SPICO Clock Project

Background

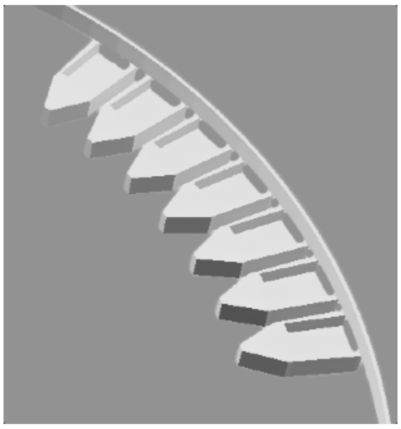
This project is a further development of my Pico Clock. Having built it I decided that it really needed a PCB. Having decided to design one I realised I could save space by ditching the stepper motor driver cards by mounting the driver chips directly and replacing the light sensor card with a simple LDR. This meant that the PCB could take the place of the driver cards and light sensor at the bottom of the case leaving a space where the Pico had been at the top adjacent to the OLEDs. This provided a space for an Adafruit FX card for sound.

Components required for SPICO

1 x PCB
1 x Adafruit FX with 2watt amplifier (Pimoroni)
2 x 28BYJ-48 stepper motors and driver boards (eBay)
2 x 0.96 I2C OLED ssd1306 (eBay)
2 x mini oval 8ohm 1watt speakers (Pimoroni)
1 x DS18B20 one wire thermometer (eBay)
1 x 60 lamp NeoPixel ring (Cool Components)
1 x LDR
1 x LED
1 x 4.7k resistor
1 x 330 ohm resistor
1 x 10mF 25v electrolytic
2 x 100pF ceramic
(1 x slide switch for AP version)

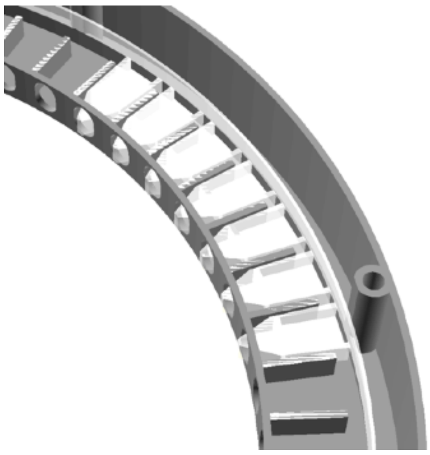
3D Printing

This render shows a a section of the light guide which fits snuggly over the NeoPixels.



Having found that the 'glass' PLA seemed more rigid than the normal stuff I decided to use it for the gears, gear tracks and motor mount. More by luck than judgement this resulted in the clock hands being like frosted glass which could be lit from their centre with an LED.
The only problem with the light guide as designed was that there was a lot of light spill from one light to its neighbours. This was partially solved by including small blanking plates in the front cover.

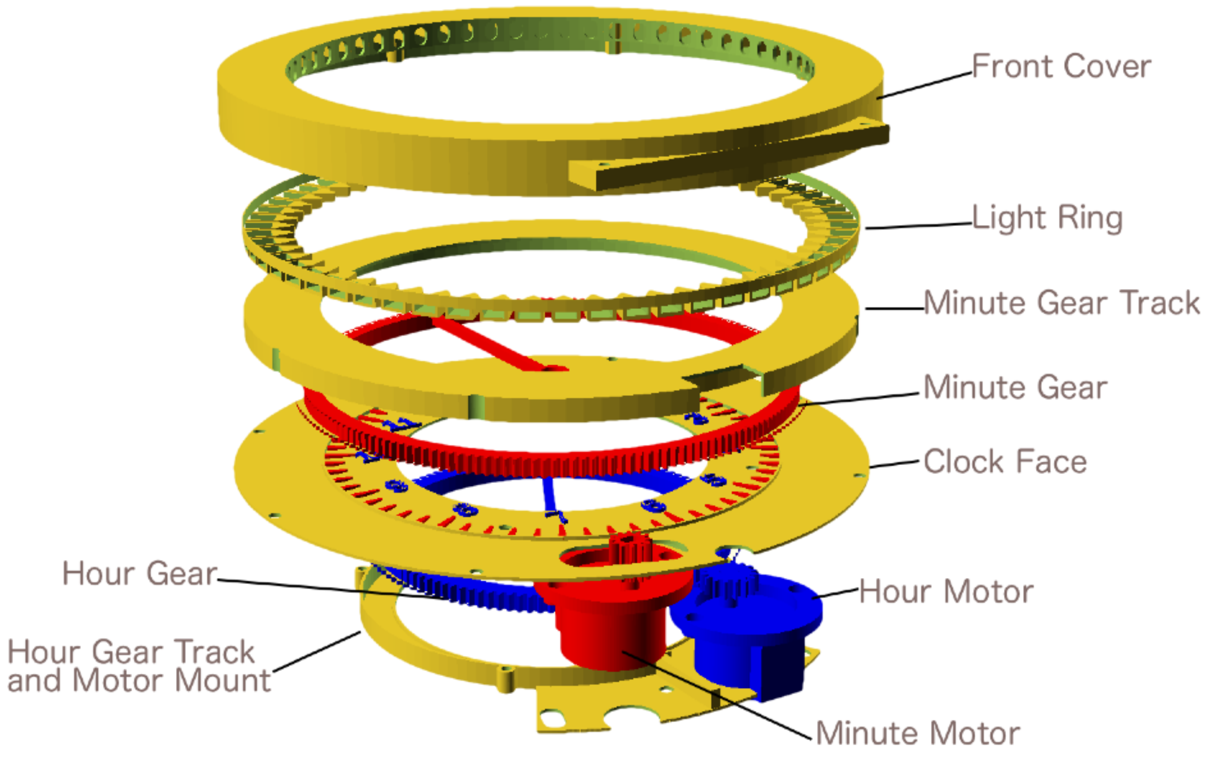
This render shows a section of the light guide fitted into the front cover.



