Making an Ambient Sound and Light Recorder

Knowledge is a process of piling up facts; wisdom lies in their simplification.
Martin H Fischer

<u>Modification Record</u>

Issue	Date	Author	Changes (Including the change authority)
A	09 July 2021	ZizWiz	Original from various notes

Contents

1. Introduction

This project will log the ambient light and sounds level. This data will allow us to work out what the background noise levels are likely to be. This type of logging can often show up some surprises for example in the Beta version I ran in mid-summer when it gets light early there is a noticeable noise increase when the birds sing their dawn chorus which start many minutes before you see any light and only lasts around 90 minutes then it goes quiet again.

2. Add SD card Storage and Real Time clock.

In this version we will add a Real time clock and an SD card reader/writer. Once we do this, we can log the data we get and in later versions we could display it on the PC or on a Graphics screen. In this example the data is stored on the SD card in the root as a file called `Datalog.csv`.

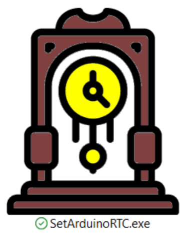
2.1. SD card Reader

This will need interfacing using SPI mode. There are other modes but they are not open source so we cannot use them. The spec of the module says it will be Ok to use a

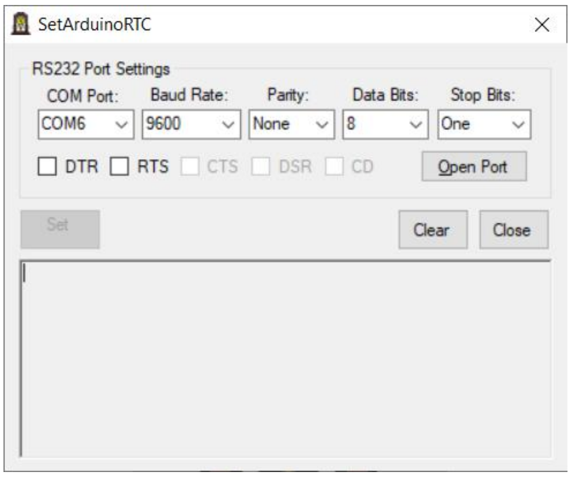
- MicroSD card up to 2GB in size. If you cannot find one that small them, you can always use a larger one but only allocate 2GB of space and format that pace only leaving the rest of the card unallocated and unformatted.
- MicroSDHC card up to 32GB in size.
- All cards should be formatted FAT16.
- All files on card should be named 8.3

2.2. Real Time Clock

Before you can use this you will need to set up the clock module DS3231. If you have just bought the clock module it is recommended to change the button battery for a new fully charged one. Wire the circuit as per the diagram shown below. Next load the code from the RTC folder called rtc.ino into the Nano. Now start the SetArduinoRTC.exe on your PC/Laptop.



You will now see a UI

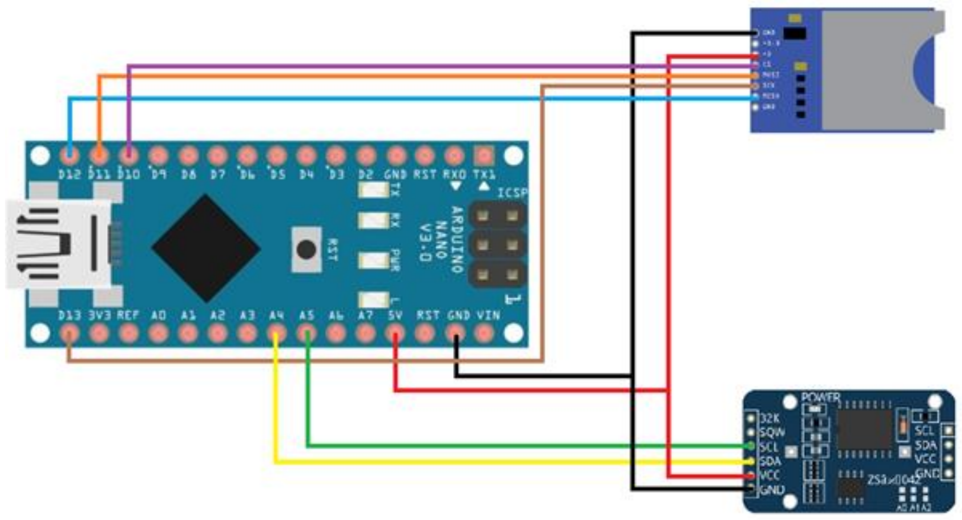


Connect this to the Nano comm port. When connected the set button will be enabled and you can now press it and the clock will be set to the time and date on your PC/Laptop.

