

Instructions:

SOLAR EXPERIMENTERS KIT

FOR MICRO:BIT

v1B



V1/V2 compatible



By David Whale

TABLE OF CONTENTS

Introduction.....	2
Parts.....	3
Using Alligator Clips.....	4
Solar Board Reference.....	5
Project 1 – Sun Finder.....	6
Project 2 – Garden Light.....	8
Project 3 – Self Charging Cooling Fan.....	10
The BBC micro:bit.....	12
Getting Code to Run on the micro:bit.....	12
Project 4 – Adding an Energy Meter.....	14
Project 5 – Energy Logger.....	22
How the Solar Store Works.....	29
Project 6 – Intelligent Cooling Fan	30
Troubleshooting.....	38
About the Author.....	39
Learning.....	39
Monk Makes Kits.....	40





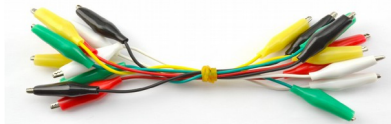
INTRODUCTION

The MonkMakes Solar Experimenters Kit for micro:bit is a project kit that allows you to experiment with harvesting energy from the sun and other light sources. It consists of a solar panel to harvest the energy, a solar store that stores the harvested energy, and a low energy light bulb and a motor that can be driven with the energy that you harvest.

There are three projects that introduce energy harvesting without the micro:bit, followed by 3 bigger projects that use the micro:bit (not provided) as an intelligent controller. The micro:bit monitors and manages the charging and discharging of the solar store.

With this project kit you will learn all about how tiny amounts of energy can be harvested from the sun and stored for later use, using a practical and experiment-led approach.

PARTS

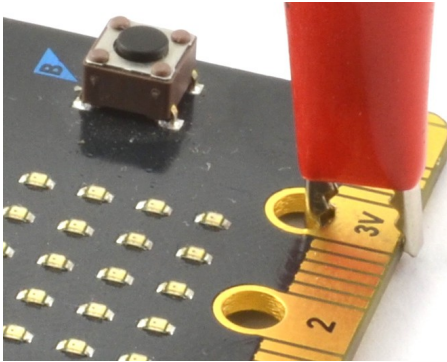
Quantity	Item	Picture
1	Solar Panel	
1	Solar Store	
1	Bulb (Please note the color and size of this blub may vary).	
1	Motor and fan	
10	Set of alligator clip leads	

You will also need:

- * A BBC micro:bit (this kit is compatible with both a V1 and a V2 micro:bit)
- * A USB data cable to connect to a computer
- * A computer (PC, Mac, Linux including Raspberry Pi) and access to the internet.

USING ALLIGATOR CLIPS

When using the alligator clips to connect your micro:bit to the MonkMakes boards, you have to be a bit careful how you connect the clips to ensure you get a good connection. The best way is to connect the clips vertically, as show below:



Connecting the alligator clips like this prevents any accidental connections between the large connectors with the holes in and the much smaller connectors (gold lines in the photo above).

The colour of the wires is not important to the functioning of the circuits you build, but you will find it useful to use a colour scheme in your projects so that it is more obvious how the circuit is wired. We suggest the following general scheme:

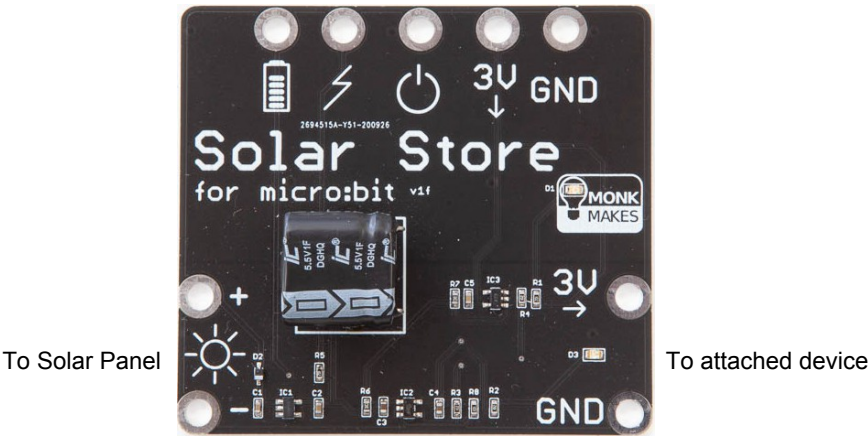
Black – for 0V (GND – short for *ground*) connections

Red – for 3V (power) connections

Yellow – for 10V (solar power) connections

SOLAR BOARD REFERENCE

To micro:bit

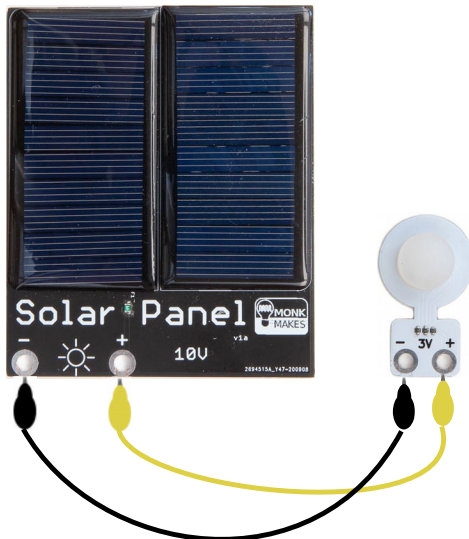


	Charge% (output) – stored charge indicator. A voltage between 0V and 3V that can be connected to a micro:bit pin.
	Level (output) – stored charge (raw). A voltage between 0V and 5V that can be looped directly to the enable pin, to auto-enable when charged. Note: Do <u>not</u> connect to micro:bit
	Enable (input) – enables the power output, so that any stored energy can be used by the attached device.
	Power (input) – provides power to the Solar Store board, primarily so that the Charge% output works. Connect 3V and GND to micro:bit.
	Solar Power (input) – connect this to your Solar Panel to harvest energy, which is stored in the Solar Store. Accepts a voltage up to 10V. Make sure that + goes to the + on the Solar Panel, and - goes to the - on the Solar Panel.
	Power (output) – connect your attached device here. This output is only enabled when the Enable pin is active <i>and</i> there is sufficient stored energy in the Solar Store. The LED (D3) will light when there is enough stored energy.
	Power LED – indicates that power is provided to the Solar Store and the Charge% indicator connection to the micro:bit is working.

PROJECT 1 – SUN FINDER

In this project you will build a device that finds where the sun is. It uses a solar panel (photo-voltaic cell) and a low energy bulb to build a device that lights the bulb when enough sunlight is nearby. It demonstrates limitations of directly connecting the Solar Panel to a device that uses the energy harvested from the sun.

Connecting Up



The Solar Panel will provide up to 10V but it won't provide a lot of current at that voltage, so it is safe to connect to the Bulb. However, avoid really bright sunlight when doing this experiment so that you don't risk overloading your Bulb.

How it Works

The sun finder project directly connects the Solar Panel to the Bulb, and shows you immediately how much energy is being harvested as light falls onto the Solar Panel. The Solar Panel converts the light into voltage by the photo-voltaic effect. This is a physical and chemical process that excites the electrons to a higher energy state, and a small voltage is generated.

This voltage travels across the wires, and if it is high enough, the LEDs inside the MonkMakes Bulb perform the reverse process and turn that voltage back into light. As more light falls on the Solar Panel, a higher voltage appears on the wires, and that higher voltage causes the Bulb to shine brighter.

But, there is a problem! If you want to power your circuits when the sun is not out or when it is in a dark room, this direct connection method will not work!