The next problem to solve was the gearing. The motors required 512 full step pulses for a complete revolution which after some thought lead me to design a minute gear with 180 teeth. This meant that ¼ turn of a 12 tooth drive gear would turn the minute gear through 3 teeth. This gives 180/3 = 60, i.e. 1 minute. A quarter turn of the motor requires 512/4 = 128 full step pulses for 1 minute of movement. A similar sum resulted in an hour gear of 120 teeth with a drive gear of 16 teeth and a ⅛ turn of the motor requiring 64 full step pulses per minute of movement.

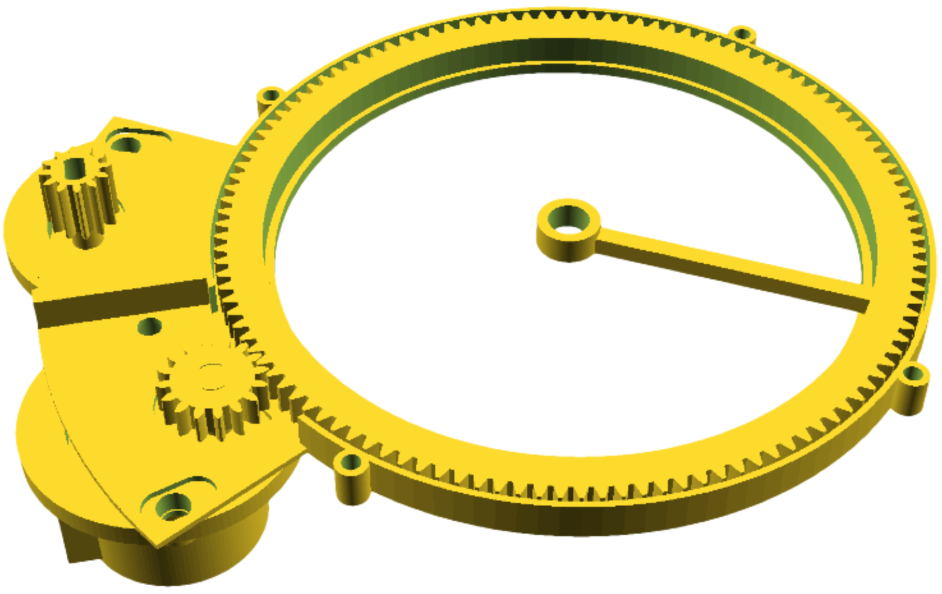
It may be obvious by now that the hands move 1 minute at a time i.e. 6 degrees. Once these gear ratios were fixed their diameter was adjusted by changing the gear module value (minute gear 0.8 and hour gear 0.9). Diameters, of course, were governed by the NeoPixel ring.
During initial experiments and breadboarding of the design I decided to include 3 OLED displays which I already had. I considered one was necessary to show WiFi connection status on power up, and the others… why not? I also had a DS18B20 one-wire digital thermometer, the light sensor and a bright white LED to illuminate the hands from their centres. As the design progressed I couldn't really see how 3 OLEDs could be accommodated without overwhelming the clock face and I also couldn't think what an extra OLED would offer as regards information. Another consideration was that my OLEDs were SPI, 7 wire, devices that required a lot of interconnect. During the early breadboarding I swapped to 2 x I2C 4 wire OLEDs but these were slower and messed up the program timing. I'm not sure whether this problem was real or imaginary but I changed back to SPI.