You can now exit the windows app and reprogram the Nano to work with the Sound and Light system.

2.3. Wiring diagram

The diagram used is shown below. Please adapt it to your needs.


D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ICSP
ARDUINO NANO V3.0
TX
RX
PWR
L
RST
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
POWER
32K
SQW
SCL
SDA
VCC
GND

2.4. Things to Note

We are using an SD Flash card. When you write to a card it copies out a block, adds the new info and then copies back the block. Each block is 512Bytes large. This means if you have lots of other items using your Ram then you may not have enough and when you compile you will get an Alert telling you the system may now be unstable and likely to have issues when it is running.

To get use less memory you can use less variables (May not be possible). Write some of your variables like strings to Flash memory rather than to RAM.

If we write the following to RAM

Serial.println("Card failed, or not present");

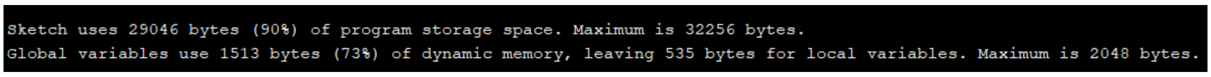
We can see it uses 28 bytes counting all the characters including spaces which are also characters.

If we save this same string to FLASH

Serial.println(F("Card failed, or not present"));

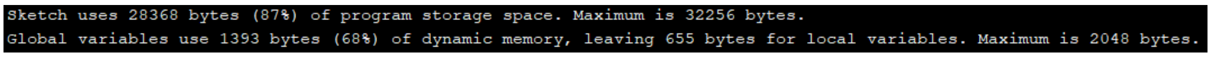
We now save 28 bytes more of RAM to use

In the code we use in this example we end with just over the minimum amount of memory free for writing to the SD card. Even though this is more than the 512bytes needed for writing to an SD card the system would not run very stably.



```
Sketch uses 29046 bytes (90%) of program storage space. Maximum is 32256 bytes.
Global variables use 1513 bytes (73%) of dynamic memory, leaving 535 bytes for local variables. Maximum is 2048 bytes.
```

After removing some items from the code mostly the “Smileys” and changing some others like making variables local to each function rather than Global, I end up with the following



```
Sketch uses 28368 bytes (87%) of program storage space. Maximum is 32256 bytes.
Global variables use 1393 bytes (68%) of dynamic memory, leaving 655 bytes for local variables. Maximum is 2048 bytes.
```

This figure seems to be a lot better and allows the system to run. If I make any future versions, I may need to look at the architecture and maybe rewrite the code in a different way but for now it is working.

The display on this version will not use smileys for reasons described above.

2.5. Details of SD card used

File System:	FAT16
Sectors per Cluster:	32 (16384 bytes)
Number of Fats:	2
Reserved Sectors:	1
Sectors per FAT:	243
Root Entries Limit:	512

3. Add Sound Monitoring.

We will add a sound monitor and record the ambient sound level once every hour. First we will need to experiment a bit to see how best to implement this.

The module we will use the MAX4466 chip. There are other on the market and some that may be better spec. This device is cheap and readily available in our box of sensors so no need for us to order and wait. In future we may look at better sensors but for now we will use this one.

For the first part of this we will just create a circuit that will have the Sound sensor and a SD card for logging. Once we are happy, we have this working we will move it in with the rest of the setup we made in v3.

