Table of Contents

Overview 3
  • Parts
  • Other Parts and Tools

Hardware 6

Construction 9
  • The Printed Pieces
  • What If I Don’t Have A 3D Printer?
  • Touch pads

Code 19
  • Setting up
  • Playing Sound Files
  • Displaying Images
  • NeoPixel Effects
  • The State Machine
Overview

Now that Halloween is (soon to be) past, what are you going to do with that HalloWing that you used in a project to scare trick-or-treaters with? Well, if you have a black cat left over from the night's entertainments (actually, any sort of cat will do, the author's just happens to be black because, well, black cats are awesome) then you can make use of the HalloWing and keep the cat entertained with this project.

Parts

Adafruit HalloWing M0 Express
This is Hallowing..this is Hallowing...
Hallowing! Hallowing! Are you the kind of person who doesn't...
https://www.adafruit.com/product/3900
PIR (motion) sensor
PIR sensors are used to detect motion from pets/humanoids from about 20 feet away (possibly works on zombies, not guaranteed). This one has an adjustable delay before firing (approx... 
https://www.adafruit.com/product/189

Mini Oval Speaker - 8 Ohm 1 Watt
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/3923

NeoPixel Ring - 24 x 5050 RGB LED with Integrated Drivers
Round and round and round they go! 24 ultra bright smart LED NeoPixels are arranged in a circle with 2.6" (66mm) outer diameter. The rings are 'chainable' - connect the...
https://www.adafruit.com/product/1586

You'll need two of these JST cables. Male or female, it doesn't matter since you'll be clipping the ends off.

Any color of the silicone wire will do. It's great stuff to work with so you might find you want all the colors.

Other Parts and Tools
1 x **Hookup wire**
Silicone Cover Stranded-Core Wire - 50ft 30AWG

1 x **Flexible ribbon cable**
Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long

2 x **3-pin JST cable**
JST PH 3-Pin to Female Socket Cable - 200mm

1 x **Micro-B USB plug**
USB DIY Connector Shell - Type Micro-B Plug

1 x **USB Micro-B Breakout Board**
https://www.adafruit.com/product/1833

1 x **Battery**
Lithium Ion Polymer Battery - 3.7v 1200mAh

2 x **20mm x 80mm prototyping board**
We use this to make touch pads.

4 x **M3 bolt**
M3 x 4mm button head hex socket bolt

2 x **M2 Bolt**
M2 x 4mm button head hex socket bolt

1 x **Pinvise and small drill bit, this is one example**
Pinvise hand drill with a bit to fit 30AWG silicone covered wire.

- Hot glue & glue gun
- Solder and soldering iron

You may also need something to give weight to the very bottom of the bowl so that it will always return to an upright position. Something that won't present a danger of shorting out the electronics and can be glued into the bottom.
Because we are building on the HalloWing and using hardware that is directly supported by it, the electronic aspect of the projects mostly consists of plugging things into the HalloWing: speaker, PIR, NeoPixels, and a battery. Two additional things are the touch pads and a USB extension.

Since the HalloWing has 3-pin JST connectors for the NeoPixels and PIR, we can clip and tin the ends of two 3-pin JST plug/cable assemblies. One can be connected directly to the PIR sensor (after carefully removing the connection pins and cleaning the holes with desoldering braid).

If you’re less than comfortable with this, be sure to get JST cables with female ends that you can just plug onto the PIR’s pins.
The other JST plug/cable will be used for the NeoPixel ring, but not until the physical assembly.