Experiments

1. How bright does the daylight need to be for the Bulb to start to glow?
2. How bright does the sun need to be for the Bulb to glow brightly?
3. What wattage of electric light bulb (as a sun substitute) gives the best glow of your Bulb?
4. Are LED light bulbs or tungsten light bulbs better for energy harvesting?
5. What happens if you cast a shadow over the solar panel when the sun shines?
6. What happens if it is totally dark?

Results

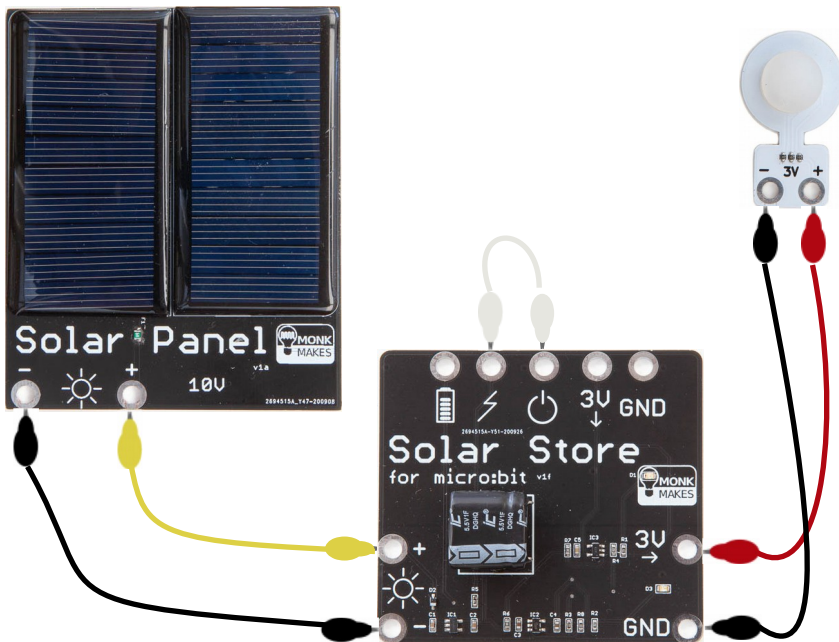
Record the observations you make from your experiments in this table, and they will be useful in later projects.

1	
2	
3	
4	
5	
6	

PROJECT 2 – GARDEN LIGHT

In this project you will build a device very similar to those stick-in-the-ground self charging garden lights, that charge up an energy store that then powers a bulb, and is the first project where you use the energy storage board. By storing the harvested energy, the Bulb will stay on even when it gets dark.

Connecting Up



The loop cable between the Level and Enable connectors on the Solar Store will make the board work in standalone mode.

This means that when the amount of stored energy in the board reaches a sufficient level, it will enable the 3V output and the Bulb will light up. When the amount of stored energy gets low, the output will automatically be disabled.

How it Works

The Solar Panel converts light into electrical energy, and supplies a voltage to the left hand terminals of the Solar Store. The Solar Store transfers this energy into its super-capacitor (the big black component), which stores the energy until it is used.

By joining together the Level and Enable terminals on the Solar Store, you have configured it so that when the voltage of its super-capacitor has charged up enough, it will enable the output stage of the board and your attached Bulb will glow; just think of this a bit like an automatic switch. But, when the Bulb glows, it will be taking energy out of the Solar Store, and eventually the voltage will drop low enough that the output stage will be turned off, and the Bulb will no longer glow.

The MonkMakes Bulb is quite a low power device, so if you have bright sunlight then enough energy will be harvested that makes the Bulb glow and there is enough left over to store inside the Solar Store to charge it up a lot. When the sun goes down, there is enough stored energy to power the low power MonkMakes Bulb for quite a long time.

Experiments to try

1. How long do you need to shine light onto the solar panel before the Bulb starts to glow dimly?
2. How long before the Bulb starts to glow brightly?
3. Once the Bulb is shining brightly, if you disconnect the Solar Panel from the Solar Board, how long is it before the Bulb goes out?
4. Try your experiments with different light sources such as the sun or different brightness indoor lighting to see how it affects the timings.
5. Do the same experiment with the fan. Are the times shorter or longer?

Results

1	
2	
3	
4	
5	

PROJECT 3 – SELF CHARGING COOLING FAN

In this project you will use your harvested and stored energy to power a bigger device: a motor that spins a fan. This device will charge up the Solar Store when the sun is out, and uses a simple home-made switch to allow you to decide when to transfer the stored energy to the motor to spin the cooling fan.

Connecting Up



By using two separate alligator clips, you can make your own simple switch.

Touch together the ends of the two white alligator clips to enable the Solar Store, and if there is enough energy stored then the fan will spin. Disconnect the two white alligator clips to turn the fan off.

How it Works

This project is similar to project 2, but by using the two alligator clips, you have made a simple switch that allows you to decide when you want the fan on. If there is enough energy stored in the Solar Store when you connect the alligator clips together, the fan will spin.

By separating the charging phase and the discharging phase into two separate phases, you can harvest and store energy when light is available without the fan using some of that energy to power it, so your Solar Store will charge faster. Because the energy is stored in the Solar Store, when the sun goes down and there is not enough light to charge any more, you have enough stored energy to spin the fan when you want to, by joining and breaking the alligator clip connection to turn the fan on and off on whenever you need it.

All practical energy harvesting systems are designed to separate the charging and discharging phases. If you do the right measurements and calculations you can then design a well-balanced system so that it always has enough energy when you need it (even though the energy source, the sun, is not always available).

Experiments

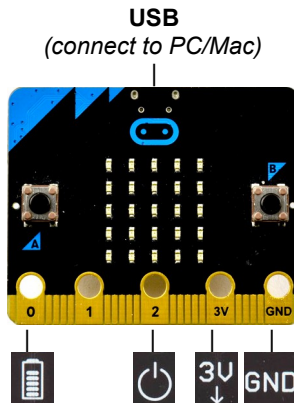
1. Connect the two alligator clips at the top to enable the board, and disconnect the fan. How long is it before LED D3 lights up to indicate some stored charge?
2. When LED D3 lights up, connect your fan to the Solar Store and time how long your fan runs for before it stops.
3. If you leave the board charging for longer, does it then allow the fan to run for longer?
4. What is the longest time you can make the fan run for before the Solar Store runs out of energy, and how long does it take with your chosen light source to charge it up to that level?

Results

1	
2	
3	
4	

THE BBC MICRO:BIT

The next projects will use the BBC micro:bit to control the Solar Store. This will allow you to build more complex projects where the energy harvesting and the energy use is further separated into distinct phases, and your micro:bit can both monitor the stored energy as well as turn things on and off intelligently.



GETTING CODE TO RUN ON THE MICRO:BIT

Entering Code by Hand

Entering code by hand is a good way to learn more about coding, as you will learn how to find and fix mistakes that you make. If you want to enter code directly, choose the correct editor for the code you are entering.