3.1. Spec of sensor

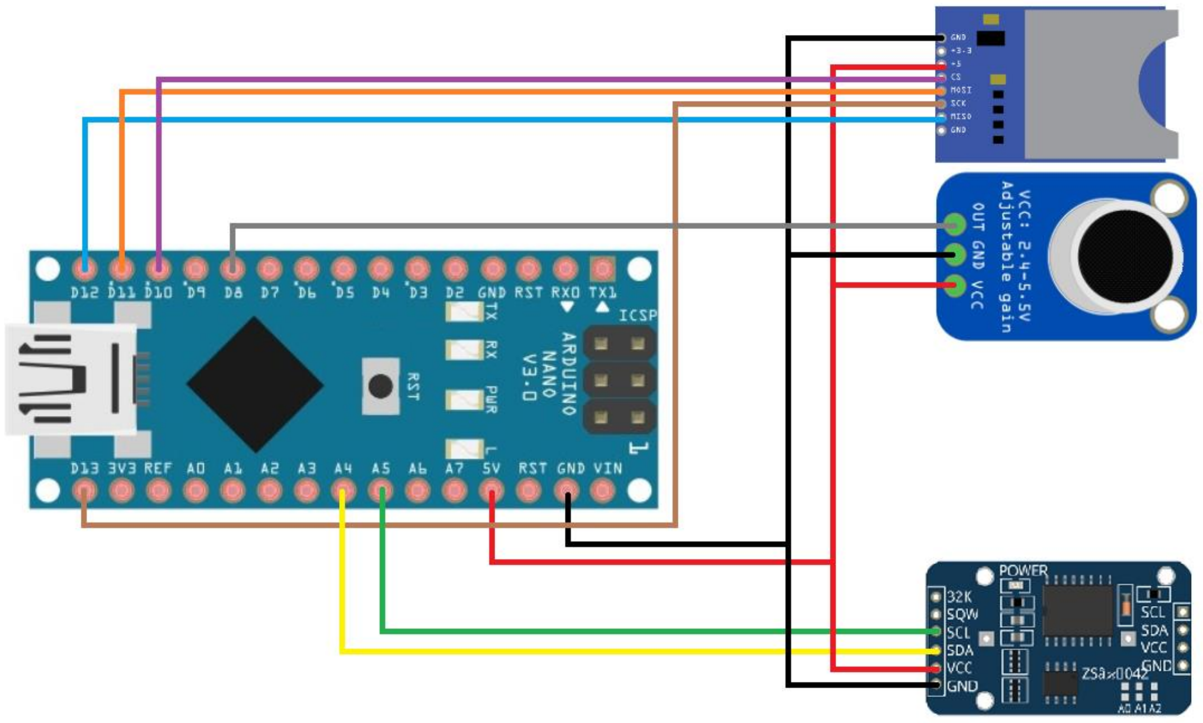
This fully assembled and tested board comes with a 20-20KHz electret microphone soldered on. For the amplification, we use the Maxim MAX4466, an op-amp specifically designed for this delicate task! The amplifier has excellent power supply noise rejection, so this amplifier sounds really good and isn't nearly as noisy or scratchy as other mic amp breakouts we've tried!

This breakout is best used for projects such as voice changers, audio recording/sampling, and audio-reactive projects that use FFT. On the back, we include a small trimmer pot to adjust the gain. You can set the gain from 25x to 125x. That's down to be about 200mVpp (for normal speaking volume about 6" away) which is good for attaching to something that expects 'line level' input without clipping, or up to about 1Vpp, ideal for reading from a microcontroller ADC. The output is rail-to-rail so if the sounds gets loud, the output can go up to 5Vpp!

Using it is simple: connect GND to ground, VCC to 2.4-5VDC. For the best performance, use the "quietest" supply available (on an Arduino, this would be the 3.3V supply). The audio waveform will come out of the OUT pin. The output will have a DC bias of VCC/2 so when its perfectly quiet, the voltage will be a steady VCC/2 volts (it is DC coupled). If the audio equipment you're using requires AC coupled audio, place a 100uF capacitor between the output pin and the input of your device. If you're connecting to an audio amplifier that has differential inputs or includes decoupling capacitors, the 100uF cap is not required.

The output pin is not designed to drive speakers or anything but the smallest in-ear headphones - you'll need an audio amplifier if you want to connect the amp directly to speakers. If you're connecting to a microcontroller pin, you don't need an amplifier or decoupling capacitor - connect the OUT pin directly to the microcontroller ADC pin.

3.2. Circuit Diagram



3.3. Issue to get around

On the sensor we can get an analogue to Digital signal. Some sensors have both and some one or the other. We will use the Digital signal. The digital signal will return 1 when silent or 0 when there is noise. It does not give us the level of noise in db but if we sample the signal for a time, we can work out how many times in a set period it is high and use this figure to indicate ambient noise. The threshold for changing between 0 and 1 can be changed on the sensor by adjusting the potentiometer. We will experiment with this.

3.4. Calibrating the Sensor

To get accurate readings out of your sound sensor, we need to calibrate it via its built-in potentiometer. By turning the knob of the potentiometer, you can set a threshold. So that when the sound level exceeds the threshold value, the digital output (OUT) will output LOW.