Plastic Components



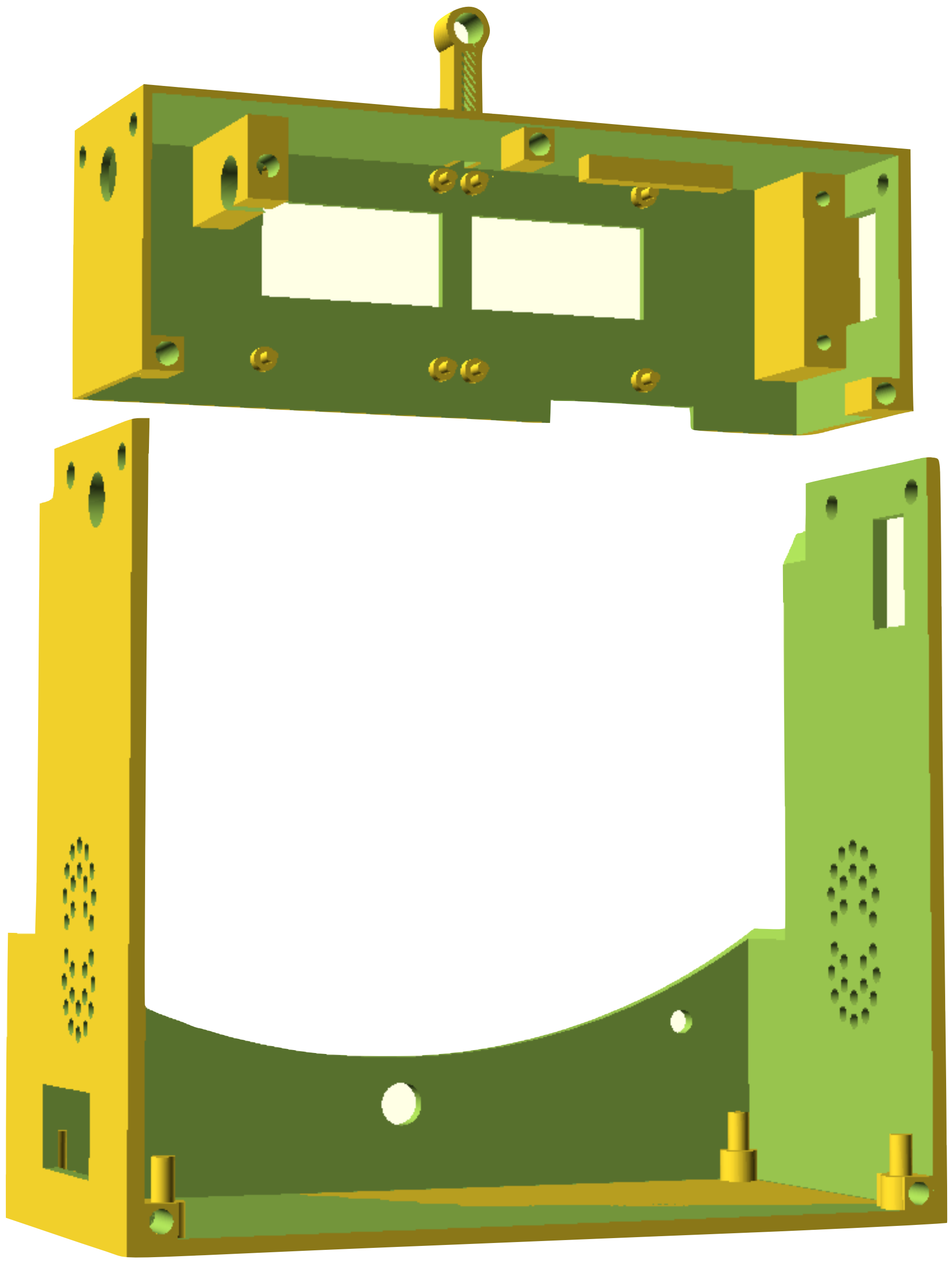
Motor Mount

This shows the back plate with the motors and the hour gear in place. Note the adjustable mount for the motors which are also raised (lowered in this picture) with plastic spacer rings.