MakeCode (block coding) can be accessed at <https://makecode.microbit.org>

MicroPython (text coding) can be accessed at <https://python.microbit.org>

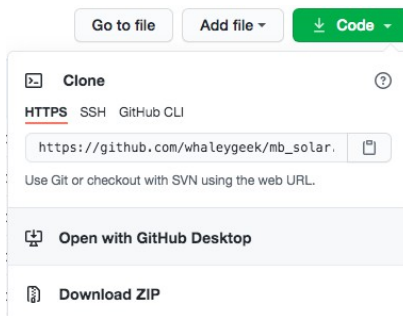
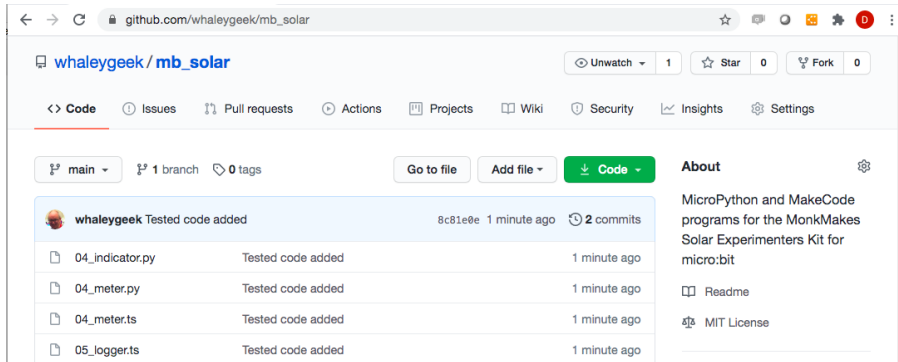
Copying the Code

If you are eager to get building, all of the code is pre-written and downloadable.

For the MakeCode (blocks) code, each program listing has a link that you can click on (or type in) to open the code directly in the MakeCode web editor.

For the MicroPython code, you can access the code from our github pages for this project. You don't need a login account in order to access github.

https://github.com/monkmakes/mb_solar_kit



Click on the CODE button, then on DOWNLOAD ZIP. The file will download into your *downloads* folder.

Unzip the zip file to a folder on your computer, and you will find all the Python programs in there. You can open these directly in the Python web editor as well as using the Python web editor to convert the code into a form that will run on the micro:bit.

Flashing Code to Your micro:bit

Once you have the program code in the correct editor, you need to *flash* it to the micro:bit. This copies the code over the USB cable into the flash memory of the micro:bit, where it then runs

Press the DOWNLOAD button, and drag & drop the *.hex* file from your *downloads* folder onto the MICROBIT drive that appears when you plug in your micro:bit.

If you haven't used the micro:bit before, there are some simple getting started guides and videos on the Micro:bit Educational Foundation website:

<https://microbit.org/get-started/first-steps/set-up/>

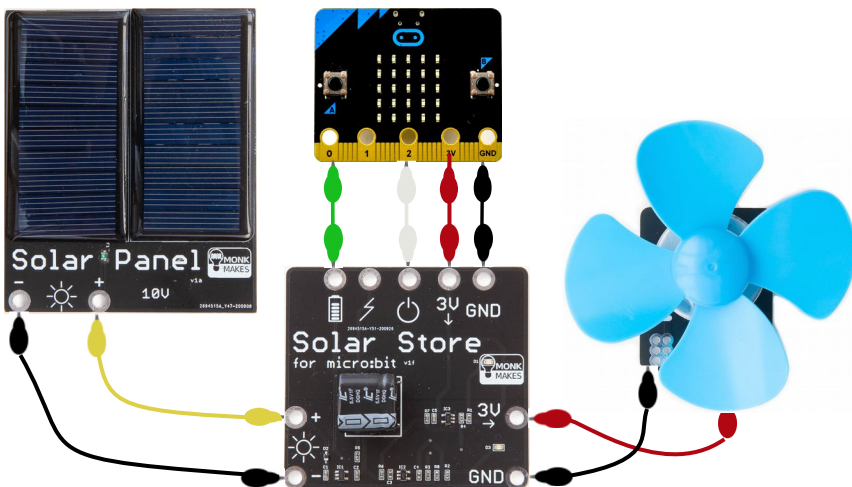
The above web page also describes a faster *direct flashing* approach, that you will find handy when doing the energy logger project.

PROJECT 4 – ADDING AN ENERGY METER

In this project you will build a device that shows an energy meter on the display of the micro:bit, so that you know how much energy is available to power anything attached to the output. In all your previous projects it has been really hard to decide how much energy is stored, because it is hard to say accurately how bright the sun is, or how bright your Bulb is.

This meter provides an accurate way to measure the stored energy in your Solar Store and display a bar-chart on the micro:bit display. If your Solar Store is getting low, you can turn off the fan to conserve energy, and when it gets hotter in your room you can turn the fan on again to cool things down a bit.

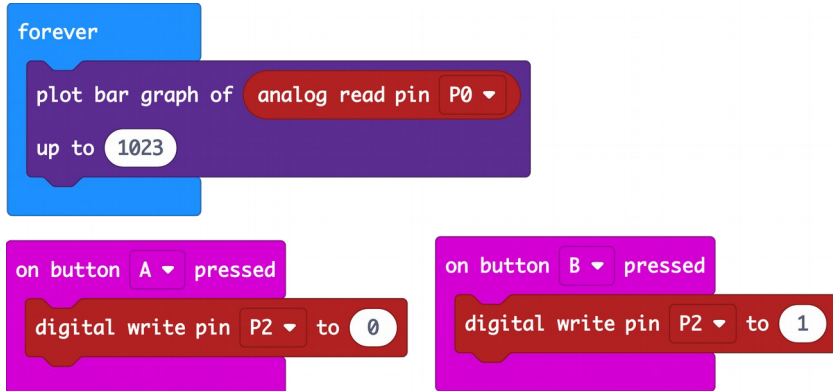
Connecting Up



Code for a Simple Bar-Chart (MakeCode)

The link for the MakeCode version of this project is here:

https://makecode.microbit.org/_R1D337cbTPe1



How it Works

Pin P0 of the micro:bit is connected to the Charge% pin on the Solar Store. This connection provides a voltage in the range of 0V to 3V that represents the stored charge in the super-capacitor on the Solar Store. The micro:bit `analog_read_pin` command reads this voltage, and turns it into a number in the range of 0 to 1023. As the voltage on the pin increases, the number increases.

MakeCode fortunately has a really easy to use bar-chart block built-in to the language, that takes that value and draws a different number of dots on the screen. Because this is a general purpose block, you also provide it with the maximum possible value (in this case 1023) so that it can correctly choose how many dots to display.

Pressing button B puts a 3V voltage on pin P2, which is connected to the Enable pin of the Solar Store, and this will route power to the right hand 3V pin on the Solar Store that then powers your fan. Pressing button A removes that voltage on pin P2 (setting it to 0V) which disables the output and the fan stops.

Note: Make sure that you also connect the 0V and 3V pins at the top of the Solar Store to the 0V and 3V pins of the micro:bit – these are needed to power a small amplifier chip that measures the Solar Store charge and presents it as a voltage on the Charge% pin. If there is not enough stored charge in the Solar Store, the fan will not run – just wait a bit longer for it to charge.

Experiments

1. Time how long it takes to charge, and record the pattern you see on the display when it is fully charged. (hint, it will charge faster in bright sunlight).
2. Disconnect the Solar Panel and press button B to make the fan spin. Time how long it takes to discharge (i.e. until the fan stops spinning). Record the pattern you see on the display when the fan finally stops.
3. See if it is possible to charge up to full charge while the fan is spinning (hint, try really bright sunlight for this). How long does it take to charge while the fan is also running?

Note: The bargraph will work better if you run the micro:bit from fresh batteries or from the USB lead connected to your computer. Batteries that are running low will cause inaccurate readings.

Results

Time to charge (fan off):	
Pattern seen for full charge:	
Time to discharge (no panel attached):	
Pattern seen for fully discharged:	
Time to charge (fan running):	

Code for a Simple Bar-Chart (MicroPython)

MicroPython doesn't have a barchart feature built in, so you have to provide the code for it yourself.