Now make the USB extension. You will want to have the USB connection available for recharging and reprogramming. Separate out about 14cm (5in) of the silicon ribbon cable. Separate it into 2 5 wire pieces. We'll only need one. Strip and tin about 1mm (1/16in) of both ends of each strand. Solder one end to the USB micro-B breakout connections, and the other to the USB micro-B plug. A quick way to ensure you have the connections right is to plug the plug into the breakout and use a continuity checker to find the common pairs. Connect those pairs. When soldering to the breakout, don't put the wire ends through the holes. Instead, solder them flat onto the pads. See the photo below.
You can also use a pre-made extension cable if you don't want to DIY

Panel Mount Extension USB Cable - Micro B Male to Micro B Female
Check out this handy MicroUSB extension cable, which will make it easy for you to enclose a device that has a B type (micro USB host) port. Great if you need to extend the USB...
https://www.adafruit.com/product/3258

The final real bit of soldering is to make the touch pads. The design was made to use double-sided 20mm x 80mm prototyping board with plated through holes. Solder a strand of thin wire (wirewrap wire works well) so as to connect all the holes on one side of the board. Clean up the other side with desoldering braid to remove any excess solder. Then clean off any resin. Isopropyl alcohol works well.

Now on to the physical construction.
Construction
The housing is composed of two pieces that snapfit together. The bottom is simply a hollow hemisphere with a ring around the top that snaps into the cover.

The cover is where everything mounts. On the inside you can see the mountings for the PIR and HalloWing as well as the cutouts for the sensor dome and screen. The place for the speaker (which is self-adhesive) and the speaker grill are also visible.

On the outside face you can, again, see the cutouts for the PIR dome and HalloWing screen as well as the speaker grill. Additionally, there are the insets for the touch pads and NeoPixel ring.

Print with supports from the print bed. The bowl should be printed open side down, while the face should be printed with the outside down. Once printed, remove the supports and clean up. Test fit the touch pads, NeoPixel ring, and speaker, tweaking as needed to have them fit well.

The USB breakout cutout requires some special attention. It is so narrow that supports were eschewed in favor of some post-print cleanup. Carefully trim material out of the side wings until the breakout slides in... just; it should be held in place with friction.
What If I Don't Have A 3D Printer?

Not to worry! You can use a 3D printing service such as 3DHubs or MakeXYZ to have a local 3D printer operator 3D print and ship you parts to you. This is a great way to get your parts 3D printed by local makers. You could also try checking out your local Library or search for a Maker Space.

You can download the STL files using these links.

- Bowl.stl
- Face.stl
Here are closeups of the snap-fit joints on the face and bowl.

Making circular snap-fit joints in Fusion 360 is covered by Noe Ruiz in this video tutorial.
Now you can mount things to the face. Start with the touch pads. Drill a small hole in each depression at the speaker end, since that's closest to the touch pads on the HalloWing. One will be near the USB breakout so be sure you stay clear of it.

Slip a piece of silicon covered wire through each hole. It's far easier if you strip and tin the ends first. Solder these to the back side of the touch pads you made earlier. Secure them into the depressions with a little hot glue, pulling the wires through as you do so.
Next is the NeoPixel ring. Hold the ring in place and mark hole locations for power, ground, and data-in. This can be done with a drill bit that will fit through the ring PCB holes, or a very fine tip marker. Remove the ring and drill the holes. Make them large enough to accept a wire from the JST connector you've prepared for the NeoPixel ring.

Thread the wires through the appropriate holes and slip the tinned ends through the ring PCB from the back. Carefully solder in place.

Put some hot glue in the ring depression and carefully seat the ring in place, while taking up the slack in the wires.
Press the USB breakout into the cutout in the side of the face. It should barely slide in, and be held in place simply by the pressure and friction from the plastic. Slide it in until the connector protrudes just slightly.

If it's loose, a couple dabs of hot glue will hold it in place.
Next up is the HalloWing. Mount it using four of the short M3 bolts.

The PIR is then mounted with the small M2.5 screws.

Next, peel the tape off the back of the speaker to reveal the adhesive. Press the speaker into place, with the leads on the same side as the HalloWing's speaker connection.

Finally connect everything to the HalloWing. The touch pads require a dab of solder to connect to the outside touch pads. Everything else plugs into the matching connections.
Connect a battery and line up the snapfit parts of the bowl and face. Taking care that all wires are tucked inside, carefully snap the pieces together.

You might find that the bowl needs some weight in the bottom. If so, some plastic covered ball bearings stuck into the bottom with hot glue should do the trick.

Code

We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here.