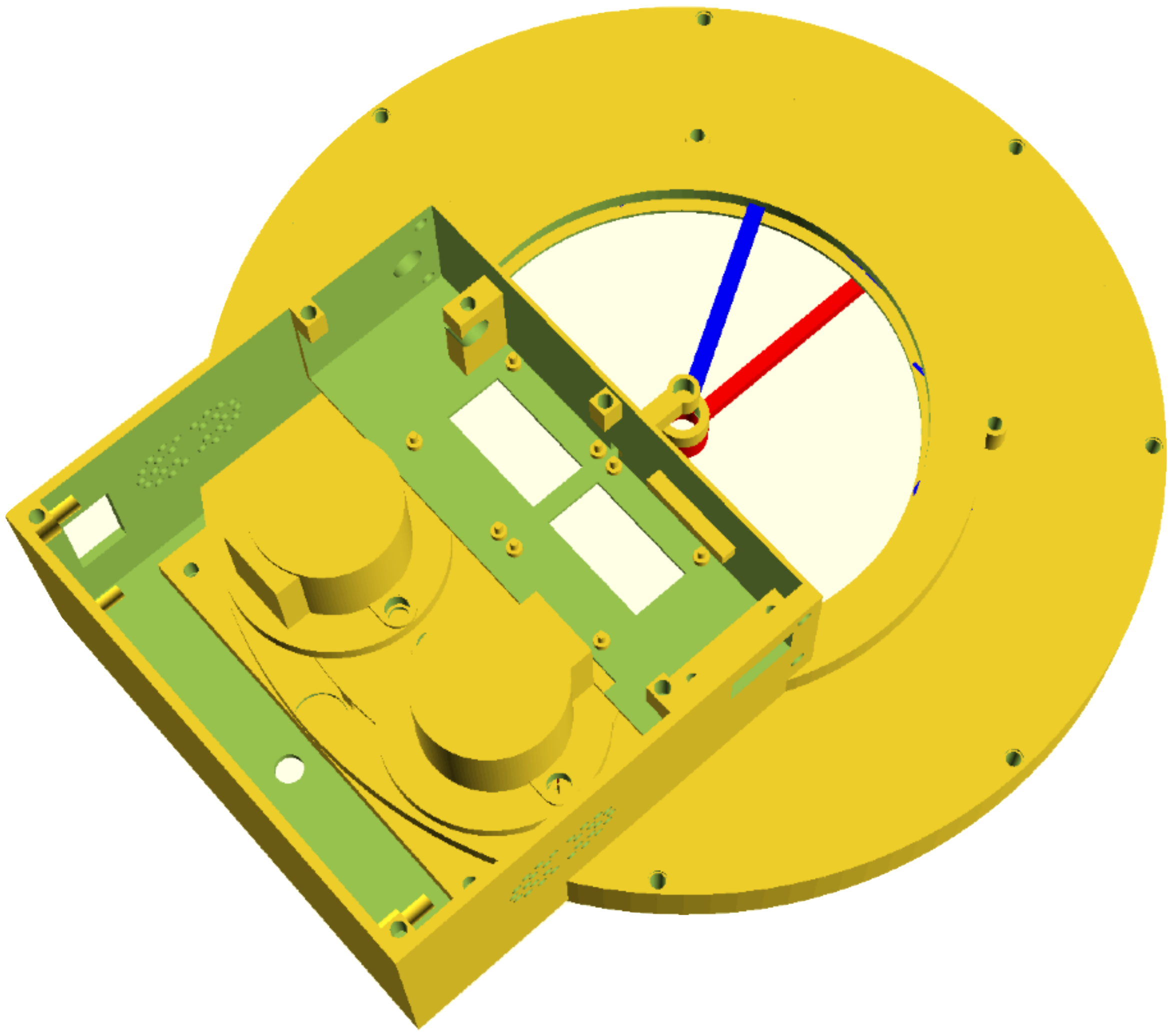


Case

This is printed in two parts for ease of printing. Also I chose to print the top section in black to match the OLEDs.



