WIFI Controllable Textile Heating Circuit
DAAN

This prototype demonstrates how to control a high power heating circuit with MOSFET's operated by a ESP2866. The circuit can therefor also work for other applications where higher power is needed while the regulating microcontroller can still run on low power. While building the circuit, we had to experiment with different components and if this circuit gets reproduced either the exact use of parts or good reading the Datasheets of the similar components is advised. The unbalanced heating of just one heating element is possible but not advised for a longer period of time.

Max Current : 3,2 A
Max Voltage: 11,5 Volt
Regulated Voltage: 5 Volt

List of parts:

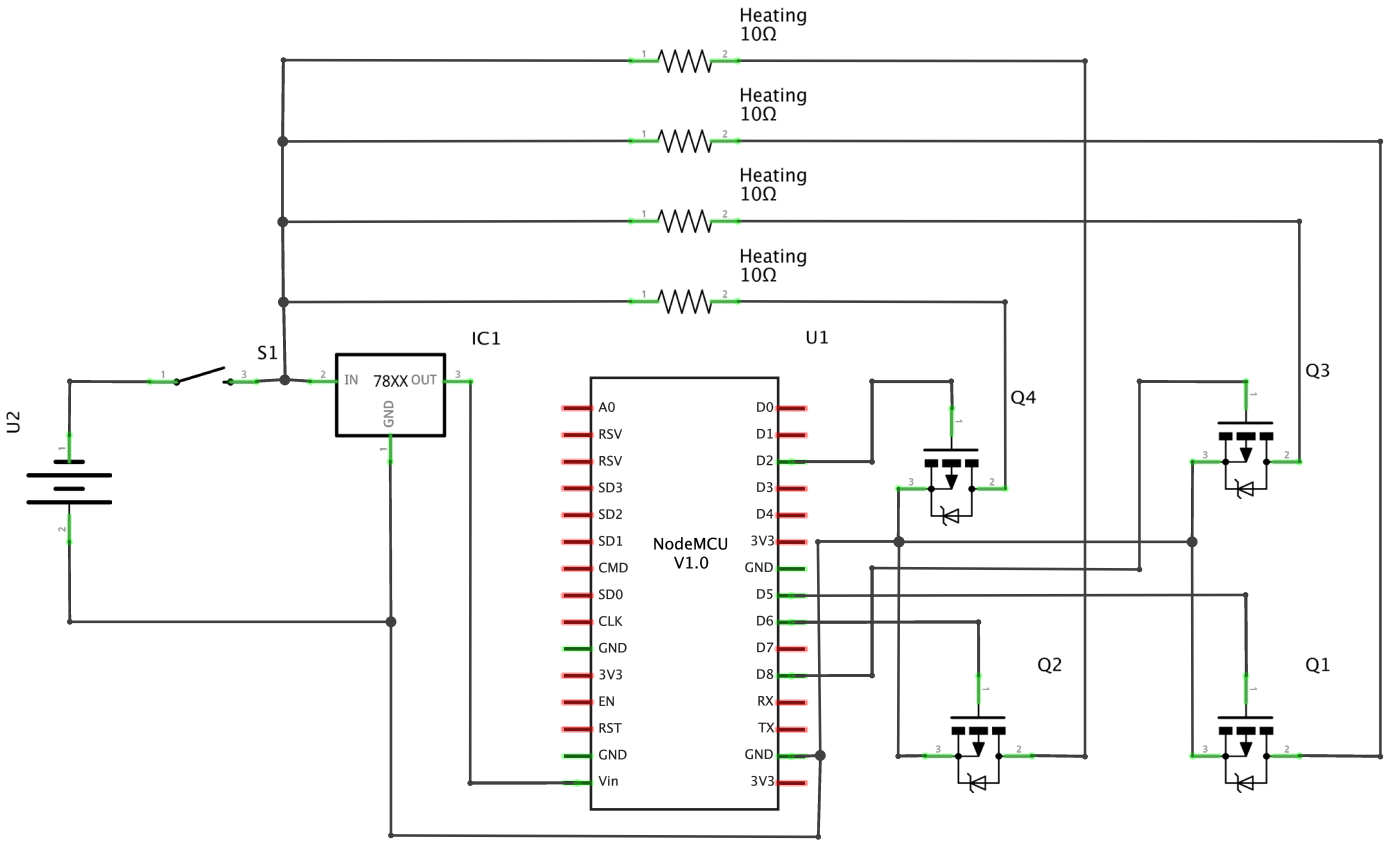
Voltage Regulator:
LM1084IT-5.0

MOSFET:
MOSFET Infineon Technologies IRLZ34NSPBF
