Since R2 does exactly the same job (it pulls pin4 to ground) you may omit R4.

D1 is a flyback diode. Flyback diodes are commonly used to suppress current spikes when inductive loads are switched off by semiconductor devices. Someone may claim that is not strictly needed here since the hotbed is a resistive load…well yes…and no….nothing is purely resistive...is it? Any diode that is capable of handling the current/voltage spikes can be used, I personally use TVS diodes (Transient Voltage Suppression) which are specifically built for this purpose.

Method 1 - PWM

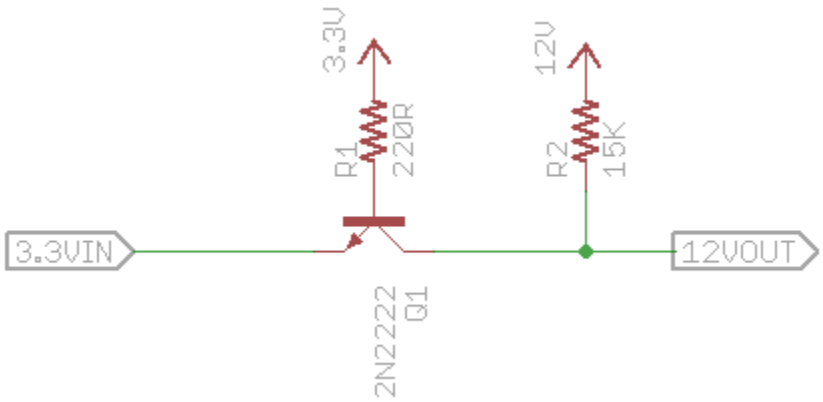
There are tons of circuits out there. Common choices are 555 timers, opamps, 74HC14, microcontrollers etc. My approach is to use an Arduino, a mini pro. More on this later. Before that we have to consider how to apply PWM. One logical approach would be to use a second Mosfet in series with the first one. Two Mosfets in series form a basic AND gate. When both the controller input and the PWM are high, both Mosfets are ON and current flows through the hotbed.



Most logic level Mosfets start to conduct with gate voltages at around 3-3.3 V. IRLB3034 for example will happily carry a 24A load with a very small gate voltage. However, although the Mosfet is ON it is not FULLY ON (saturated). SInce a small gate voltage is applied the Mosfet is not saturated, it operates in its linear or resistive region. As the name implies, in the resistive region Mosfets are..well…resistive and they can get hot…really hot. Most common PWM circuit designs will produce a wave with amplitude of around 5V. A 24A load is more than sufficient on its own to produce a descent amount of heat. PWMing that load in the resistive region can only make things worse. There are a couple of things you can do to compensate:

a) A heatsink and a fan will provide descent cooling.
b) A level shifter may be used to convert a 5V PWM to a 12V one.

Level shifters are relatively simple circuits to make.