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial.
Start by loading the HalloWing build of CircuitPython (at least 4.0) onto the HalloWing. See this part of the above guide for instructions if you aren’t familiar. Be sure this is done before snapping the case together, as the reset signal is not accessible from outside.

**Setting up**

As usual, after the required imports, the code starts by setting up the various bits of hardware in use.

The PIR is just a digital input.

```python
pir = digitalio.DigitalInOut(board.SENSE)
pir.direction = digitalio.Direction.INPUT
```

Touch pads are also quite straightforward.

```python
touch_1 = touchio.TouchIn(board.TOUCH1)
touch_4 = touchio.TouchIn(board.TOUCH4)
```

As is audio.

```python
audio = audioio.AudioOut(board.SPEAKER)
```

The display is a bit more work.

```python
backlight = digitalio.DigitalInOut(board.TFT_BACKLIGHT)
backlight.direction = digitalio.Direction.OUTPUT
backlight.value = False
splash = displayio.Group()
board.DISPLAY.show(splash)
```

Neopixels are configured as usual. Note the use of `EXTERNAL_NEOPIXEL` so that the pixels attached to the connector will be used, and not the one on the HalloWing board. Once configured they’re all turned off.

```python
pixels = neopixel.NeoPixel(board.EXTERNAL_NEOPIXEL, 24, brightness=.2)
pixels.fill((0, 0, 0))
pixels.show()
```
Playing Sound Files

The Hallowing has a built-in amplifier with a speaker connector and volume control. Additionally, it has 8 megabytes of SPI flash rather than the 2 that the M0 Express boards have. That provides plenty of room for sound and image files.

To play a `wav` file, the file is opened and used to create a `WaveFile` object. Then the `audio` object created above is used to play it. In this simple version of the code, it waits while the file plays. The file is then closed before returning. The downside of this is that nothing else happens while the sounds plays. Making this more asynchronous is an area to explore.

```python
def play_wave(filename):
    wave_file = open(filename, "rb")
    wave = audiocore.WaveFile(wave_file)
    audio.play(wave)
    while audio.playing:
        pass
    wave_file.close()
```

Displaying Images

Displaying an image is somewhat similar: open the file, use it to create an `OnDiskBitmap` object, which is then used to create a `Sprite` object, that is then loaded into the display system.

Note that the display's backlight is turned off before loading the image and `wait_for_frame` is used to loop until the image has finished displaying, at which time the backlight is turned on again.

To keep things simpler the backlight is driven by a digital output rather than a PWM output. The result is that we can only turn it on or off, but not fade it.

```python
def show_image(filename):
    # CircuitPython 6 & 7 compatible
    image_file = open(filename, "rb")
    odb = displayio.OnDiskBitmap(image_file)
    face = displayio.TileGrid(
        odb,
        pixel_shader=getattr(odb, 'pixel_shader', displayio.ColorConverter())
    )

    # CircuitPython 7+ compatible
    # odb = displayio.OnDiskBitmap(filename)
    # face = displayio.TileGrid(odb, pixel_shader=odb.pixel_shader)

    backlight.value = False
    splash.append(face)
```
board.DISPLAY.refresh(target_frames_per_second=60)
backlight.value = True

NeoPixel Effects

Random colors are used a few times so there is a function to generate one.

```python
def random_colour():
    return (randrange(255), randrange(255), randrange(255))
```

In the ATTRACT state (more on that later) the pixels are randomly twinkled. Rather than clearing the entire ring each time, lit pixels are tracked and individually turned off. While this code updates all 6 twinkling pixels at a time, by keeping track of individual pixels it could change them one at a time. The code is hardcoded to work with 6 pixels. It could easily be made more flexible by using the size of the index list instead.

```python
twinkle_indicies = [0, 0, 0, 0, 0, 0]
def twinkle():
    for p in range(6):
        pixels[twinkle_indicies[p]] = (0, 0, 0)
        twinkle_indicies[p] = randrange(len(pixels))
        pixels[twinkle_indicies[p]] = randomColour()
    pixels.show()
```

Finally, there is a function to fill the ring with a random color.

```python
def solid():
    pixels.fill(random_colour())
    pixels.show()
```

The State Machine

The operation of the toy is governed by a finite state machine.