1 HEXFET 3.8 W D2PAK

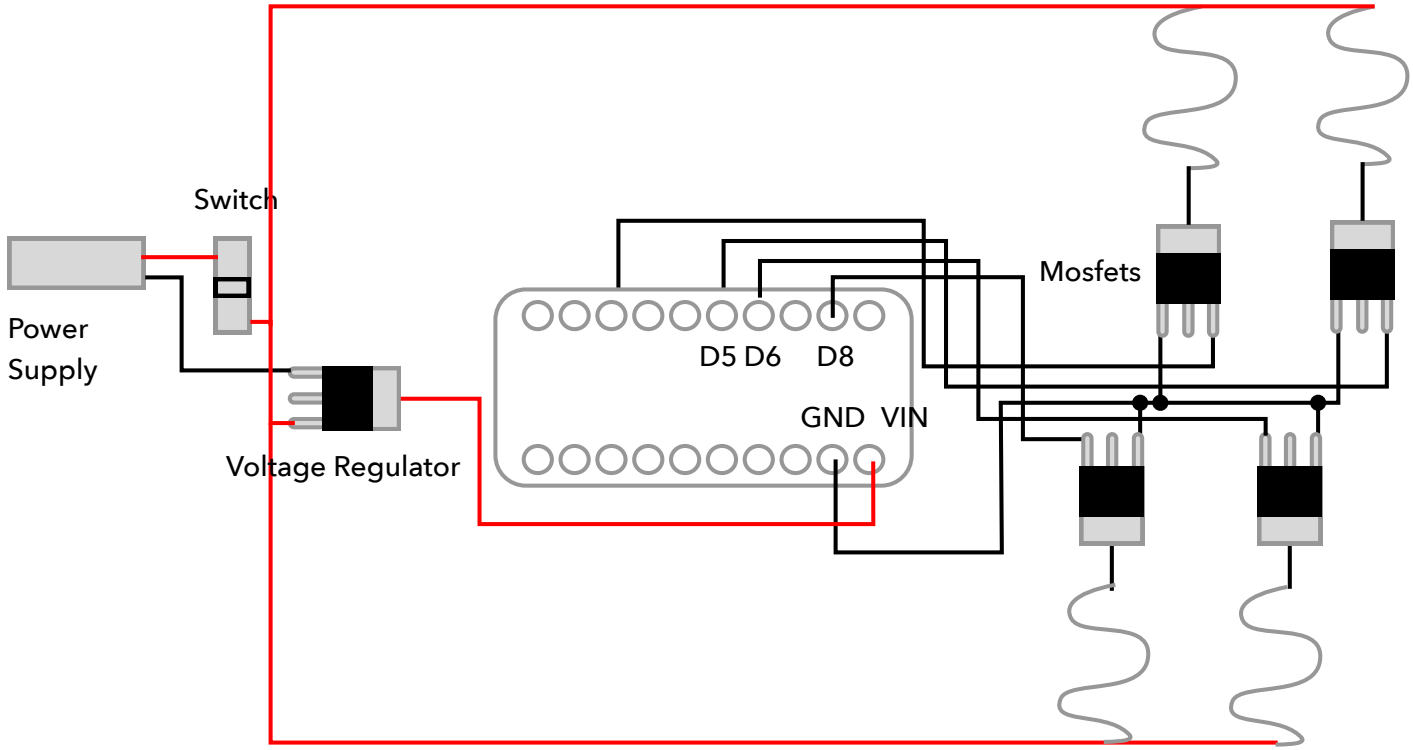
Microcontroller:
NodeMCU-DevKit 1.0

Heating yarn:
BEKINOX VN12-3x275-175s

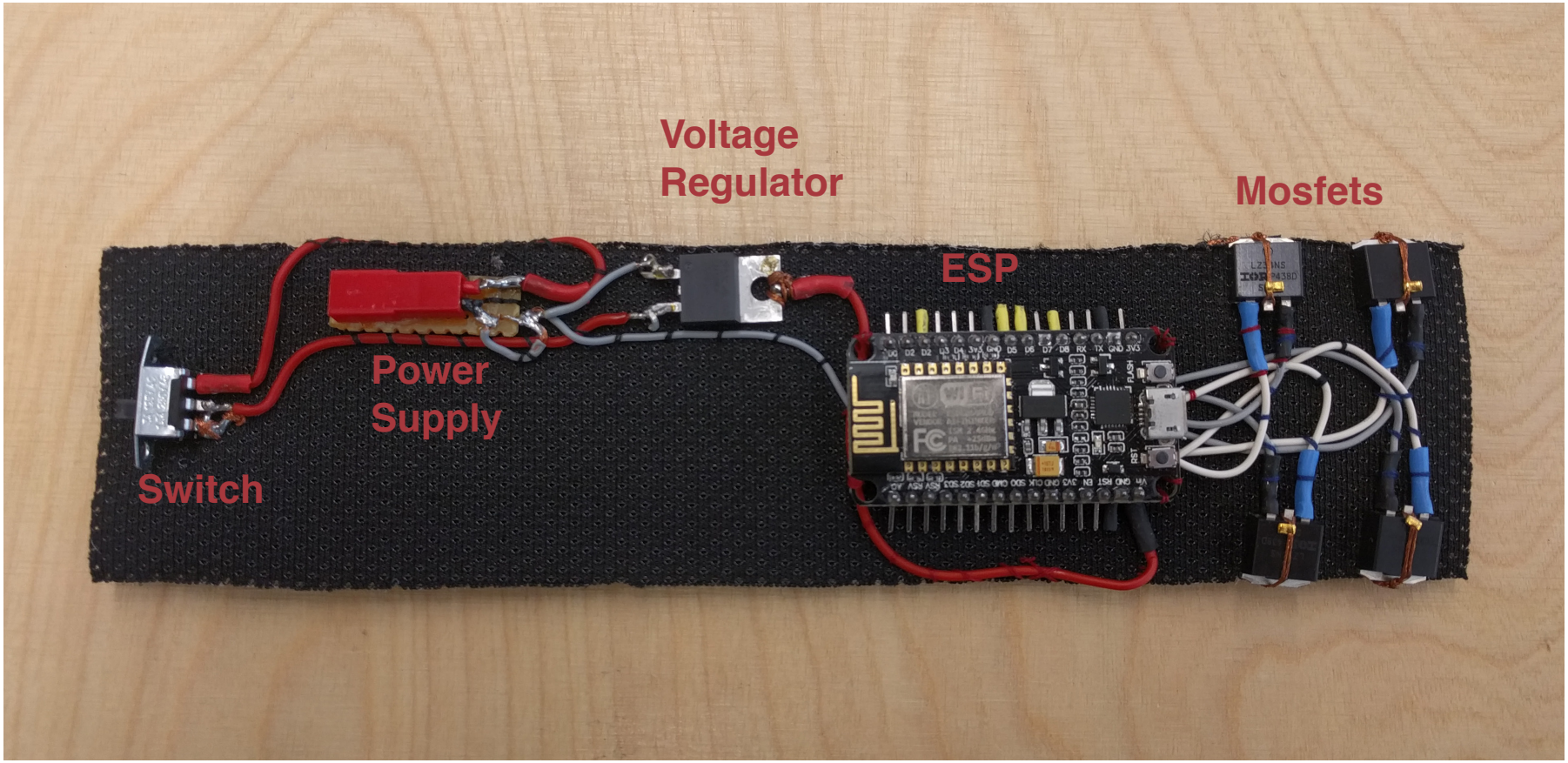
Circuit


Heating
10Ω
Heating
10Ω
Heating
10Ω
Heating
10Ω
S1
IC1
78XX
IN
OUT
GND
U2
U1
NodeMCU
V1.0
A0
RSV
RSV
SD3
SD2
SD1
CMD
SD0
CLK
GND
3V3
EN
RST
GND
Vin
D0
D1
D2
D3
D4
3V3
GND
D5
D6
D7
D8
RX
TX
GND
3V3
Q4
Q3
Q2
Q1