Case with clock mounted



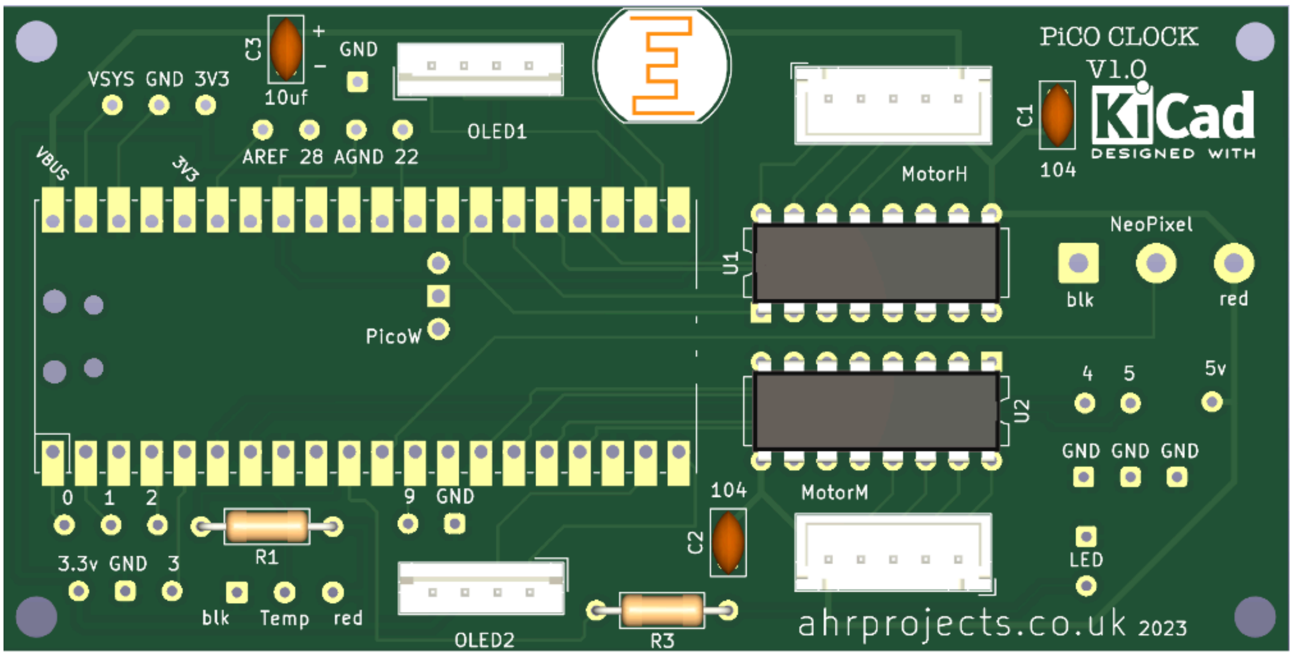
Besides having space for all the components the case needs to give access to the Pico so that a USB cable can attach to it. The lugs on the bottom hold the PCB. The lugs around the OLED windows hold the OLEDs in place. The hole on the right side (as shown) is for the USB connection to the Adafruit FX card which screws onto a small shelf. The hole on the left and the adjacent bracket is for the temperature sensor. The hole at bottom centre is for the light sensor (LDR) which has to be mounted on the PCB with sufficient wire length to allow for bending at right-angles. The rectangular hole bottom left gives access to the Pico for program modification and power in normal use. It's intended that the lugs are melted with a soldering iron to fix components in place, but it wasn't necessary, in my case, to melt the PCB lugs as the PCB mounting holes had to be opened up a bit for a snug fit.

Clock Construction Notes

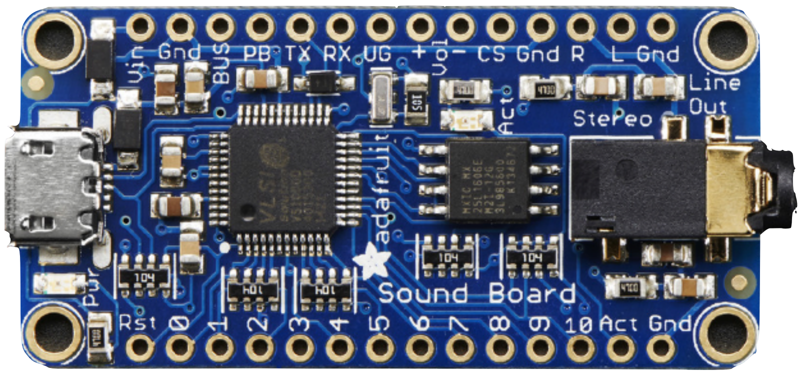
All parts are screwed together with 5mm long domed hex head M3 screws. These self tap into the plastic 2.8mm holes as printed. Most parts have been designed to lay fat on the bed with no support with the exception of the front cover which has an inset for mounting to the case. The case itself also has some support requirements.
The gears may have to be dressed once printed particularly if there is first layer spread. The better this is done the smoother they will run. The drive spur gears are a tight fit on the motor shafts and will need relieving. The minute and hour hand guides should have any spurious bits of plastic removed. And to ensure smooth running the gears and guides benefit from an application of PTFE spray. The clock face numbers and minute markers are embossed for easy painting.
The light ring is fiddly to insert into the front cover. I accidentally broke the outer ring while trying to do this which made it much easier to manipulate. The trick is to feed it in at an angle making sure to align the next two or three sections as you go. It’s a very satisfying fit once the circle is completed and clicks nicely into place.

The PCB

Below is a render from KiCad showing all components mounted except for the pico-w.
The stepper driver ICs were simply unplugged from driver boards included with the motors. The 5 pin motor sockets were unsoldered from the driver boards and reused. New sockets were used for the OLEDS. The NeoPixel ring comes with input and output wires terminated in a plug and socket. The output wires with their plug were unsoldered and connected to the NeoPixel pads on the PCB. This gave a plug/socket connector to the NeoPixel ring. The centre illuminating LED connects to the two pads marked LED. The resistor R3 is 330 ohms. The Adafruit FX card is a bit more involved and I wish I'd put a socket on the PCB for it. However all it's connections are in the bottom left corner (as displayed here).