Wikipedia defines a finite state machine as:

> It is an **abstract machine** that can be in exactly one of a finite number of **states** at any given time. The FSM can change from one state to another in response to some external **inputs**; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

This machine has 6 states and its transitions are driven by 4 events.
Here's the state diagram, the notes what happens when each state is entered, and the transitions between states, annotated with the events that triggers each one. No event means it's followed unconditionally as soon as possible (sometimes that's after a sounds plays or an image is displayed).

The ARMED state is where the toy is sitting dormant, watching for activity. When motion is detected, the ATTRACT state is transitioned to. If accelerometer activity is detected, the ROCKED state is transitioned to. Note that this isn't currently implemented and is an area for extension.

In the ATTRACT state the toy attempts to get the cat's attention. Currently this is done by twinkling the pixels. A likely extension is to play a compelling sounds as well but would require asynchronous sound playing as mentioned earlier. If the touch pads are touched, the machine moves to the TOUCHED state. If it is rocked the ROCKED state is transitioned to. Finally, if the timeout expires, the machine goes back to the ARMED state.

In the TOUCHED state, the pixels turn red and a random sound file is played. When the sound finishes, it transitions to the PLAY state.
The ROCKED state hasn't been implemented yet and transitions immediately to the PLAY state.

The PLAY state is the core state when the toy is active. While in it, something interesting can be displayed and the pixels continually show random colors. Touch, motion, and accelerometer events cause a transition to the appropriate state. If a timeout occurs the ARMED state is transitioned to.

Finally, the MOVED state currently does nothing other than transition immediately back to the PLAY state. It's there primarily as a hook for later extension.

The timeout was mentioned several times above. Moving out of the ARMED state sets a timeout to one minute. If no activity occurs in that time, the ARMED state is returned to (from the ATTRACT or PLAY states). Any activity causes the timeout to be reset to a minute. The result is that as long as there is activity the toy stays active. When the cat becomes bored and wanders off, the timeout will expire and the toy will go back to sleep.

States are implemented by two functions: enter_state and handle_event. The enter_state function is called with the state being entered and is responsible for doing whatever is appropriate for each state. The handle_event function implements the transitions. Depending on the event and the current state, the appropriate new state is entered.

def enter_state(state):
    global current_state, timeout_time, update_time
    current_state = state

    if state == ARMED_STATE:
        timeout_time = 0
        update_time = 0
        backlight.value = False
        pixels.fill((0,0,0))
        pixels.show()

    elif state == ATTRACT_STATE:
        splash.pop()
        show_image(images[1])                 # here kitty
        reset_timeout()
        set_update(0.1, twinkle)

    elif state == PLAY_STATE:
        splash.pop()
        show_image(images[2])         # good kitty
        set_update(2.0, solid)

    elif state == TOUCHED_STATE:
        reset_timeout()
        update_time = 0
        pixels.fill((128, 0, 0))
        pixels.show()
        play_wave(sounds[randrange(len(sounds))])
        enter_state(PLAY_STATE)
elif state == MOVED_STATE:
    enter_state(PLAY_STATE)

elif state == ROCKED_STATE:
    reset_timeout()
    enter_state(PLAY_STATE)

def handle_event(event):
    if event == TIMEOUT_EVENT:
        enter_state(ARMED_STATE)

    elif event == MOVE_EVENT:
        if current_state == ARMED_STATE:
            enter_state(ATTRACT_STATE)
        elif current_state == PLAY_STATE:
            enter_state(MOVED_STATE)

    elif event == TOUCH_EVENT:
        if current_state in [ARMED_STATE, ATTRACT_STATE, PLAY_STATE]:
            enter_state(TOUCHED_STATE)

    elif event == ROCK_EVENT:
        if current_state in [ARMED_STATE, ATTRACT_STATE, PLAY_STATE]:
            enter_state(ROCKED_STATE)

They should be fairly understandable. There is one thing of note: entering the
ATTRACT and PLAY states call the function `set_update` with an interval value and
one of the NeoPixel functions described earlier. In the main loop, the function is called
at the interval specified. This is what allows the dynamic behavior within a state.