This program is called `04_meter.py`.

```
from microbit import *

P0_MAX = 812

def barchart(y, v, vmax):
    v = min(v, vmax)
    leds = int(v * 5 / vmax)
    for x in range(leds):
        display.set_pixel(x, y, 9)

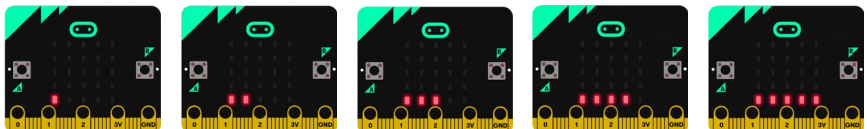
# main program
while True:
    reading = pin0.read_analog()
    display.clear()
    barchart(4, reading, P0_MAX)

    if button_a.was_pressed():
        pin2.write_digital(0) # off
    if button_b.was_pressed():
        pin2.write_digital(1) # on

    sleep(1000) # 1 second
```

Barchart Screens

The barchart moves from the left to the right by showing more dots for a larger value, like this:



How It Works

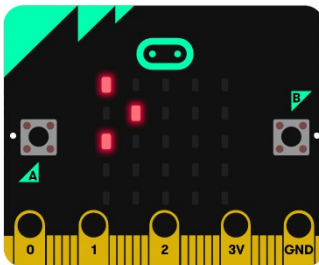
The `P0_MAX` constant sets the maximum value expected from pin P0, and this is the value that represents 100% charged. You might have to experiment with the value here to get the best performance from your barchart.

`barchart()` draws a horizontal line on the display, by turning on a different number of LEDs. 'y' is the position on the display where the barchart will appear, 'v' is the value to display, and 'vmax' is the maximum value to expect. The value is capped (with `min()`) and scaled (with the divide) to turn on between 0 and 5 LEDs.

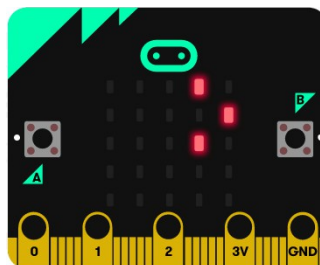
The main program loops round forever reading the Charge% pin, displaying a barchart, and checking the buttons. Button A = fan off, Button B = fan on.

Screen Designs for a Charge Direction Display

Let's build a new display feature that shows whether your Solar Store is being charged or discharged. This will make it easier for you to decide whether to use the fan or conserve energy for later.



CHARGING



DISCHARGING

Code for a Charge Direction Display (MicroPython)

This program is called `04_indicator.py`.

```
from microbit import *

P0_MAX = 812
SIGNIFICANT = 3
CHARGING = Image("90000:09000:90000:00000:00000")
DISCHARGING = Image("00090:00009:00090:00000:00000")

diffs = [0 for i in range(10)]
```

```

sumd = 0
prevval = None

def trend(newval):
    global prevval, sumd
    if prevval is not None:
        olddiff = diffs.pop(0)
        newdiff = newval - prevval
        diffs.append(newdiff)
        sumd = sumd - olddiff + newdiff
        print(prevval, newval, newdiff, sumd)

    prevval = newval
    if abs(sumd) >= SIGNIFICANT:
        return sumd
    return 0 # no change

def bargchart(y, v, vmax):
    v = min(v, vmax)
    leds = int(v * 5 / vmax)
    for x in range(leds):
        display.set_pixel(x, y, 9)

while True:
    reading = pin0.read_analog()
    display.clear()

    t = trend(reading)
    if t < 0:
        display.show(DISCHARGING)
    elif t > 0:
        display.show(CHARGING)

    bargchart(4, reading, P0_MAX)

    if button_a.was_pressed():
        pin2.write_digital(0) # off
    if button_b.was_pressed():
        pin2.write_digital(1) # on

    sleep(1000)

```

How it Works

The two new images are set up in the constants `CHARGING` and `DISCHARGING`. Each number represents one pixel on the micro:bit screen, 0 is off and 9 is full on. These names are upper case because Python programmers use this convention to remind them that they are constant and shouldn't change throughout the program.

The main program and `barchart()` functions are almost identical to the previous program, so let's just look at the lines in **bold** that have changed in this program, first looking at the main program.

The reading is read from pin P0 as before, and represents the amount of stored charge in the Solar Store. This is passed into the new function `trend()` which works out whether the reading is going up or going down, and depending on the direction of change, it displays either the `CHARGING` or `DISCHARGING` image. If the trend is 0, no image is displayed. Because the display is cleared and re-drawn completely each time round the main loop, if there is no change in the amount of stored charge, neither of these images is displayed.

The `trend()` function is a little bit complex, but basically what it does is to record a history of the last 10 readings from the P0 pin as 'differences'. If the difference is positive then the amount of stored charge has increased since last time, and if it is negative it has decreased since last time. Readings from the P0 pin will jump up and down a little bit naturally due to electrical noise and interference, but what is important is the overall long term trend of the direction of the readings.

Every time `trend()` is called it shifts the numbers left along the list by one (like a big shift register) and keeps a running total. If this running total becomes positive enough, then it indicates the Solar Store is charging, and if it becomes negative enough it indicates it is discharging. Anything else is just treated as insignificant random noise, and ignored.

You can fine-tune the accuracy of this trend analyser by changing the constant `SIGNIFICANT`. A smaller number will make it more sensitive, but it will pick up more of the random electrical noise that will be present at P0. A larger number will be less sensitive. This detector will easily detect the change in charge that occurs when the fan or the Bulb is attached to the output. It might not always completely detect tiny changes in charge that occur using indoor lighting, but it will detect when the sun is shining on the Solar Panel.

Experiments

1. Try different light sources. How bright a light is needed before the CHARGING indicator lights up?
2. When you have a lot of charge stored, press the B button to turn the fan on. How long does it take before the DISCHARGING indicator appears?
3. Leave the fan running for at least 10 seconds then press the A button to stop the fan. How long does it take before the DISCHARGING indicator disappears?
4. Is it possible in bright sunlight with the fan running, to see anything other than the CHARGING indicator? Why do you think this is/isn't possible?
5. Experiment with the `SIGNIFICANT` variable and the number in the `range(10)` line, and see if you can improve the stability and accuracy of the trend analyser.
6. Paste your code into the web editor at <https://python.microbit.org> then press the CONNECT button, then FLASH the code to the micro:bit and finally press the OPEN SERIAL button. You should see numbers printed to the screen every second. Watch the last number on the line change as you switch the fan on and off or shine bright sunlight on the solar panel.

Results

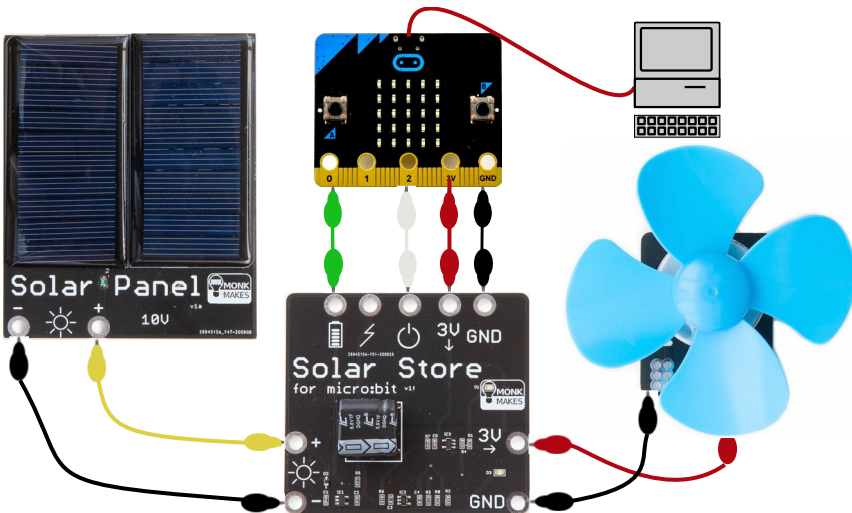
1	
2	
3	
4	
5	
6	

PROJECT 5 – ENERGY LOGGER

In this project you will build a device that can be connected to your personal computer, and it will allow you to draw graphs of energy usage in real time. This will make it easier for you to measure and understand how long it takes to both harvest and consume energy.