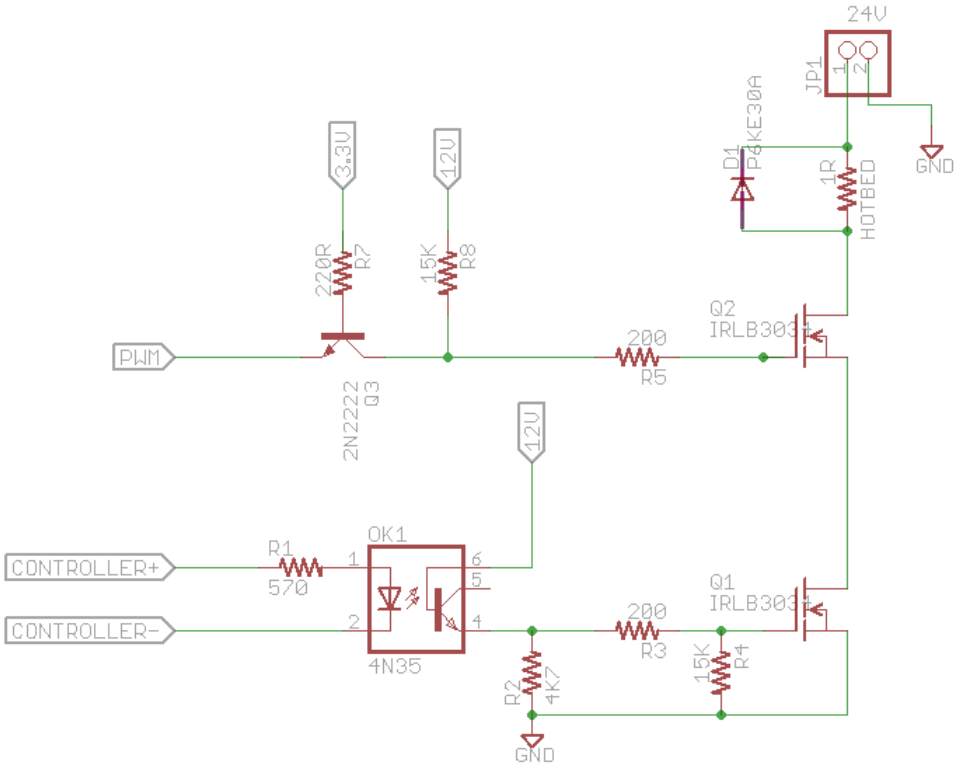


When input is 3.3V Vbe is OFF, transistor is OFF and output is pulled up to 12V via R2.

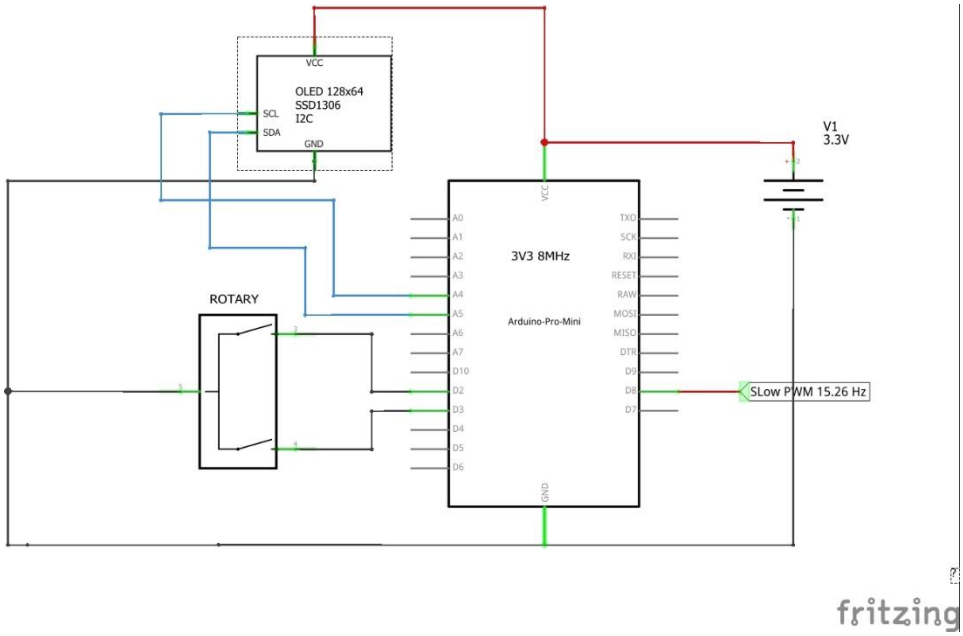
When input is 0V Vbe is ON, transistor is ON and output is “pulled” to a few millivolts ~GND.

For the lazy ones, Ebay is the right place to get one for a couple of euros.

All level shifters (BJT and Mosfet ones) work on the same pull-up (R2) principle. So if one is used , R6 (pull down) should be removed otherwise R6 will form a nice voltage divider with the pull up resistor R2 and less than 12V will end up at the Mosfet’s gate…no good done.



The next step is the actual PWM generation. An Arduino mini pro (3.3V 8MHz version) was used for PWM, a rotary encoder for controlling the duty cycle and a 1.3" OLED display. (the 8MHz clock frequency is irrelevant, it just happened to have one around)



The tricky bit here is to accurately control the rotary encoder (it is not that trivial). The good news is that there are some smart guys out there that they have already done all the hard work for you. One of them is SimonM83, you can find his excellent work here:

http://www.instructables.com/id/Improved-Arduino-Rotary-Encoder-Reading/

What follows is Simon's code with a few extra bits for displaying the duty cycle value to the OLED.

```
#include <U8glib.h>
#include <string.h>

int PWMPin = 9; //15.26
int pinA = 2; // Our first hardware interrupt pin is digital pin 2
int pinB = 3; // Our second hardware interrupt pin is digital pin 3

volatile byte aFlag = 0; // let's us know when we're expecting a rising edge on pinA to signal that the encoder has arrived at a detent

volatile byte bFlag = 0; // let's us know when we're expecting a rising edge on pinB to signal that the encoder has arrived at a detent (opposite direction to when aFlag is set)

volatile int encoderPos = 100; //this variable stores our current value of encoder position. Change to int or uin16_t instead of byte if you want to record a larger range than 0-255

volatile int oldEncPos = 0; //stores the last encoder position value so we can compare to the current reading and see if it has changed (so we know when to print to the serial monitor)
volatile int reading = 0; //somewhere to store the direct values we read from our interrupt pins before checking to see if we have moved a whole detent

U8GLIB_SH1106_128X64 u8g(U8G_I2C_OPT_NONE); // Initialise OLED Display

void setup()
{
    u8g.setFont(u8g_font_fur30);
    pinMode(pinA, INPUT_PULLUP); // set pinA as an input, pulled HIGH to the logic voltage (5V or 3.3V for most cases)
    pinMode(pinB, INPUT_PULLUP); // set pinB as an input, pulled HIGH to the logic voltage (5V or 3.3V for most cases)
    attachInterrupt(0,PinA,RISING); // set an interrupt on PinA, looking for a rising edge signal and executing the "PinA" Interrupt Service Routine (below)
    attachInterrupt(1,PinB,RISING); // set an interrupt on PinB, looking for a rising edge signal and executing the "PinB" Interrupt Service Routine (below)
    TCCR1B = TCCR1B & 0b11111000 |0x05;
    analogWrite(PWMPin, encoderPos*255/100);
    draw(encoderPos);
}

void loop()
{
    readRotary();
}

void draw(int s)
{
      int strW,strH;
      enum {BufSize=9};
      char x[BufSize];
      snprintf (x, BufSize, "%i %%", s);
      strH = u8g.getFontAscent();
      u8g.firstPage();

      strW = u8g.getStrWidth(x);
      do
      {
          u8g.drawStr((128-strW)/2, 32+strH/2,x);
      } while( u8g.nextPage() );
}
```

```
void PinA()
{
  cli(); //stop interrupts happening before we read pin values
  reading = PIND & 0xC; // read all eight pin values then strip away all but pinA and pinB's values
  if(reading == B00001100 && aFlag)
  { //check that we have both pins at detent (HIGH) and that we are expecting detent on this pin's rising
edge
    if (encoderPos>5)
    {
          encoderPos = encoderPos - 5; //decrement the encoder's position count
          analogWrite(PWMPin, encoderPos*255/100);
          draw(encoderPos);
    }
    bFlag = 0; //reset flags for the next turn
    aFlag = 0; //reset flags for the next turn
  }
  else if (reading == B00000100) bFlag = 1; //signal that we're expecting pinB to signal the
transition to detent from free rotation
  sei(); //restart interrupts
}

void PinB()
{
  cli(); //stop interrupts happening before we read pin values
  reading = PIND & 0xC; //read all eight pin values then strip away all but pinA and pinB's values
  if (reading == B00001100 && bFlag)
  { //check that we have both pins at detent (HIGH) and that we are expecting detent on this pin's rising
edge
    if (encoderPos<100)
    {
          encoderPos = encoderPos+5; //increment the encoder's position count
          analogWrite(PWMPin, encoderPos*255/100);
          draw(encoderPos);
    }
    bFlag = 0; //reset flags for the next turn
    aFlag = 0; //reset flags for the next turn
  }
  else if (reading == B00001000) aFlag = 1; //signal that we're expecting pinA to signal the
transition to detent from free rotation
  sei(); //restart interrupts
}

void readRotary()
{
  if(oldEncPos != encoderPos)
  {
        oldEncPos = encoderPos;
  }
}
```