The main loop is fairly simple: if it's time to call the update function it does so, then
the four events are generated (by calling `handle_event`) as appropriate depending
on sensor activity. It uses some sensor processing functions to detect the leading
edge of sensor activity (i.e. there was no motion last time it checked and now there
is).

```python
was_moving = False

def started_moving():
    global was_moving
    started = False
    moving_now = pir.value
    if moving_now:
        started = not was_moving
        was_moving = moving_now
    return started

was_touching = False

def started_touching():
    global was_touching
    started = False
    touching_now = touch_1.value or touch_4.value
    if touching_now:
        started = not was_touching
        was_touching = touching_now
    return started
```
def started_rocking():
    return False

#-------------------------------------------------------------------------------
# Image and sound filenames
images = ['please_standby.bmp', 'here_kitty.bmp', 'good_kitty.bmp']
sounds = ['Cat_Meow_2.wav', 'Cat_Meowing.wav', 'kitten3.wav', 'kitten4.wav']

#-------------------------------------------------------------------------------
# Get started and loop, looking for and handling events
show_image(images[0])  # waiting display
time.sleep(3)
arm_time = 0
armed = True
enter_state(ARMED_STATE)

while True:
    now = time.monotonic()
    if update_time > 0 and now > update_time:
        update_time += update_interval
        update_function()
    if timeout_time > 0 and now > timeout_time:
        handle_event(TIMEOUT_EVENT)
    elif started_moving():
        handle_event(MOVE_EVENT)
    elif started_touching():
        handle_event(TOUCH_EVENT)
    elif started_rocking():
        handle_event(ROCK_EVENT)

And that's it. It's simple but could be the jumping off point to something more complex if memory was larger, possibly using a Feather M4 Express and the PropMakerWing.

The images and sounds are are available, zipped, at the link below as well as in the GitHub repository.

Here's the full code.py file.
```python
import time
from random import randrange
import board
import displayio
import digitalio
import touchio
import audioio
import audiocore
import neopixel
# import adafruit_lis3dh

# Setup hardware
pir = digitalio.DigitalInOut(board.SENSE)
pir.direction = digitalio.Direction.INPUT
touch_1 = touchio.TouchIn(board.TOUCH1)
touch_4 = touchio.TouchIn(board.TOUCH4)
audio = audioio.AudioOut(board.SPEAKER)
backlight = digitalio.DigitalInOut(board.TFT_BACKLIGHT)
backlight.direction = digitalio.Direction.OUTPUT
backlight.value = False
splash = displayio.Group()
board.DISPLAY.show(splash)

# setup neopixel ring
pixels = neopixel.NeoPixel(board.EXTERNAL_NEOPIXEL, 24, brightness=.2)
pixels.fill((0, 0, 0))
pixels.show()

# setup accelerometer
# i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
# lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)

# Play a wav file

def play_wave(filename):
    wave_file = open(filename, "rb")
    wave = audiocore.WaveFile(wave_file)
audio.play(wave)
    while audio.playing:
        pass
    wave_file.close()

# Display an image on the HaloWing TFT screen

def show_image(filename):
    # CircuitPython 6 & 7 compatible
```

© Adafruit Industries
image_file = open(filename, "rb")
odb = displayio.OnDiskBitmap(image_file)
face = displayio.TileGrid(
    odb,
    pixel_shader=getattr(odb, 'pixel_shader', displayio.ColorConverter())
)
# # CircuitPython 7+ compatible
# odb = displayio.OnDiskBitmap(filename)
# face = displayio.TileGrid(odb, pixel_shader=odb.pixel_shader)
backlight.value = False
splash.append(face)
board.DISPLAY.refresh(target_frames_per_second=60)
backlight.value = True

# Neopixel routines

def random_colour():
    return (randrange(255), randrange(255), randrange(255))