The first 2 connections on the FX card are 'Vin' and 'Gnd'. These are connected to the '3.3v' and 'GND' pads on the PCB. The 'UG' pin must be linked to 'Gnd' on the FX card to enable UART control. Then the 'RX' pad is connected to pad '0' on the PCB. The 'Act' pad on the FX card is connected to pad '1' on the PCB and the 'Rst' pad is connected to pad '2 'on the PCB. The FX card shown below is the headphone version. SPICO uses the amplified version, but the connections are the same.


Vin Gnd BUS PB TX RX UG + Vol − CS Gnd R L Gnd
Line Out
StePeo
Act
adafruit
Sound Board
Pwr
Rst 0 1 2 3 4 5 6 7 8 9 10 Act Gnd

The thermometer is a DS18B20 one wire device which needs a pull up resistor. The connections for this are ‘blk’, ‘Temp’ and ‘red’ on the PCB. The pull up resistor R1 is 4.7k. The OLEDS, as sourced, were 4 wire I2C devices with connector pins soldered in. In my case these were removed and four wires were connected directly to the OLED pads. The 4 pin plugs I used had wires attached, so this worked for me. However, the sockets used for the motors and OLEDS must have 2.5mm (0.1 inch) spacing and are often supplied in plug/socket pairs.
I found that the wires to OLED2 should be kept fairly long to enable easy assembly. Check that everything fits on the plastic lugs and the holes for the thermometer are round before attempting a final build. The OLEDs can be fixed in place once enough confidence is gained that everything fits. A 5 wire cable is needed between the FX unit and the PCB. I had some available, but ribbon cable could be used. I found that leaving the top part of the component housing unfixed during assembly of the electronic components made things easier. The FX card should also be left unattached until the clock is set in the component housing. The screws to hold the FX card in place must be chosen and checked. The mounting holes may need drilling out to suit. There is an access hole for the FX card USB connector so that changes to the sound files can be made after assembly.

Software Description

This is written using Micropython which is a subset of Python 3.

The basic program loops around checking the ambient light value and waiting for asynchronous client connections.

Before the loop starts the Pico attempts to connect to a local WiFi service. If this fails after 10 attempts the program stops. If it connects successfully a call is made to an NTP server to get the current time. If this fails the program stops.

If the time request from the NTP server is successful it's written to the Pico's real time clock, but regardless of this success an asynchronous client server is started. If successful however a one second repeating timer is created, light mode is set and current temperature is recorded. This is the current working state of affairs.

The program loop gets interrupted by the one second timer. This timer plays the relevant prevailing sound file (usually 'tick-tock') and checks the real time clock for changes. The seconds value will, of course, always change and the NeoPixel array and OLED displays will change accordingly, but the minutes and hours may stay unchanged. If the seconds value is 30 then the temperature is re-recorded. If the seconds value is 0 then the minute motor is activated to move the minute hand on one minute and if the minute value is a divisible by 12 the hour motor is activated to move a one minute segment. If speech is required for the current minute then that is initiated. A lot happens every time the one second timer fires so the software has been developed to make things happen in parallel. The NeoPixels, motors and FX card messaging are all coded using PIO state-machines, but even so things still stutters a bit when the zero seconds value is hit. In the first Pico Clock threading was used to drive the motors but that has been abandoned in favour of PIO.

During the program loop the client server gets a look in and checks for client connections.
If a client connects it's sent a web page containing various forms. The client server then waits for requests from the client, but hands back control to the clock until the next sleep.

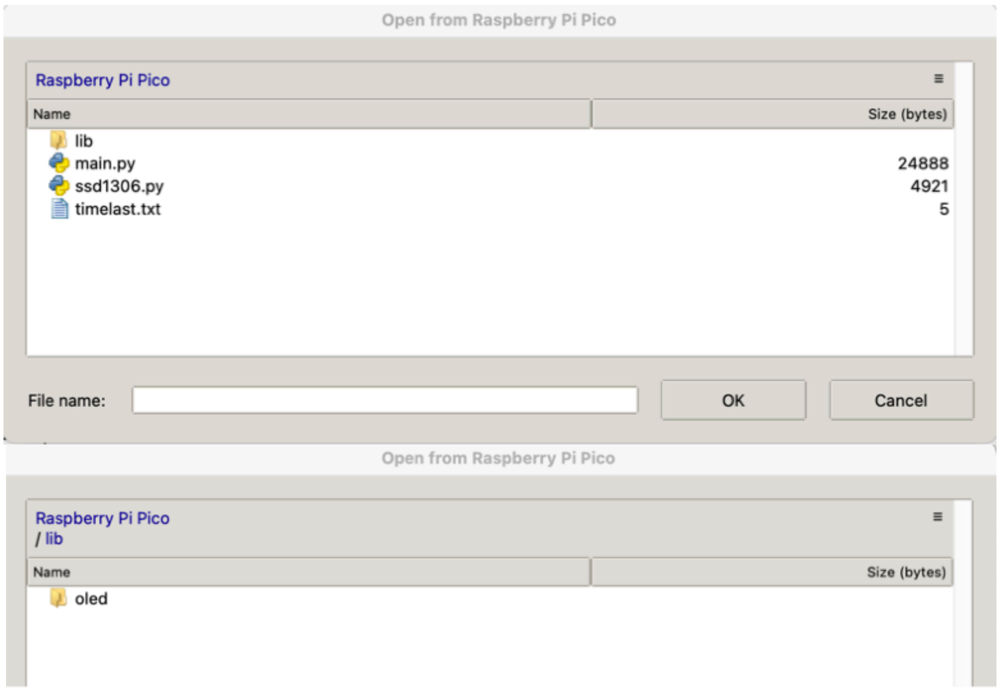
Hand synchronisation is assured by writing the current hand locations to a file every time they change. At initial power on SPICO waits for a full minute to start before synchronising the hands. This ensures that they will be in the right place before the following minute starts. The program calculates the best direction to move the hands to achieve this.