You will use this device to record some key measurements about the energy you are harvesting, so that the right design decisions can be made in the final project, based on the amount of light you have around you that can be harvested and stored.

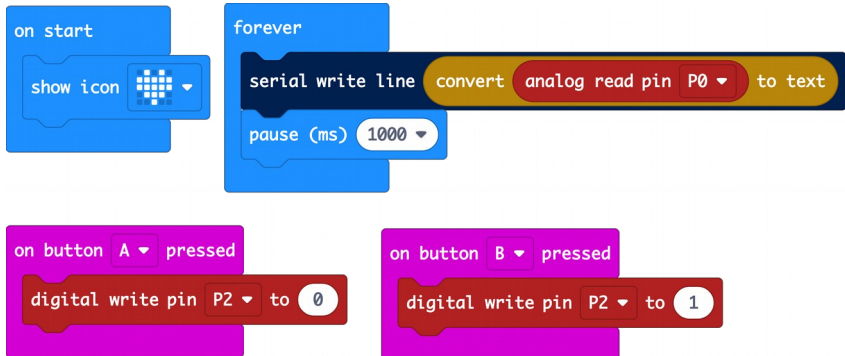
Connecting Up



Code for the Energy Logger (MakeCode)

The code for this project can be accessed here:

https://makecode.microbit.org/_Avq2HMMfp2gp



How it works

The code loops round once per second and takes a reading from the P0 pin. This reading is sent via the serial port, which transfers the data over the USB lead to your computer. A reading will be somewhere between 0 (for 0V) and 1023 (for 3V), and the voltage at the P0 pin will change as the amount of stored charge changes.

Button A and Button B are used to turn the output stage of the Solar Store off and on respectively, so that you can switch between charging and discharging. While you are doing this, you are capturing a history trace of the amount of energy stored in the Solar Store.

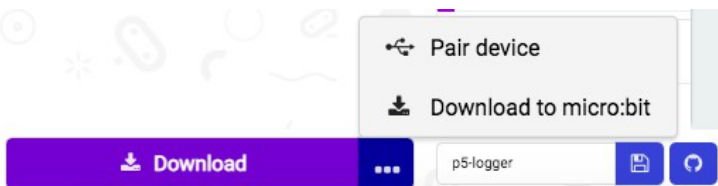
You can use this information to work out how quickly the Solar Store charges and discharges in specific light conditions, and this information will be vital later to correctly configure the final intelligent project.

Connecting to Your PC

MakeCode has a useful feature that allows you to stream live data from a micro:bit, and it will draw a live graph of that data. It also allows you to store the same data in a file that can be loaded and analysed in other programs.

Pairing Your micro:bit With MakeCode

To use the graphing feature, you first have to pair your micro:bit with MakeCode. Click on the *pair device* link on the MakeCode page and follow the on-screen instructions.



Note: If your micro:bit won't pair, you may need to upgrade the firmware stored on its USB interface chip. If so, follow the easy step by step instructions here:

<https://support.microbit.org/support/solutions/articles/19000019131-upgrade-the-firmware-on-the-micro-bit>

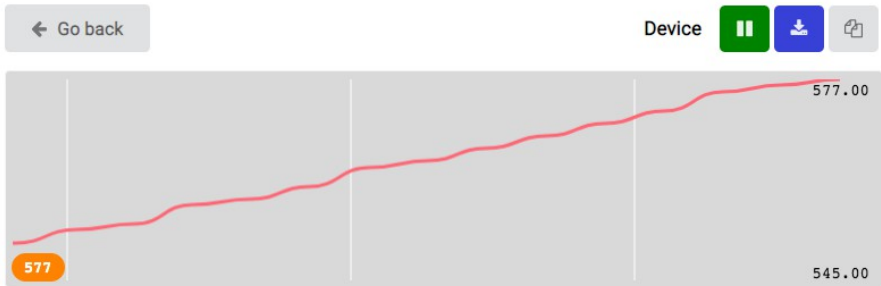
Finally, click *download to micro:bit* to flash the code directly to your micro:bit.

Showing the Device Console Graph

Now that your micro:bit is paired, it can send serial data over the USB lead to MakeCode, and you should see a new *show console device* button.



Press that button and then you should see a live graph of energy data displayed:



Capturing Data From a Charge and Discharge Cycle

In order to work out the key parameters of your system, you need to do a complete charge and discharge cycle.

1. Charge your system up completely by arranging for the Solar Panel to be near a bright light or in sunlight. As the system charges up you will see the graph line in MakeCode increasing and the number at the top right of the graph will get bigger and bigger before it stops. On our system, the charging stopped at about 800.
2. Now discharge the system totally by pressing button B which turns the fan on, and you will see the graph line slowly decreasing and the number at the bottom right of the graph reducing. On our system, the fan stopped at about 218, but it might vary slightly on your system.
3. Because there might have already been some stored charge in your system when you did step 1, re-do step 1 again to fully charge the system, so you get a really accurate set of readings.

You now have a complete data log containing the biggest and smallest stored charge readings (Top of Charge – TOC, and Bottom of Charge – BOC), how long it takes to charge up from your light source, and how long it takes to use all that energy by running the fan.

Downloading Data

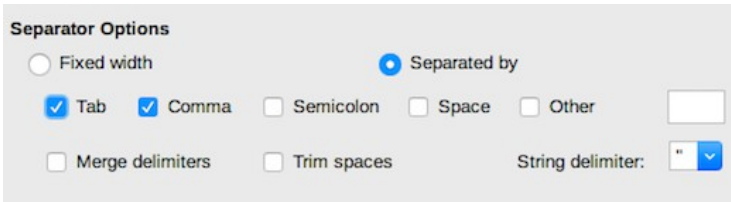
To download the data file, first pause the capture so no more new data comes in, by pressing the red button at the top. Then press the little document icon at the far right of the graph to download the data into your normal downloads folder.



Getting Data into a Spreadsheet Program

You can now open this file with either LibreOffice or Excel. Let's look at how to do that in LibreOffice, which is a free program that can be downloaded from: <https://www.libreoffice.org/> (but the steps in Microsoft Excel will be similar).

Open LibreOffice and choose FILE then NEW/SPREADSHEET, then FILE followed by OPEN from the menu. Find the downloaded microbit-data.txt file in your downloads folder. You will be presented with an import dialog, and the data from MakeCode is separated by tab characters, so tick the 'tab' checkbox then press the OK button to import the file.



Analysing the Data

Now that the data file is inside a spreadsheet you can do all sorts of analysis and graphing of it. The key parameters to calculate are the minimum and maximum readings, as these represent the Bottom of Charge (BOC) and Top of Charge (TOC) parameters of your system, which you will need later.