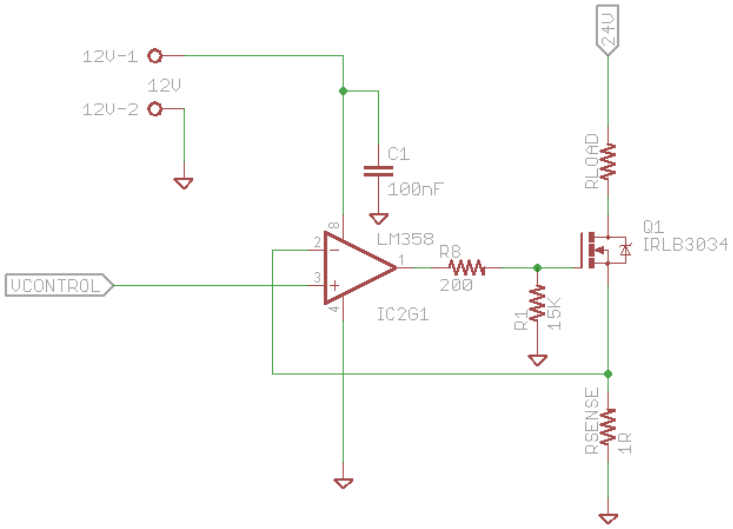
The native PWM frequency on PIN 9 for a 16MHz UNO is 31250 Hz. For an 8MHz version this comes down to 15625 Hz (31250/2). There are prescalers available for further reducing PWM frequency. This is what `TCCR1B = TCCR1B & 0b11111000 |0x05` line does in the code above. It sets a 1024 prescaler so final PWM frequency is 15.25 Hz.

- Pins 5 and 6 are paired on timer0
- Pins 9 and 10 are paired on timer1
- Pins 3 and 11 are paired on timer2

Changes on pins 3, 5, 6, or 11 may cause the delay() and millis() functions to stop working. Other timing related functions may also be affected. Changes on pins 9 or 10 will cause the Servo library to function incorrectly. Since no servo related functions/timers are used in the code, Pin 9 was chosen for PWM and prescaling was applied. However, there is no real need for prescaling. It is only mentioned (and left in the code) for educational purposes and in case fine tuning is desired.

Method 2 – Current Adjustment

Mosfets are voltage controlled resistors. The amount of current flow from Drain to Source depends on the voltage between the Gate and the Source (see the graph a few pages above). In theory, by adjusting Vgs the max current through the transistor is limited. In theory….In practice, as current flows, transistor gets hotter, Rds drops, more current flows, more heat and so on. In real life different Vgs/Id characteristics exist for different junction temperatures. What is needed here is feedback control that will compensate for temperature changes and keep the current flow steady. The circuit below is the most common one used when current control is required.



The principle of operation is rather simple:

The input voltage at the negative input of the op-amp is equal to I_{LOAD} x $R_{SENSE.}$ An op-amp is by nature happy when both its inputs are equal and it will do anything under its power to do so. Therefore it will alter its output voltage (=drive voltage to MOSFET gate) until

$V_{control}$ = I_{LOAD} x R_{SENSE} i.e. I_{LOAD} = $V_{control}$ / R_{SENSE}

If for example R_{SENSE} is chosen to be 1Ω then the transfer function between I_{LOAD} and $V_{control}$ is:

I_{LOAD} = $V_{control}$

If 3A load current is needed then the voltage at the the op-amp's input ($V_{control}$) should be 3V.

There are two main advantages:

-There is a linear relation between I_{LOAD} and $V_{CONTROL}$

- I_{LOAD} is constant and not affected by temperature changes in Mosfet.

However there are also disadvantages:

a) The power dissipation on R_{SENSE}

If for example the required current is 3A and R_{SENSE} is 1Ω, then R_{SENSE} should be capable of safely dissipating 9W (the power dissipation on a resistor is equal to I^2xR)

b) The input and output voltages of an op-amp cannot exceed its supply rail voltage

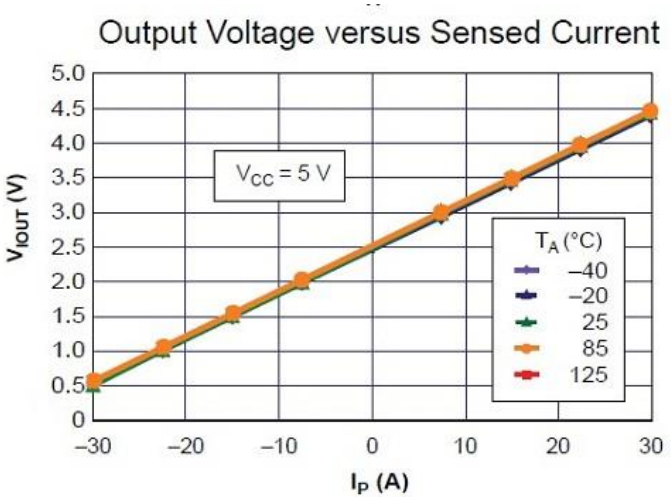
Both the above may look ok for relatively small currents, however when it comes to large currents as in the case of a hotbed, both those factors become significant. To simplify calculations let's just assume that the required hotbed current is 20A. If R_{SENSE} = 1Ω it should be capable of dissipating I^2xR i.e 400 Watts !!!!!! plus the fact that the op-amp to be used should happily operate with 20V on its input (remember that $I_{LOAD} = V_{CONTROL} / R_{SENSE}$ and since I_{LOAD} is 20A then $V_{CONTROL}$ should be 20V for an R_{SENSE} of 1Ω). If R_{SENSE} is decreased to 0.25Ω then the required power dissipation comes down to 100W and the maximum voltage at the op-amp's input 5V. Getting hold of such resistor might be tricky and in many cases expensive.