The Sun and Moon values are calculated from the current year day. Attempts to get this information from the Internet were abandoned once algorithms were found that gave reasonable results.

If no PCB is available then the software, as posted, could be picked over for any project using stepper motors, sound, NeoPixels, temperature/light sensors and OLEDs, or any permutation of these. The PIO stepper driver caused me problems, as my first attempt lost steps. In the event it turned out that the order of pushing data onto the FIFO made a difference. The step count had to precede the step pattern. No doubt somebody with a full understanding of PIO state machines will know why. Also I suspect that Micropython isn't the best way to delve the depths of the Pico as my attempts at threading to core 1 refused to behave alongside a repeating timer interrupt.

Setup

Before the software can be run on the Pico the Micro-python environment must be set up. This is a case of downloading the latest UF2 file from e.g. https://www.raspberrypi.com/documentation/microcontrollers/micropython.html and loading onto the Pico. There are also some extra modules that are needed.



Name	Size (bytes)
lib	
main.py	24888
ssd1306.py	4921
timelast.txt	5

Name	Size (bytes)
oled	

Shown above are the top directory and the /lib directory.The oled directory loaded into /lib and ssd1306.py shown here at the top level, but could also reside in /lib. These files are included in the software download but must be placed using Thonny IDE. The main.py file is the program itself, and timelast.txt file is written by the program and records the last position of the hands and the state of BST.