Circuit/Plan



Circuit/Plan



General

The circuit is designed for regulating the electrical current which flows through the heating yarn. Therefore the Heating Yarn is connected almost directly to the power-source. The Mosfets, which can switched by the microcontroller regulate the draining from the Heating Yarn. Although the system operates in the spectrum of Extra-low voltage (< 120V DC) it can serve up to 3,2 A from the battery and therefore should be handled with caution. Touching or direct skin contact with the heating elements (the Bekinox) when full powered is not advised. (and not tested) Furthermore this is a Prototype and should be handled like one.

Function

In our prototype we used a Node MCU, an open source IoT platform. The Hardware of this Single-board microcontroller is based on a ESP-12 module. The ESP-12 module serves as a low-cost Wi-Fi chip with full TCP/IP stack and MCU (microcontroller unit) capability. While keeping microcontroller functionalities like Pulse-width modulation, the Node MCU can also provide websockets and is therefore compatible with a lot of protocols already in use in different IOT and smart home environments like MQTT. The Prototype can work in three ways:

1. As a standalone server and access point in an Ad Hoc Network:
The Node MCU is programmed to work as the access point (for other devices to connect) in a small Ad-hoc-Network. It serves a HTML website to a fixed IP. The website provides gui elements which can be used to control the shawl. This connection structure does not rely on a pre-existing infrastructure but shows, when used, a high number of devic- disconnects and proves therefore to be to fragile for extensive user testing.

2. As a standalone server in existing Wifi Network:
The Node MCU is programmed to work as a web server serving http in the local wifi network. It serves a HTML website to a fixed IP. The website provides gui elements which can be used to control the shawl. Every Prototype has a different IP Address and a custom GUI elements tailored to its functions.

3. As a MQTT Client in in an existing Wifi Network with :
MQTT (Message Queue Telemetry Transport) is an publish-subscribe-based messaging protocol for use on top of the TCP/IP protocol. Thereby a central broker is responsible for distributing messages to interested clients based on the topic of a message. The Node MCU is programmed as a client in an existing MQTT Network: It can subscribe to a specific topic, which is managed and updated by the broker. So the Prototype can be easily integrated in bigger and complex IOT Environments.

Microcontroller

The microcontroller, a Node MCU is programmed with mircopython. After deploying the firmware to it, the Node can be programmed via REPL over the serial port directly. REPL works like an interactive python interpreter. Programs can also be uploaded via the provided script. (webrepl_cli.py)
To have the program executed when the controller starts, it needs to be named:
main.py

While working with the prototype we discovered that the Node MCU serves http much more stable when integrated in an existing wifi network. Therefore we gave it a fixed ip in the Lab Network.

The used source code can be find attached.

https://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html

http://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html#deploying-the-firmware

https://github.com/micropython/webrepl

Microcontroller

The microcontroller, a Node MCU is programmed with mircopython. After deploying the firmware to it, the Node can be programmed via REPL over the serial port directly. REPL works like an interactive python interpreter. Programs can also be uploaded via the provided script. (webrepl_cli.py)
To have the program executed when the controller starts, it needs to be named:
main.py

While working with the prototype we discovered that the Node MCU serves http much more stable when integrated in an existing wifi network. Therefore we gave it a fixed ip in the Lab Network.