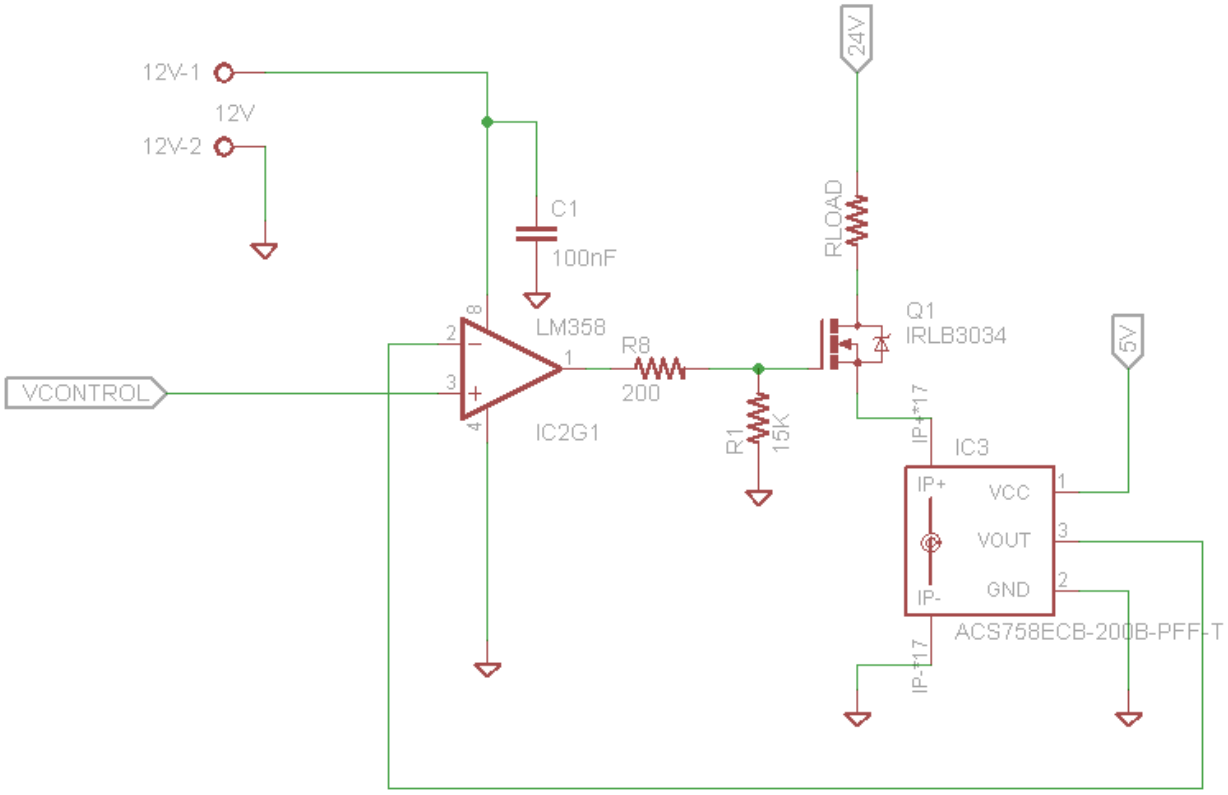
Since the load is resistive and has roughly the same resistance as $R_{SENSE,}$ the later can be eliminated and R_{LOAD} could be used in its place. For a hotbed resistance of 1.3Ω the transfer function becomes

$I_{LOAD} = 0.77 \times V_{CONTROL}$

Power dissipation is not the major issue here but for a max current of 20A the op-amp should be capable of handling 20A/0.77 = 26 Volts. This implies painful market search in terms of op-amp specs as well as non-standard supply voltages. Hopefully the famous ACS712 Hall Effect Current Sensor can be called to the rescue here. Ebay is again the place to look for, there are hundreds of them for a couple of euros. There are different current ratings available, the required one in this case is 30A.