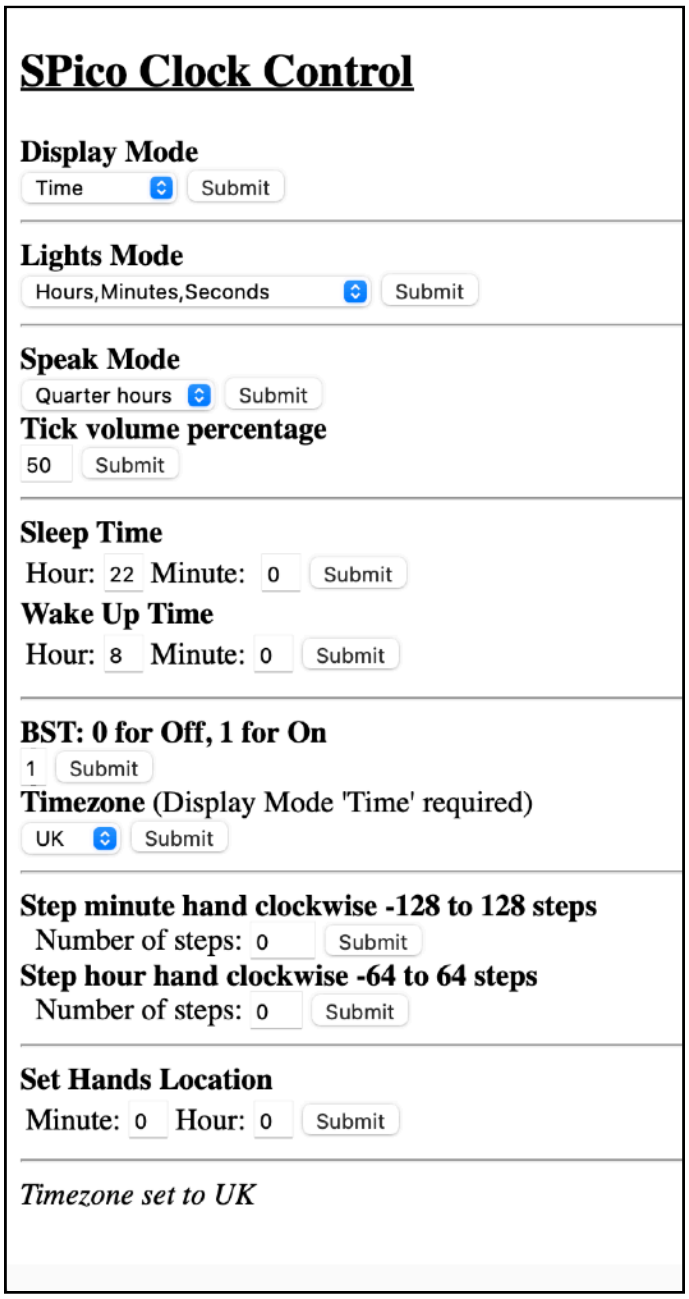
Commissioning

Your WiFi SSID and PASSWORD are configured by setting GPIO pin LOW. This is what the AP switch does in Version 7. On start-up a configuration web page is accessed on address '192.168.4.1' by first connecting to WiFi SSID 'SPICO'.

There is a routine hashed out just before the main loop for setting the hands. I find this useful when commissioning, but don't forget to hash it out once run. The web browser interface can also be used to set the hands before a switching off and restarting.

Web Page

On connecting a browser to the clock's address a web page is loaded as shown below. (This is now updated in Version 7)


SPico Clock Control

Display Mode
Time [Submit]

Lights Mode
Hours,Minutes,Seconds [Submit]

Speak Mode
Quarter hours [Submit]
Tick volume percentage
50 [Submit]

Sleep Time
Hour: 22 Minute: 0 [Submit]
Wake Up Time
Hour: 8 Minute: 0 [Submit]

BST: 0 for Off, 1 for On
1 [Submit]
Timezone (Display Mode 'Time' required)
UK [Submit]

Step minute hand clockwise -128 to 128 steps
Number of steps: 0 [Submit]
Step hour hand clockwise -64 to 64 steps
Number of steps: 0 [Submit]

Set Hands Location
Minute: 0 Hour: 0 [Submit]

Timezone set to UK

This displays several submit forms which are self explanatory and offer NeoPixel display options, Minute and hour hand nudge values, initial hand location setting, BST status and timezone setting, etc..
These are my selection of what seemed useful and can be easily changed in the program.
The nudge and hand setting forms are very useful when commissioning and correcting the clock.
The BST setting enables the overwriting of the programs ‘on’ state and the new state is stored along with the hand locations. This change immediately moves the hour hand but doesn’t correct the display until the next minute is entered. When the clock is restarted BST goes back to its ‘on’ state regardless, which requires a couple of minute cycles to sort the display out.
Set timezone drops down a list of timezone names which are stored in an array in the program along with their shift from UK time. This has no effect on the hands, just the time of day display. The time of day in words stays aligned to UK time.
Speak modes are controlled from a drop down list with wake-up and sleep times configurable. These set the times that the clock will be silent. The drop down lists defaults are aligned with the programs startup conditions.

Sound Files

The Adafruit FX card with 2 watt amplifier comes in two versions, 2Mbyte and 16Mbyte. SPICO uses the 2MByte version, which holds all the sound files used with a bit to spare. There are, of course, limitless possibilities with regard to sounds the clock might make given enough memory space.
The FX card has several configuration options. It can be triggered via its GPIO pins or sent instructions via UART. The latter is used on SPICO. The UART uses a PIO state machine with transmit only. The FX card understands both files by name and files by number (order stored). Files called by name proved much slower than by number so number is used. This means that files must be loaded onto the FX card in the correct order and this only works correctly under MS Windows. MacOS insists on writing a ‘BIN’ file which messes things up.

The WAV files called by the software, as posted, must be stored on the FX card in the following order:
it’s one, It’s two, it’s three, it’s four, it’s five, it’s six, it’s seven, it’s eight, it’s nine, it’s ten, it’s eleven, it’s twelve, five, ten, fifteen, twenty, twenty-five, thirty, thirty-five, forty, forty-five, fifty, fifty-five, o’clock, beep, tick.

Version 7 software

Since the initial release of SPICO its development continues.
There were several issues with the original design mainly to do with configuration.

1. A switch has now been added to pull GPIO 9 low which puts SPICO into Access Point mode. When in AP mode SPICO offers out a web page allowing selection of a WiFi router for subsequent connection and configuration of Latitude/Longitude location.
2. The configuration values set via the client mode web page are now stored in a file which is loaded on start-up. Preferences are maintained over power cycles.
3. The OLED display data has been modified for better use of the available space. If the AP switch is activated during normal running the current WiFi client address is shown.
4. The way alternative timezones are handled is now configured as a ‘name’ and ‘hours difference’ value rather than a selection form a list.

Some of these changes result in additional files stored on the pico.

Hardware changes

Modifications have been applied to the STL files.

1. The temperature sensor hole has been enlarged to provide easier assembly.
2. A mounting point and access hole have been added for the AP switch.
3. The PCB mounting studs have been shortened for easier assembly.
4. The back cover has been modified for an improved fit.