You can scroll up and down the data in your spreadsheet to find the maximum and minimum readings. A quicker way might be to use the `=MAX()` and `=MIN()` functions of the spreadsheet to calculate these numbers from the range of data in column A for you, like this:

1488	817		
1489	816		
1490	min	217	
1491	max	=max(a1:A1489)	
1492			

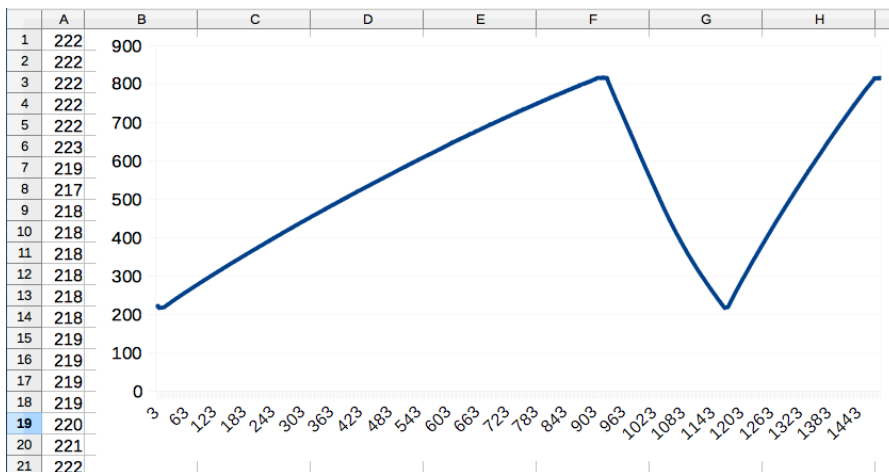
1488	817		
1489	816		
1490	min	217	
1491	max	818	
1492			

To calculate how long it takes to charge your system, look at the row number of the last reading, then scroll back until the reading is back at its minimum value. Because the program logs data once per second, taking away these two spreadsheet row numbers tells you how many seconds it took to charge.

Finally, to calculate how long it takes to discharge your system, collect the row number of the value that was at the lowest value (the end of the discharge cycle), and scroll back until the value is at its highest again (the start of the discharge cycle). Take away these two spreadsheet row numbers, and it will tell you how many seconds it takes to discharge a fully charged system using the fan.

Graphing the Data

A picture is always much easier to understand, so you can also use your spreadsheet to draw a graph of the data to help you visualise the charge and discharge cycles. Select column A in the spreadsheet then use INSERT/ CHART from the menu. Choose a line graph without points and press FINISH, and you will see something like this:



In the above graph, you can see a charge cycle, a discharge cycle, and a second charge cycle. In the experiment that we did, it was a bit shady outside the first time it charged, but the sun came out the second time, which is why it took longer for the first charge compared to the second charge.

Results

Parameter	Value	Note
Bottom of Charge reading (BOC)		Range 0..1023
Top of Charge reading (TOC)		Range 0..1023
Time from TOC to BOC		seconds
Time from BOC to TOC		seconds

Understanding the Data

So, why is all this data important? Remember, the numbers you calculate here will be unique to your environment and lighting conditions.

The duty cycle of a system, refers to how long you are charging for, and how long you are discharging for. So a duty cycle of 50% spends half of its time charging, and half of its time discharging.

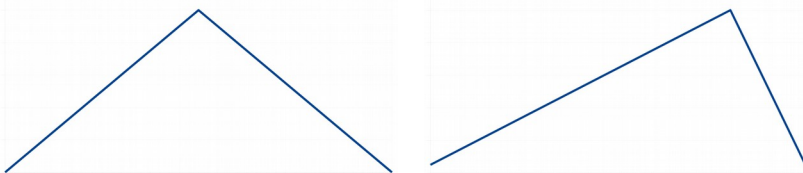
When designing an energy harvesting system, you have to store enough energy in the charge cycle, in order to power your equipment for long enough in the discharge cycle. It is also important to balance the time it takes to charge, against the time it takes to discharge. If it takes 4 hours to charge a system and you can then only use the fan for 2 seconds, that is not a very well designed system.

Think back to the garden light project, as long as there are enough sunlight hours in the day to store enough energy and the bulb is low power enough, the garden light will light the garden path for the whole night time.

When designing a system, the designer can choose bigger solar panels, a bigger energy store, or a more energy efficient piece of attached equipment that uses the stored energy slower. They can also analyse the timing of the system so that there is always enough time to charge the energy store sufficiently, to power the attached equipment for long enough to be useful, before it can be recharged.

Understanding Duty Cycle

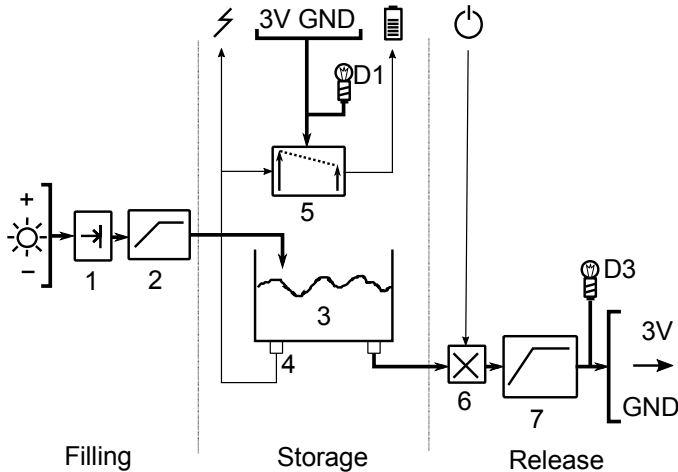
Duty Cycle is a term that describes what percentage of time is spent charging and discharging, and it is important to design a system with the correct duty cycle, so that it always works. If you don't store enough charge in your Solar Store, then your Bulb or fan won't run for long enough. Below is a graph that shows the duty cycle of two different systems:



In the left hand graph the duty cycle is 50%, which means it takes as much time to charge as it does to discharge. In the right hand graph, the duty cycle is 25%, which means it spends 3 quarters of its time charging, and a quarter of its time discharging. Any energy harvesting system can be fine tuned into a well balanced system by changing the rate of harvestable energy, the size of the store, how energy efficient the attached device is, how long it is expected to be charging and how long it is expected to be discharging.

HOW THE SOLAR STORE WORKS

There are a number of components and parts to the Solar Store, and knowing a little bit about how a circuit works can help you to decide how to best use it in your projects. Let's take a look at the three main parts of the board and what they do.



Filling Stage

A voltage from the Solar Panel appears at the board, and a one-way valve (1) prevents damage to the circuit if the panel is wired up the wrong way round. The voltage from the Solar Panel is then 'limited' (2) which converts it from the 10V range of the Solar Panel, to the 5V range expected by the storage system.

Storage Stage

A super-capacitor (3) stores the energy. A sensing circuit (4) checks how much energy is stored, and feeds this to the Level terminal, which can be looped over to the Enable terminal to make the board self-operating without a micro:bit.

The level sensor can reach up to 5V but the micro:bit can only safely sense up to 3V, so the voltage is scaled into the range 0-3V for the micro:bit by scaling circuitry (5). This needs power to operate, which is provided by the two power-in terminals at the top, and the board power LED (D1) also lights up.