What this lovely device does, it produces a voltage that is directly proportional to its input current. For a 5V power supply, the output is 4.5V for 30A and 0.5V for -30A. When no current flows the output is equal to Vcc/2 = 2.5V. For the 30A version the transfer function is 66mV per A. Since the output for 0A current flow is Vcc/2 then for a 5V power supply the actual output is not 66mv/A but **2.5V+**0.066V/A

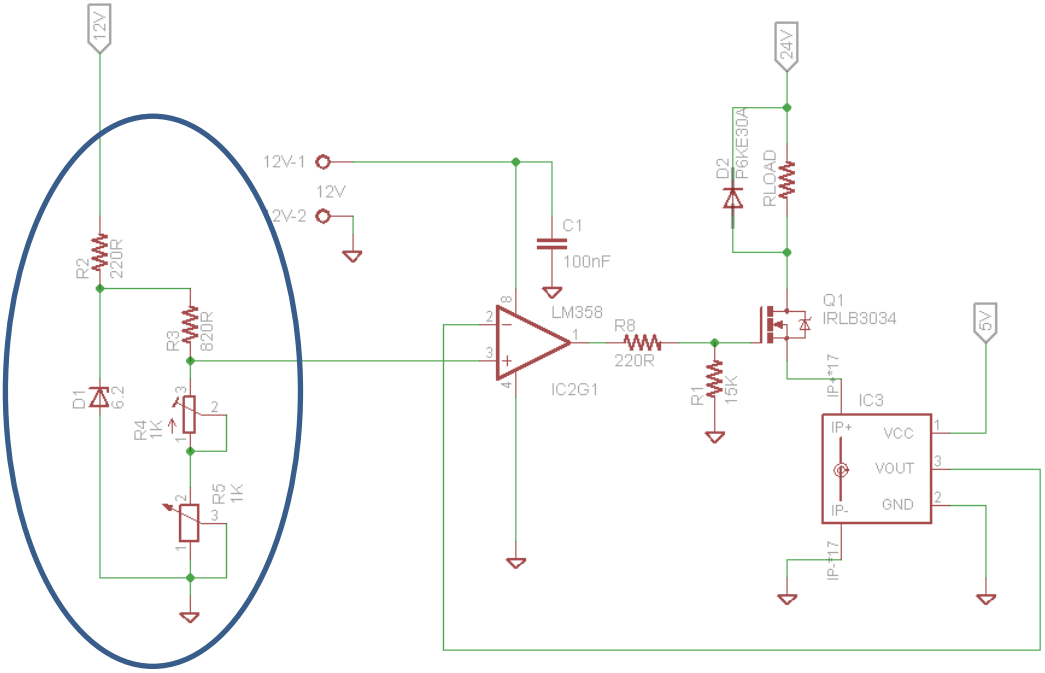




For 13A hotbed current $V_{CONTROL}$ should be 13A*0.066V/A+2.5V = 3.358 V

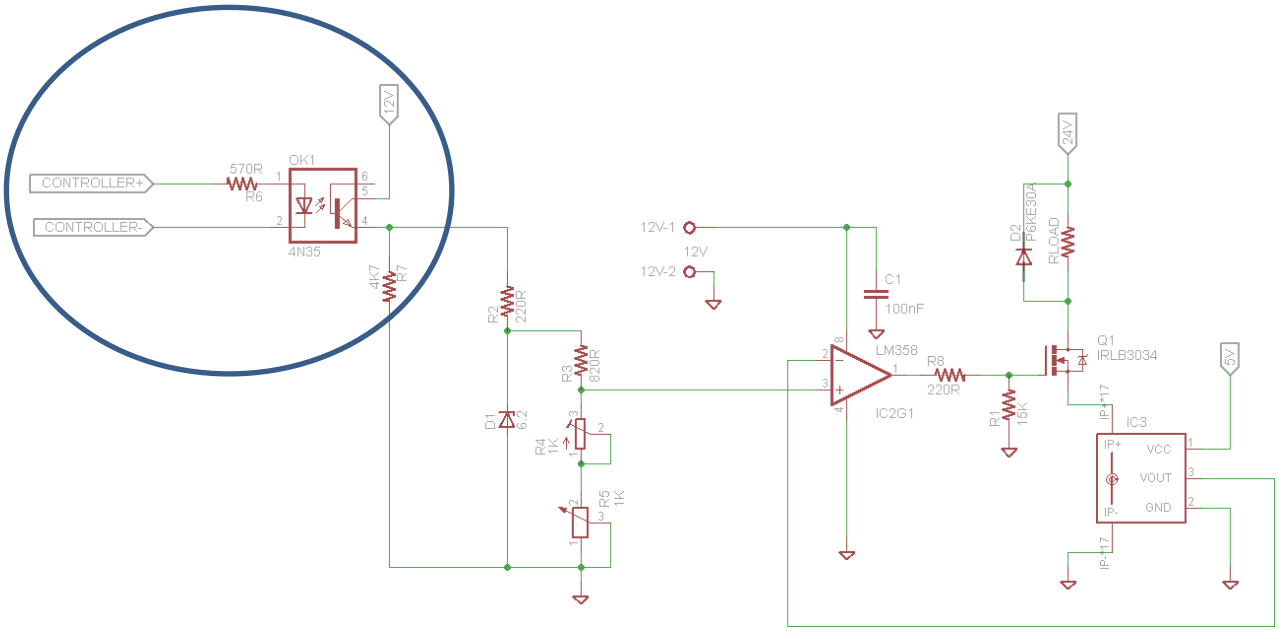
For 24A hotbed current $V_{CONTROL}$ should be 24A*0.066V/A+2.5V = 4.08 V

To summarize, for a current range 0A-24A the corresponding $V_{CONTROL}$ range is from 2.5V to 4V. The following circuit addition does exactly that. Current adjustment is done through R5.