If your sensor has a Status LED then to calibrate the sensor, start clapping near the microphone and adjust the potentiometer until you see the Status LED on the module blink in response to your claps.

If you do not have a sensor LED then you could add a LED to the circuit attached to another digital pin and then when you get a digital low from the sensor then can turn on the LED.

3.5. Code

We found a library that is very useful here. It is called NoDelay.h. When we use this library it will run a timer and when it reaches the value stated it will then trigger a function to run.

For this to work you need to declare the function (in our case runsound()) before you declare the noDelay.runsoundtime(time, function) with arguments. Below we set the function runsound to run every 12 minutes.

```
void runsound();
noDelay runsoundtime(720000L, runsound);
```

Then in the loop just call noDelay.runsoundtime.update() which will check if set time has past and if so will run set function

```
void loop() {
  runsoundtime.update();
}
```

Inside the function runsound we start a loop that lasts 1 minute. During this loop we count the number of times the sensor has heard a sound.

We found that a loop value of `10600000` will equate to just over 60seconds. We could use `millis()` but we have found our method to work well at present. Maybe in the future we will change it. This value number of sounds heard is then written to the SD card with a time stamp.

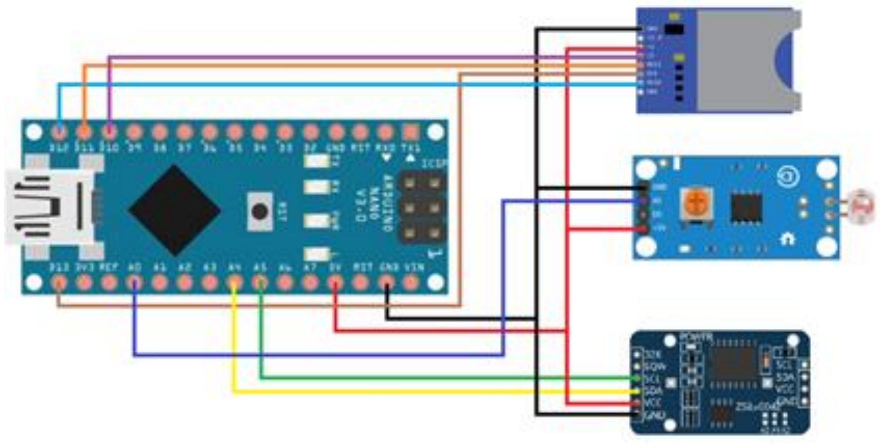
The data can be removed and then plotted to a graph. The graph would then show the ambient noise, not volume but how busy with noisy the background is.

4. Add Light Intensity Monitoring.

The following is a list of rough Lux that you could expect in various light conditions.

Illuminance (lux)	Surfaces illuminated by
0.0001	Moonless, overcast night sky (starlight)
0.002	Moonless clear night sky with airglow
0.05 – 0.3	Full moon on a clear night
3.4	Dark limit of civil twilight under a clear sky
20 – 50	Public areas with dark surroundings
50	Family living room lights
80	Office building hallway/toilet lighting
100	Very dark overcast day
150	Train station platforms
320 – 500	Office lighting
400	Sunrise or sunset on a clear day.
1000	Overcast day;
10 000 – 25 000	Full daylight (not direct sun)
32 000 – 100 000	Direct sunlight

4.1. Circuit Diagram



4.2. Code

The lux (symbol: lx) is the SI unit of luminance, measuring luminous flux per unit area. It is equal to one lumen per square meter. In photometry, this is used as a measure of the intensity, as perceived by the human eye, of light that hits or passes through a surface.

The human eye can see between 380–740 nano metres so our sensor should see the same range. In our case the sensor using will give an analogue reading between 0 and 1024. As the sensor we are using is a cheap one it accuracy will not be the best but enough to give an indication of the ambient light conditions. There is a somewhat rough formula that relates the resistance of a sensor (Rldr) to the light in Lux. That is:

Rldr = 500/Lux   or   Lux = 500/Rldr kΩ

```
//Light Intencity Measurement
//www.circuits4you.com

double Light (int RawADC0)
{
  double Vout=RawADC0*0.0048828125;
  //int lux=500/(10*((5-Vout)/Vout));//use this equation if the LDR is in the upper part of
the divider
  int lux=(2500/Vout-500)/10;
  return lux;
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Light Intensity:");
  Serial.print(int(Light(analogRead(0))));
  Serial.println(" Lux");
  delay(1000);
}
```

5. Ambient Noise and Light Intensity Monitoring.

In this project we will use a light sensor and a noise sensor to record the ambient levels every 12 minutes. Both our sensors use a LM393 comparator. We will log this data to an SD card. The clock module we will use is the DS3231.