The used source code can be find attached.

https://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html

http://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html#deploying-the-firmware

https://github.com/micropython/webrepl

Sourcecode 1/3

```
import network
import time
import ubinascii
import micropython
import socket
import machine

p2 = machine.Pin(4)
pwm2 = machine.PWM(p2)
p5 = machine.Pin(14)
pwm5 = machine.PWM(p5)
p6 = machine.Pin(12)
pwm6 = machine.PWM(p6)
p8 = machine.Pin(15)
pwm8 = machine.PWM(p8)
heatarray = [pwm8,pwm6,pwm5,pwm2]
beginningvalue = 400
beginningvalue_small = 200

html = """<!DOCTYPE html>
<html lang="en">
<html>
<head><meta charset="utf-8">
<style>
body {background-color: white;}
h2   {color: black;}
p    {color: red;}
.button {
   background-color: grey; /* Green */
   border: none;
   color: white;
   padding: 14px 25px;
   text-align: center;
   text-decoration: none;
   display: inline-block;
   font-size: 25px;}

.button2 {background-color: #50C88C;} /* Blue */
.button3 {background-color: black;} /* Red */
</style>
<title>Thermohaptic heat shawl</title> </head>
<h2>Thermohaptic heat shawl</h2>
<form>
<button class="button button2" name="LED" value="ON0" type="submit">Heating Full</button>
```

Sourcecode 2/3

```
<button class=”button button3” name=”LED” value=”ONSMALL” type=”submit”>Heating Soft</button><br><br>
<button class=”button button3” name=”LED” value=”ONFIRST” type=”submit”>Heating Group1</button>
<button class=”button button3” name=”LED” value=”ONSECOND” type=”submit”>Heating Group2</button>
<button class=”button” name=”LED” value=”OFF0” type=”submit”>Heating off</button><br><br>
</form>
</html>
“””

def do_connect():
  sta_if = network.WLAN(network.STA_IF)
  ap_if = network.WLAN(network.AP_IF)
  if not sta_if.isconnected():
    print(‘connecting to network...’)
    sta_if.active(True)
    sta_if.ifconfig((‘192.168.0.88’,’255.255.255.0’,’192.168.0.1’,’8.8.8.8’))
    sta_if.connect(‘Daan’, ‘08774323’)
    while not sta_if.isconnected():
      pass
  print(‘network config:’, sta_if.ifconfig())
  ap_if.active(False)

do_connect()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((‘’, 80))
s.listen(5)
while True:
  conn, addr = s.accept()
  print(“Got a connection from %s” % str(addr))
  request = conn.recv(1024)
  print(“Content = %s” % str(request))
  request = str(request)
  LEDON0 = request.find(‘/?LED=ON0’)
  LEDOFF0 = request.find(‘/?LED=OFF0’)
  LEDONSMALL= request.find(‘/?LED=ONSMALL’)
  LEDONFIRST = request.find(‘/?LED=ONFIRST’)
  LEDONSECOND = request.find(‘/?LED=ONSECOND’)
  #print(“Data: “ + str(LEDON0))
  #print(“Data2: “ + str(LEDOFF0))
  if LEDON0 == 6:
    #for i in range(0,4):
    #  beginningvalue = beginningvalue +100
    #  heatarray[i].duty(beginningvalue)
    heatarray[0].duty(beginningvalue+150)
    heatarray[1].duty(beginningvalue+250)
    heatarray[2].duty(beginningvalue+350)
```

Sourcecode 3/3

```
    heatarray[3].duty(beginningvalue+400)
    print('TURN LED0 ON')
  if LEDOFF0 == 6:
            for i in heatarray:
                    i.duty(0)
    beginningvalue = 400
    print('TURN LED0 OFF')
  if LEDONSMALL == 6:
    heatarray[0].duty(beginningvalue_small+150)
    heatarray[1].duty(beginningvalue_small+250)
    heatarray[2].duty(beginningvalue_small+350)
    heatarray[3].duty(beginningvalue_small+400)
    print('TURN LED0 SMALL')
  if LEDONFIRST ==6:
    heatarray[0].duty(100)
    heatarray[1].duty(beginningvalue+250)
    heatarray[2].duty(beginningvalue+350)
    heatarray[3].duty(100)
  if LEDONSECOND ==6:
    heatarray[0].duty(beginningvalue+150)
    heatarray[1].duty(100)
    heatarray[2].duty(100)
    heatarray[3].duty(beginningvalue+400)
  response = html
  conn.send(response)
  conn.close()
```