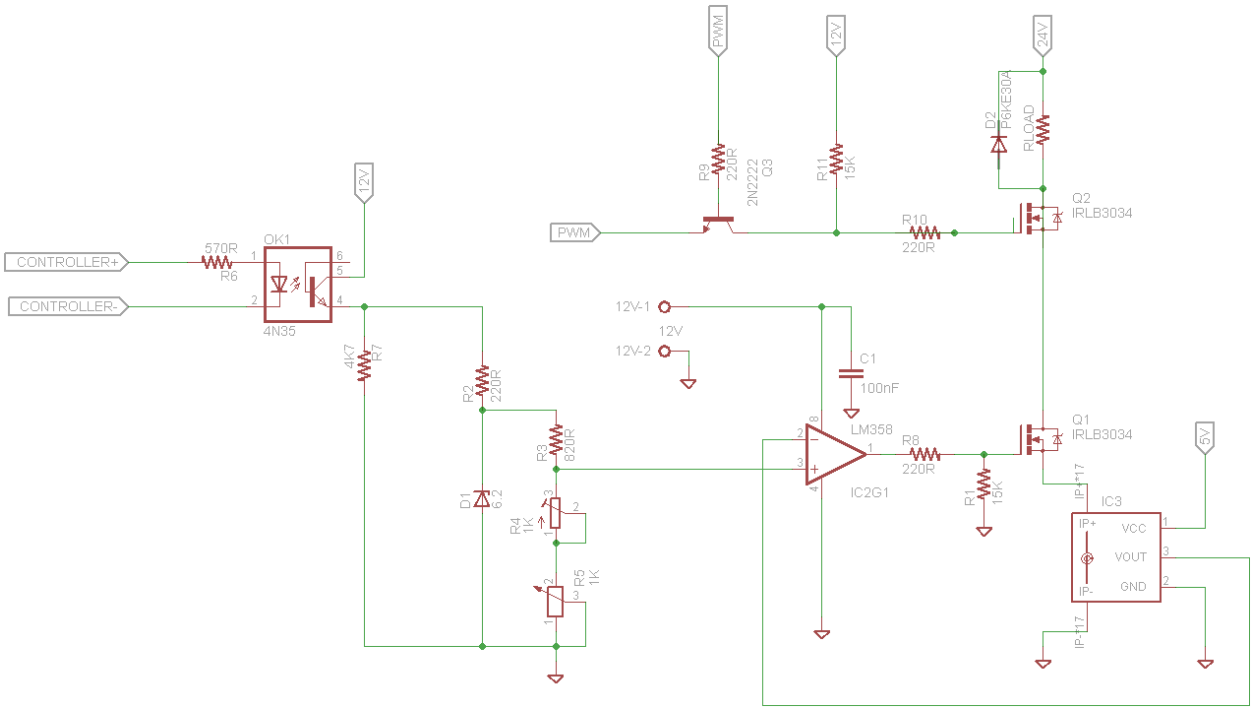


Zener D1 is there to provide a constant voltage reference of 6.2 V. R4 is a multi-turn 1K trimmer needed for an accurate 0A reference of 2.5V. For proper calibration R5 needs to be temporarily removed and pin 1 of R4 connected to ground. When R4 is around 550-560 Ohm the input to pin 3 of the op-amp should be 2.5V. R5 can then be reconnected to the circuit. By fully sweeping R5 the voltage at pin 3 varies between 2.5V and 4V which means 0A to 24A through the Mosfet.