# Set 6 random pixels to random colours.
# Keep track of which are lit so they can be turned off next time
twinkle_indices = [0, 0, 0, 0, 0, 0]

def twinkle():
    for p in range(6):
        pixels[twinkle_indices[p]] = (0, 0, 0)
        twinkle_indices[p] = randrange(len(pixels))
        pixels[twinkle_indices[p]] = random_colour()
    pixels.show()

# Fill the ring with a random colour

def solid():
    pixels.fill(random_colour())
    pixels.show()

# The state machine

ARMED_STATE = 0
ATTRACT_STATE = 1
PLAY_STATE = 2
TOUCHEd_STATE = 3
MOVED_STATE = 4
ROCKED_STATE = 5
TIMEOUT_EVENT = 0
MOVE_EVENT = 1
TOUCH_EVENT = 2
ROCK_EVENT = 3

TIMEOUT = 60

timeout_time = 0
current_state = -1
update_function = None
update_time = 0
update_interval = 0

def reset_timeout():
    global timeout_time
timeout_time = time.monotonic() + TIMEOUT

def set_update(interval, func):
    global update_interval, update_time, update_function
    update_function = func
    update_interval = interval
    update_time = time.monotonic() + interval

def enter_state(state):
    global current_state, timeout_time, update_time
    #    print("Entering state {0}".format(state))
    current_state = state
    if state == ARMED_STATE:
        timeout_time = 0
        update_time = 0
        backlight.value = False
        pixels.fill((0, 0, 0))
        pixels.show()
    elif state == ATTRACT_STATE:
        splash.pop()
        show_image(images[1])  # here kitty
        reset_timeout()
        set_update(0.1, twinkle)
    elif state == PLAY_STATE:
        splash.pop()
        show_image(images[2])  # good kitty
        set_update(2.0, solid)
    elif state == TOUCHED_STATE:
        reset_timeout()
        update_time = 0
        pixels.fill((128, 0, 0))
        pixels.show()
        play_wave(sounds[randrange(len(sounds))])
        enter_state(PLAY_STATE)
    elif state == MOVED_STATE:
        enter_state(PLAY_STATE)
    elif state == ROCKED_STATE:
        reset_timeout()
        enter_state(PLAY_STATE)

def handle_event(event):
    #    print("Handling event {0}".format(event))
    if event == TIMEOUT_EVENT:
        enter_state(ARMED_STATE)
    elif event == MOVE_EVENT:
        if current_state == ARMED_STATE:
            enter_state(ATTRACT_STATE)
        elif current_state == PLAY_STATE:
            enter_state(MOVED_STATE)
    elif event == TOUCH_EVENT:
        if current_state in [ARMED_STATE, ATTRACT_STATE, PLAY_STATE]:
            enter_state(TOUCHED_STATE)
    elif event == ROCK_EVENT:
        if current_state in [ARMED_STATE, ATTRACT_STATE, PLAY_STATE]:
            enter_state(ROCKED_STATE)
# Check for event triggers

was_moving = False

def started_moving():
    global was_moving
    started = False
    moving_now = pir.value
    if moving_now:
        started = not was_moving
    was_moving = moving_now
    return started

was_touching = False

def started_touching():
    global was_touching
    started = False
    touching_now = touch_1.value or touch_4.value
    if touching_now:
        started = not was_touching
    was_touching = touching_now
    return started

def started_rocking():
    return False

# Image and sound filenames

images = ["please_standby.bmp", "here_kitty.bmp", "good_kitty.bmp"]
sounds = ["Cat_Meow_2.wav", "Cat_Meowing.wav", "kitten3.wav", "kitten4.wav"]

# Get started and loop, looking for and handling events

show_image(images[0])  # waiting display
time.sleep(3)
arm_time = 0
armed = True

enter_state(ARMED_STATE)

while True:
    now = time.monotonic()

    if update_time > 0 and now > update_time:
        update_time += update_interval
        update_function()

    if timeout_time > 0 and now > timeout_time:
        handle_event(TIMEOUT_EVENT)

    elif started_moving():
        handle_event(MOVE_EVENT)

    elif started_touching():
        handle_event(TOUCH_EVENT)

    elif started_rocking():
        handle_event(ROCK_EVENT)