5.1. No Delay Library

We found a library that will help us called NoDelay which is available from: https://github.com/M-tech-Creations/NoDelay

5.1.1. About this Library

From the author of the library notes:

- To start create a noDelay variable. From here you are able to set a delay time or a delay time and function to run or nothing at all.
- To check it if the delay time has passed, call the update function check time and run the set function.
- Using the start function, you are able to restart the timer for delay. This allows you more control over then a delay with trigger.
- The delay time can be changed using setDelay using a time in milliseconds.

5.1.2. Usage

```
void runsound();//Must declare function before noDelay is called
noDelay runsoundtime(720000L, runsound); //Creates a noDelay variable set to 12min and after that it wil call runsound()

void loop() {
  runsoundtime.update();//will check if set time has past and if so will run set function
}
```

5.2. runsound()

In this function we will sample the noise in a loop for 1 minute. Every time we hear a noise via the digital pin we count it as a 1. We sum all the 1s and this gives us the ambient sound for that period. The nosier the background the higher the number we get.

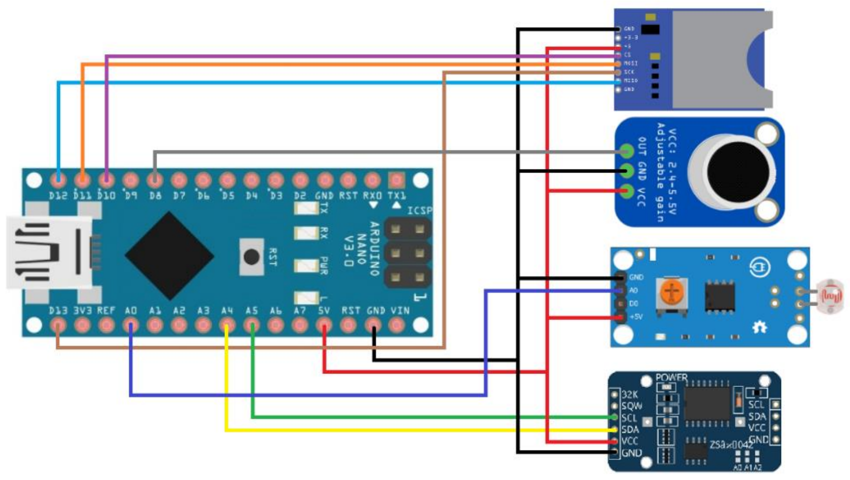
5.3. runlight()

In this function we read the analogue input and convert it to lux which is what we record. The following table is a list of rough Lux that you could expect in various light conditions.

Illuminance (lux)	Surfaces illuminated by
0.0001	Moonless, overcast night sky (starlight)
0.002	Moonless clear night sky with airglow
0.05 – 0.3	Full moon on a clear night
3.4	Dark limit of civil twilight under a clear sky
20 – 50	Public areas with dark surroundings
50	Family living room lights
80	Office building hallway/toilet lighting
100	Very dark overcast day
150	Train station platforms
320 – 500	Office lighting

400	Sunrise or sunset on a clear day.
1000	Overcast day;
10 000 – 25 000	Full daylight (not direct sun)
32 000 – 100 000	Direct sunlight

5.4. Circuit Diagram


D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ICSP
TX
RX
PWR
L
RST
ARDUINO NANO V3.0
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
VCC: 2.4-5.5V Adjustable gain
OUT GND VCC
GND
A0
D0
+5V
POWER
32K
SQW
SCL
SDA
VCC
GND
ZS-042

5.5. Code

The code is run every 12 minutes. The value on A0 is the ambient light. When the Sound runs, we run for 60 seconds and sample the value on D8. Every time we see sound which is LOW, we add 1 to our total. After 60 seconds this total is the ambient sound level.