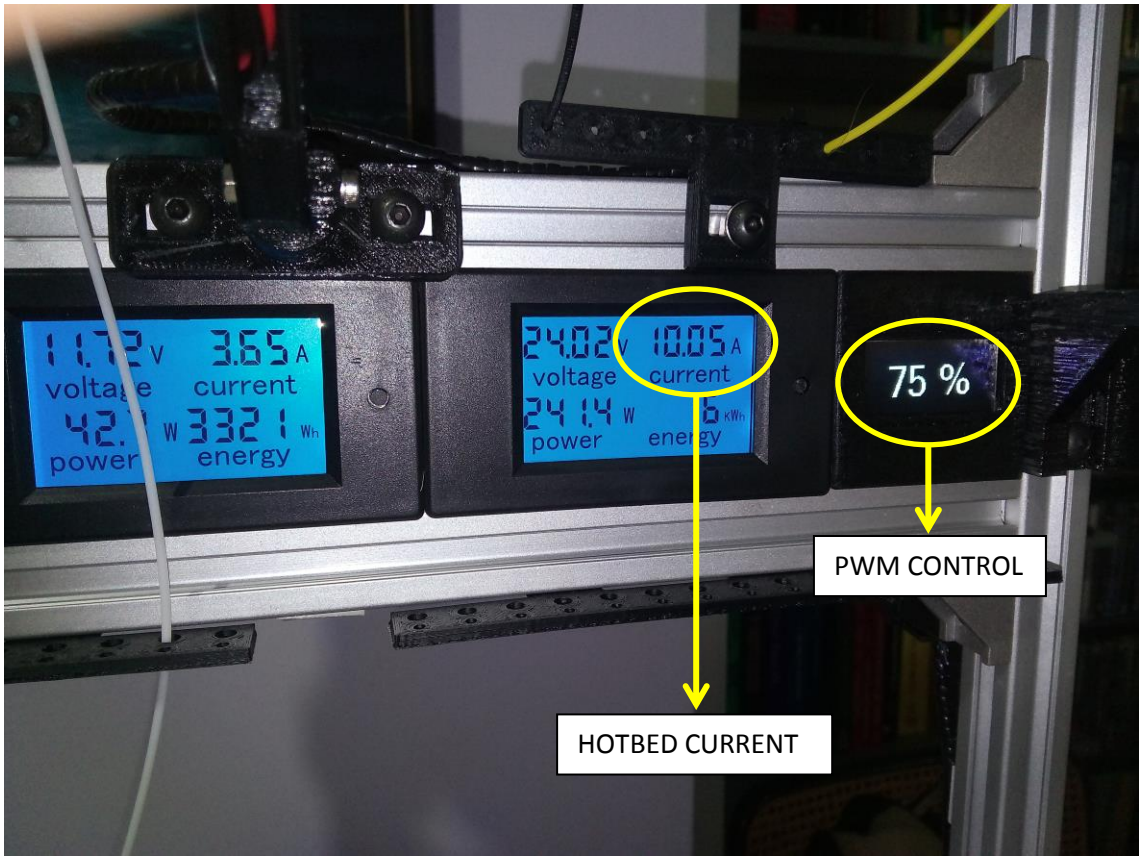
Cooling is again very important here. In this case the Mosfet is well in its resistive region and there is no trick to get away with it (like the level shifter in the PWM case). As far as the op-amp is concerned an LM358 was used. Most single rail supply op-amps will do the job. The final step is to interface the controller.



It is a bit of over-engineering but you could always put both methods together



Finally a couple of screenshots from my printer:



Happy printing!!!!