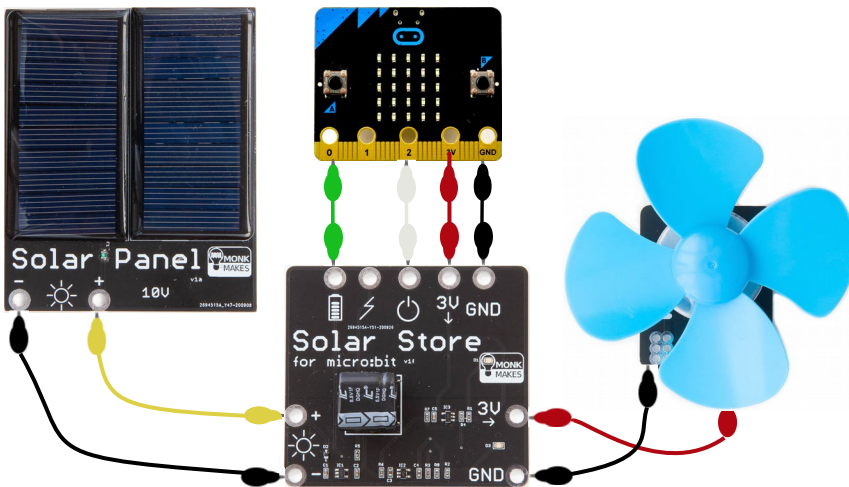
Release Stage

Energy is released by opening the supply valve(6), and this is done by applying a voltage to the enable terminal. Finally, because the voltage might be up to 5V, it is limited(7) down to 3V for the attached circuitry, and also lights LED (D3).

PROJECT 6 – INTELLIGENT COOLING FAN

In this final project, you will use all the measurements you took from the previous project, and build an intelligent cooling fan. This cooling fan will only turn on when things are warming up, and it will turn off automatically when things cool down. As a final modification, you will also add some code that estimates how much longer the fan will run for and display a count-down on the screen.

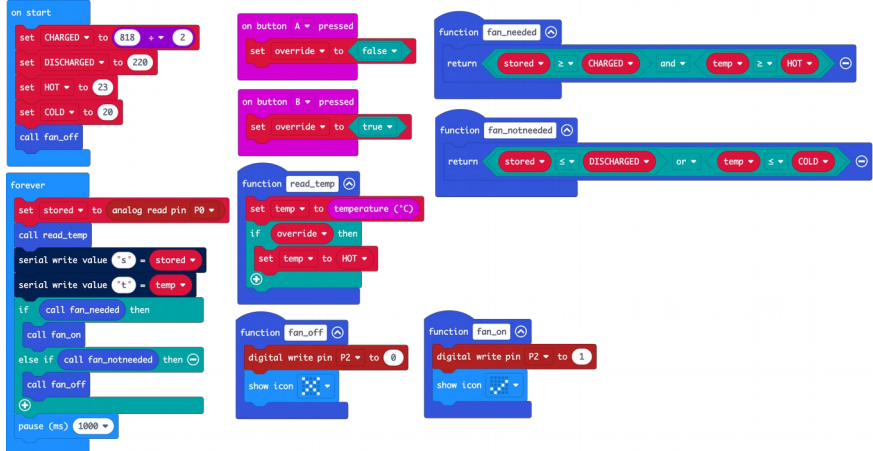
Connecting Up



Code for the Intelligent Cooling Fan (MakeCode)

You can access this code from this link:

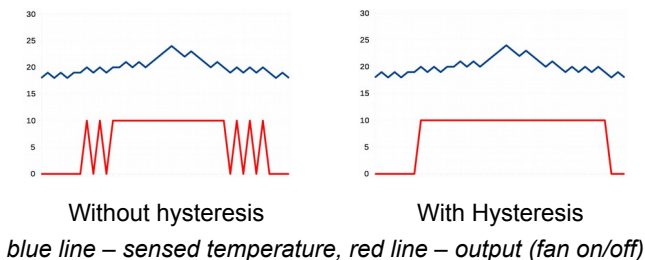
https://makecode.microbit.org/_ATgfJtK9fMa5



How it Works

Using the numbers for TOC and BOC from project 5, fill in the CHARGED and DISCHARGED constants in this program, to calibrate it for your specific environment. Note how the CHARGED constant is divided by two; this is because you want to allow your fan to start if things are getting hot, even when there is some charge in the Solar Store, rather than waiting for full charge.

This program uses *hysteresis* to prevent lots of jittery switching. This just means that there are two thresholds, one for turning on and one for turning off, and the values are slightly different. This is just like how a central heating thermostat works, to prevent repeated boiler starting and stopping. Both the temperature sensing and the stored charge sensing have hysteresis applied to them to stop this jittering.



The main loop of this program senses the temperature and stored charge once per second, and consults two functions `fan_needed` and `fan_notneeded` to decide whether the fan is needed or not. The logic inside these functions could have been used directly without a function, but using a function makes it easier for you to fine tune this detection logic independently of the main program. Both functions return a `TRUE` or a `FALSE` depending on the desired outcome.

The `fan_needed` function decides that the cooling fan is needed only when there is sufficient stored charge in your Solar Store to allow the fan to run for a long time, as well as it being hot enough to need some cooling action.

The `fan_notneeded` function decides that the cooling fan is not needed if the charge is running low, or if it is no longer hot. Notice also that in the main program the `fan_needed` function is called first and the `fan_notneeded` function is called in the 'else' part of the code, and due to the different thresholds checked, this implements the hysteresis shown earlier.

Buttons A and B are provided for an override that allows you to request the fan manually. Press B to override, and press A to cancel the override. There must be enough stored charge before the override will work though.

By splitting the program into little sections like this, you can easily change how it decides if the fan is needed or not, and change what happens when the fan is turned on and off, without disturbing the main program.

Code for the Intelligent Cooling Fan (MicroPython)

This is program p6-intelligentfan.py.

```
from microbit import *

CHARGED = 818/2
DISCHARGED = 220
HOT = 23
COLD = 20
override = False
temp = 0

def read_temp():
    global temp
    temp = temperature()
    if override:
        temp = HOT

def fan_needed():
    return stored >= CHARGED and temp >= HOT

def fan_not_needed():
    return stored <= DISCHARGED or temp <= COLD

def fan_on():
    pin2.write_digital(1)
    display.show(Image.YES)

def fan_off():
    pin2.write_digital(0)
    display.show(Image.NO)

while True:
    stored = pin0.read_analog()
    if button_a.was_pressed():
        override = True
    if button_b.was_pressed():
        override = False
    read_temp()
```

```

if fan_needed():
    fan_on()
elif fan_not_needed():
    fan_off()
print(stored, temp)
sleep(1000)

```

Adding Time Estimation

In project 5, you also calculated how long it takes your system to charge up to top of charge, and discharge down to bottom of charge. These times significantly affect the duty cycle of the system and will limit how long and how often you can use the fan, based on your specific lighting conditions.

The MicroPython code below adds a time estimator based on these numbers, and this will display a number on the screen indicating how many seconds of fan-time are possible with the current charge.

