Grüzi YouTubers. Here is the guy with the Swiss accent. With a new episode around sensors and microcontrollers.

In video #131 we pimped our Raspi. One of the new features was a cooling fan which switched automatically on if CPU temperature was too high. Viewers suggested to make the speed of the fan variable. And this is, what we will do today. We also will learn a few things about Pulse Width Modulation, PID controllers, MOSFET transistors, and even a little of magic.

So, let’s start: in video #131 I used a simple concept: A MOSFET transistor to switch the fan on and off and a Python program which measures the CPU temperature and switched the MOSFET on, when the temperature was too high. Very simple. I used a beefy IRLZ44N transistor, because it is logic level. In video #137 I showed, that these FETs switch on at a voltage of below 2.5 volt. This is, why they are ideal for microprocessors with 3.3 volt logic.

First, let’s do some tests on the breadboard: I used the same diagram as in the last video and we see, that the fan starts to run full speed if we connect the gate of the transistor to 3.3 volt and it is switches off if we connect the gate to ground.

I connect now a button between 3.3 volt and the gate to switch the voltage. As soon as I push the switch, the fan starts to run. Unfortunately, the fan continues to run even if I open the switch. Not what we expected. Right?

So, lets measure the gate voltage to check, if I made a wiring mistake. If we look at the oscilloscope, we see, that the gate voltage is zero volts. But we also see, that, as soon as I connected the probe, the fan stopped. Magic?

Let’s try again. And this time, I leave the probe connected, because we want to «see» the truth. The voltage really switches between zero and 3.3 volt. But the fan now behaves differently: It does exactly, what we want. It switches on and off. Great. But what happened? The explanation is somehow hidden. But we get a clue, if we look very close to this curve: The signal is not a clean rectangle: When we open the contact, it does not go immediately to zero, it has a curve, which is typical for the discharge of a capacitor. Maybe, there is somewhere a capacitor? Yes, there is a capacitor between the gate of a MOSFET and ground. And if we leave the gate open, this capacitor stays charged, the gate voltage stays high, and the transistor stays on.

What happened when I connected the probe? Each probe of an oscilloscope has an input resistance. As soon as I connected the probe to the gate, this capacitor was discharged and the transistor switched off. BTW: This behavior is completely different compared with a bipolar NPN transistor and is typical for a FET.

To prove my point, I replace the FET transistor with a 1 nF capacitor. And you see, its behavior is very similar. Of course, it does not switch the fan… And if we have a look into the data sheet of the transistor, we find the value of this capacitance, too.

So, best practice is to connect a resistor between gate and ground, in parallel to the capacitance. In our case, the Raspi GPIO has a quite low resistance. So, this resistor is already “built-in” and we do not need to add an additional one.

For the next step, we have to know the maximum speed of the fan. It is about 7800 RPM at 5 volt.

As we saw, I can switch the fan on and off. If I do this fast, then the fan cannot get to its full speed anymore. And it does not have time to stop completely. Some of you know, that I still know the Morse code, but this switching is too exhausting even for me. So, I replace my finger and connect a waveform generator to the gate. It switches the voltage every 0.5 seconds on and off. What happens? The RPM starts to fluctuate and the meter shows around 4000 RPM. If I reduce the percentage of the on-time to 20%, the fan has the time to go to a complete stop and the meter shows around 1000 RPM, obviously with a lot of fluctuation. It is easy to reduce this fluctuation: We just increase the switching frequency of the signal to let’s say 10 Hz. Now, there is no more fluctuation and the meter shows a more or less constant 1000 RPM. If I increase the duty cycle to 50% again, we get 4000 RPM, which is around 50% of the maximum 7800 RPM. Very good. With this system, we obviously can vary the speed of the fan. And we can use a digital 3.3 volt signal to do so. Which fit’s exactly the possibilities of our Raspi.

If we increase the frequency even more, nothing special happens. Only if we zoom in to the edge of the curve, we see, that the voltage goes higher than 5 volt. In our situation, this is not dangerous, because our MOSFET can handle that. But in other situations, where we switch coils, this voltage can get much higher, and can even destroy the transistor. Fortunately, there is a simple trick available: We connect a diode, preferably a fast Schottky type, in parallel to the coil (or in our case, fan). And now, this overshooting is gone.

Another thing I have to mention if we drive MOSFETs with PWM signals: Charging and discharging the gate capacitor draws current from the driving pin. The higher the frequency, the higher the average current, because a capacitor behaves like a short circuit at high frequencies. This is, why you have to add a protection resistor between the GPIO and the gate. This resistor limits the current. Today, we only use very low frequencies. So, this is not too dangerous for our Raspi and I did not implement one.

Now, our hardware is ready. Let’s go to the software. In video #131 we already had a software which started automatically at boot-up, reads the temperature, and switches the fan. So, we use exactly this software and add a few lines.

First, we have to be able to generate a pulsed signal. This is also called PWM, or pulse width modulated signal. Fortunately, the Raspis can create such a signal at their GPIO pins. We use Python for our program to control the fan. In Python, the command to enable PWM on a pin is:

myPWM=GPIO.PWM(fanPin,50) where fanPin is the pin number, and 50 is the frequency. As we found out, the frequency is not so important for our application. That’s why I choose 50 Hertz.

Now, we can start the pin with the command myPWM.start(50), where 50 is the duty cycle. If we later want to change the duty cycle, we have to use the command myPWM.ChangeDutyCycle(fanSpeed), where, fanSpeed is the variable for the duty cycle.

Now we have nearly everything necessary to fulfill our task. The only missing part is how to determine how fast the fan should run. Let’s assume, we want a stable temperature of 45 degrees. Then, we could measure the actual temperature and calculate the difference. Let’s assume, we measure 55 degrees. Then, the difference is 10 degrees, and let’s assume, we want full speed at this temperature. Then, we would have the following formula:

fanSpeed (in %) = 10 \* (actualTemp – desiredTemp)

Now, let’s assume the temperature difference gets smaller and smaller. Then, the speed of the fan gets smaller and smaller. But unfortunately, the temperature will never exactly reach the 45 degrees, also, because the fan will stop at about 10% duty cycle.

This regulator is called p regulator because it is proportional. If we want to get a more precise control, we have to add an I part, an integration part. To achieve this, we sum up all differences between the desired and the actual temperature. If our fan stops at let’s say 46 degrees, and we add the differences every second, this value adds up quickly and the fan starts to run, even with this small temperature difference. This is, why I also implemented an I part in my program.

If you watch my video #28, you see the term “PID controller”. So, there is one letter missing: The D or differential part. This part is used to get a faster reaction. This is important for self-balancing systems, for example. With our fan, speed of regulation is not so important. This is, why I did not implement a D part. So, our controller is only called “PI controller”.

Guys, who followed the instructions from video #131 and already have a fan implemented just need to get the improved program. All the others have to watch video #131 and follow the instructions there.

Finally, we check, if everything works. To do that, I switch the program off and the CPU temperature rises to 65 degrees. As soon as we start the program, the fan starts to run with 100% speed. The temperature starts to decrease, and the fan also decreases its speed till we reach 45 degrees. Then, the temperature oscillates around our desired temperature, and the fan needs only 30 to 40% of its power to keep the temperature at this CPU load. It would immediately adapt its power if the CPU load would change. So, we now have a quiet fan and a reasonable CPU temperature. I think, this is really “cool”.

So, summarized, we know now how to use PWM on a Raspi and we also know how to properly drive a Mosfet transistor. And we programmed a simple PI controller. Fortunately, there was no Magic, just a lack of understanding. As it is quite often in life

I hope, this video was useful or at least interesting for you. If true then like. Bye