6. Powering the system from a Power Bank

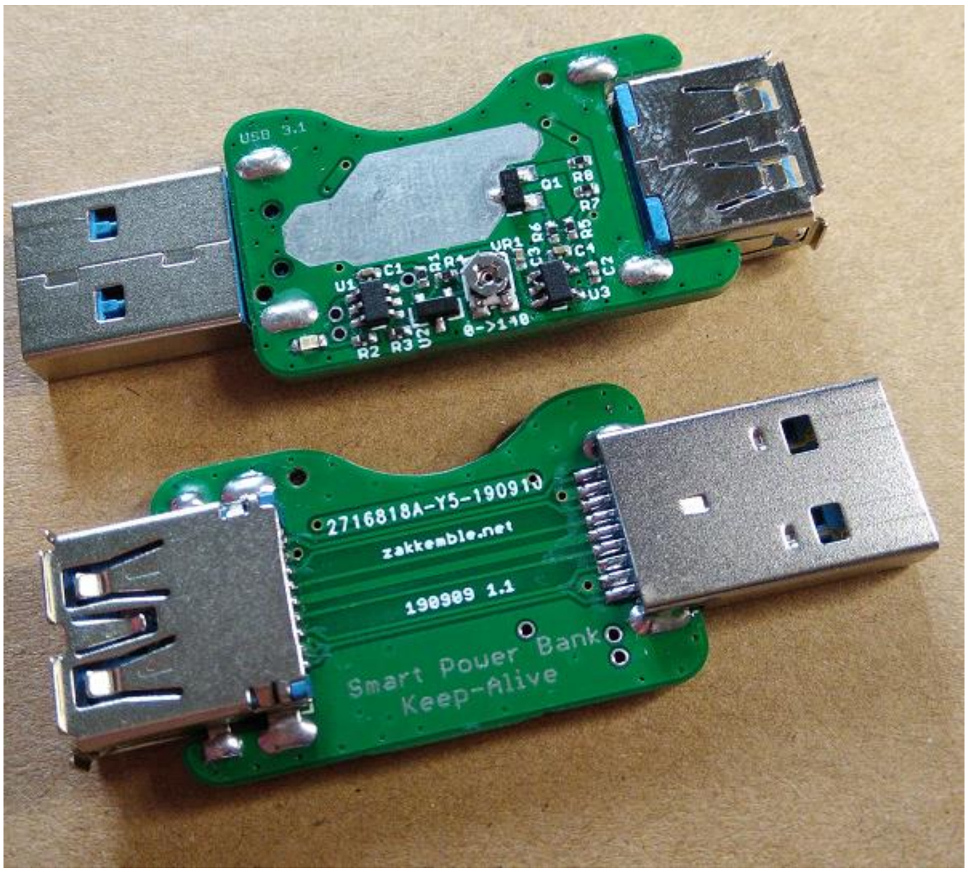
We thought this would be the best way to run this as it is portable. Unfortunately, all the power banks we had to hand conform to EU standards. This means that as the Nano does not draw enough power they shut off after a few seconds.

After some searching and getting advice from others, I set out trying to find something that would act like a load without draining the battery. I came across several “Smart Power Bank Keep-Alive” devices.

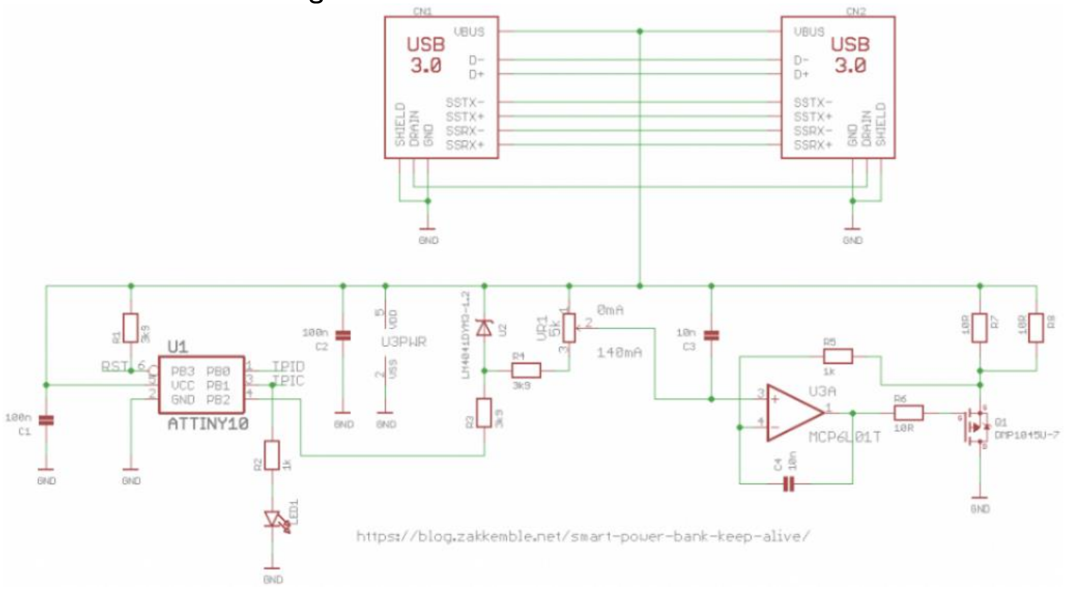
6.1. Solution 1

More information on this item can be found at the following websites:

- https://blog.zakkemble.net/smart-power-bank-keep-alive/
- https://github.com/zkemble/SmartPowerBankKeepAlive



6.1.1. Circuit Diagram



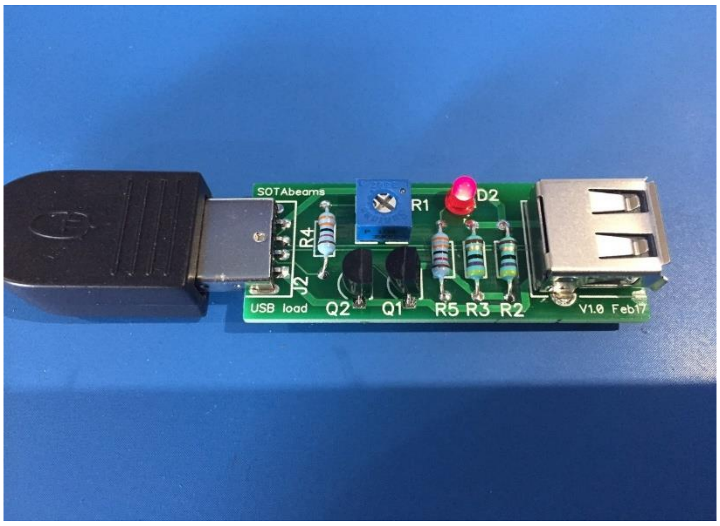
6.1.2. Result

This item was used and once the load had been adjusted it kept the power supplied.

6.2. Solution 2

More information on this item can be found at the following website:

- https://www.sotabeams.co.uk/usb-battery-pack-keep-alive-load/

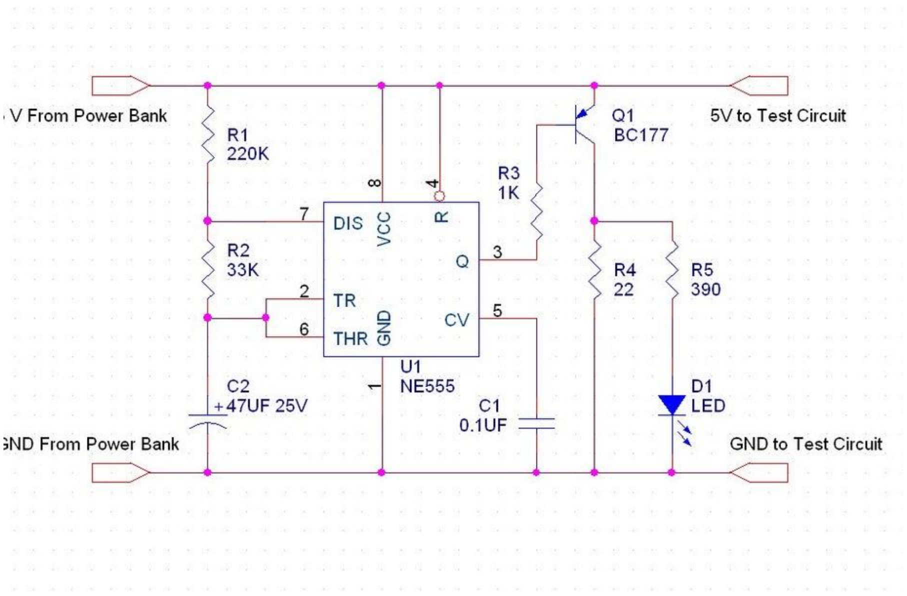


At the moment all the parts have arrived to build this design but it is waiting to be built.

6.3. Solution 3

More information on this item can be found at the following website:

- https://www.instructables.com/Current-Pulsing-Keeps-Power-Bank-Active/



The parts have not yet been ordered to build this one yet.