The new/modified lines are marked in **bold**. This is program `p6-estimator.py`.

```

from microbit import *

TOC = 818 # c
BOC = 220

CHARGED = TOC/2
DISCHARGED = BOC
DISCHARGE_TIME = 250
RATE = (BOC - TOC) / DISCHARGE_TIME # m
HOT = 23
COLD = 20
override = False
temp = 0

def remaining(v):
    return DISCHARGE_TIME - (v - TOC)/RATE

FONT = ( # WhaleySans font, 2x5 digits only
("99","99","99","99","99"),
("09","09","09","09","09"),
("99","09","99","90","99"),
("99","09","99","09","99"),
("90","90","99","09","09"),
("99","90","99","09","99"),
("99","90","99","99","99"),

```

```

("99","09","09","09","09"),
("99","99","00","99","99"),
("99","99","99","09","99")
)

def img(n):
    lg = FONT[int(n/10)]
    rg = FONT[int(n%10)]
    c = ""
    for r in range(5):
        c += lg[r] + "0" + rg[r]
        if r != 4:
            c += ':'
    return Image(c)

def digits(n):
    if n > 99:
        display.show(Image.CHESSBOARD)
    else:
        display.show(img(n))

def read_temp():
    global temp
    temp = temperature()
    if override: temp = HOT

def fan_needed():
    return stored >= CHARGED and temp >= HOT

def fan_not_needed():
    return stored <= DISCHARGED or temp <= COLD

def fan_on():
    pin2.write_digital(1)
    #display.show(Image.YES)

def fan_off():
    pin2.write_digital(0)
    #display.show(Image.NO)

```

```

# main program
while True:
    # sensing
    stored = pin0.read_analog()
    if button_a.was_pressed():
        override = True
    if button_b.was_pressed():
        override = False
    read_temp()
    # control
    if fan_needed():
        fan_on()
    elif fan_not_needed():
        fan_off()

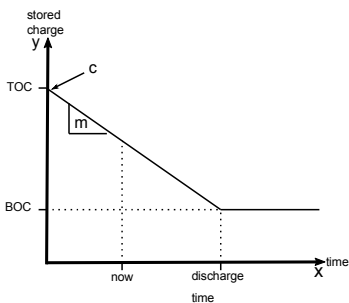
    # display
    digits(remaining(stored))
    print(stored, temp)
    sleep(500)

```

How it Works

This is a really simple estimator that makes a number of quite broad assumptions about the system, but it is good enough as a rough indicator to the user of how long they can use the fan for.

To understand how the estimator works, we need to look at a bit of school math, namely the equation of a straight line. A graph will help to explain all the key parts of the estimator.



The equation of a straight line is:

$$y = mx + c$$

y = stored charge (read from P0)

m = slope of line (rate of use of charge)

x = time fan has been on for

c = TOC (charge when fully charged)

This graph shows the discharge cycle that you captured earlier in project 5. When the Solar Store is fully charged it is at the Top of Charge (TOC). Turning on the fan then consumes the energy in the Solar Store, and this powers the fan and slowly discharges the Solar Store, as shown by the downwards diagonal line.

You already know what the readings are for the Top of Charge (TOC) and Bottom of Charge (BOC), and you previously timed how long it took your fan to use all the energy from TOC to BOC (discharge time). These points are labelled on the graph.

The time left until the fan goes off (which is when the charge reaches BOC) is the difference between the time *now* and the total *discharge time* of the system.

So, if we can pump the stored charge reading from P0 into this graph and work out the distance from *now* until *discharge time*, that will give us an estimate of how many seconds are left before the fan will switch off.

Using a little bit of school math, we can re-arrange the equation to put *x* on the left hand side. Subtract *c* from both sides, and divide both sides by *m*, and we get:

$$x = (y - c) / m$$

Realising that this is the time we have been discharging for, and that we need the time remaining, take away *now* from *discharge time* to get the final equations used in the Python code:

```
TOC = 800 # c
BOC = 40
DISCHARGE_TIME = 300 # secs from TOC to BOC (with fan)

RATE = (BOC - TOC) / DISCHARGE_TIME # m

def remaining(v):
    return DISCHARGE_TIME - (v - TOC)/RATE
```

The final bit of code to mention is the display of this number. Because the number of seconds of run time remaining for a full charge could be a multi digit number of seconds, it would be quite hard to read a number that repeatedly scrolls across the screen.

So, this code uses a special 2-digit font that will show the number 00 – 99 on the screen. Anything more than 99 seconds is a long time, so a checkerboard image is displayed if the number is quite big.

The stored charge and temperature are also displayed via the serial console twice per second, so press the *open serial* button in the Python editor to see the numbers, if you want to watch the algorithm working.

TROUBLESHOOTING

The Bulb is not lighting – make sure you have enough charge stored in the Solar Store, as the Bulb has a minimum turn-on voltage. Also make sure you have connected the Enable signal of the Solar Store. Check that the Bulb is connected the right way round.

The fan is not turning – the motor used in the fan is quite low power, but it is not as low power as the bulb. If you have checked all the advice above about the Bulb and it still does not turn the fan, try harvesting more energy into the Solar Store before you enable the fan.

The charge% reading never changes – make sure you have connected the 3V and 0V pins at the top of the solar charge board to the associated pins on the micro:bit.

The Solar Store is taking a long time to charge – try to find a brighter light source, such as going outside and using full daylight or sunlight. Also try different types of lighting indoors, LED lighting and tungsten/halogen lighting provide different amounts of harvestable energy.

The Solar Store doesn't work very long – the Solar Store can only store a limited amount of energy before it needs recharging, and it won't store large amounts of energy like a battery might. Also if the equipment you attach draws more power then it will discharge much more quickly. The secret to getting the best from an energy harvesting system is to arrange for enough time to harvest the energy, and use low power equipment attached to the output pins.

The micro:bit sometimes ejects the drive when running for a long time – use a shorter USB cable.

The console/graph in MakeCode isn't working – check that you have the latest firmware loaded into the USB interface chip of your micro:bit by following the instructions on this support article:

<https://support.microbit.org/support/solutions/articles/19000019131-upgrade-the-firmware-on-the-micro-bit>

My Solar Store lasts forever – if you are in very bright sunlight, it is possible that there is enough harvested energy to simultaneously charge your Solar Store and power your fan. Try the experiment in the shade, if this is the case.

You can also get general support about the micro:bit device from the Micro:bit Educational Foundation support pages here: <https://support.microbit.org>

ABOUT THE AUTHOR

David Whale is an embedded software engineer and a STEM ambassador who volunteers in schools in the UK. He has been an active member of the micro:bit community since its inception. He contributed to the original BBC micro:bit project, advising The IET and BBC and helping to write and deliver training courses to teachers around the UK. He helped to form the Micro:bit Educational Foundation and was their technical support engineer for the first 2 years, where he was given the honorary title of 'Micro:bit Wizard'. He continues on his life-long mission to inspire the next generation of engineers and scientists.

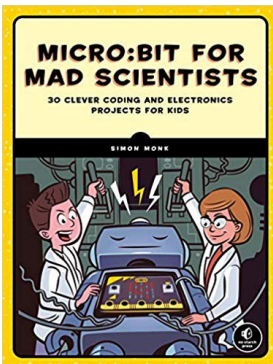
LEARNING

micro:bit Programming

If you want to learn more about programming the micro:bit in MicroPython, then you should consider buying Simon Monk's *Programming micro:bit: Getting Started with MicroPython* book, which is available from all major book sellers.

For some interesting project ideas, you might also like *micro:bit for Mad Scientists* from NoStarch Press.

You can find out more about books by Simon Monk (the designer of this kit) at: <http://simonmonk.org> or follow him on Twitter where he is @simonmonk2



MONK MAKES KITS

For more information on this kit, the product's home page is here:

https://monkmakes.com/mb_solar

As well as this kit, MonkMakes produce all sorts of kits and gadgets to help with your maker projects. Find out more, as well as where to buy here:

<https://www.monkmakes.com/products>

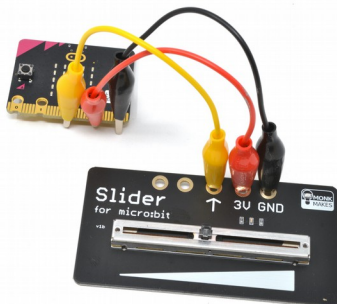
You can also follow MonkMakes on Twitter @monkmakes



Electronics Starter Kit 2 for micro:bit



Air Quality Kit for micro:bit



Slider Kit for micro:bit