ESP-NOW

UIFLow's implementation of Espressif's ESP-NOW communications technology.

SAMPLE

ESP-NOW

Introduction

ESP-NOW is a protocol developed by Espressif, which enables multiple devices to communicate with one another without using Wi-Fi. The protocol is similar to the low-power 2.4GHz wireless connectivity that is often deployed in wireless mice. So, the pairing between devices is required prior to their communication. After the pairing is done, the connection is secure and peer-to-peer, with no handshake being required.

Conventions uses in this document.

ESPNOW, ESP-NOW, ESP-Now and variants mean the same thing. Some documents online use one format while other documents use other formats. I have used **ESP-NOW** in text throughout this document and espnow in code, but if you spot it done differently, please message me so that I can Correct this mistake.

```
Text like this is used to present code and API's.
```

Disclaimer

While every attempt has been made to make sure the information provided is accurate and up to date, due to the ever changing code and devices, API's and example code may become broken.

If you find something in this publication that is incorrect or not working, you can leave messages for me on the **M5Stack FaceBook Group, M5Stack official forum, the Discord channel** or you can email me: **ajb2k3@gmail.com**

Setup

Hardware and Software Requirements

- M5Stack Core, Core2, Stick, Atom. (StickV and StickT are not usable).
- M5Burner,
- UIFlow,
- Thonny.

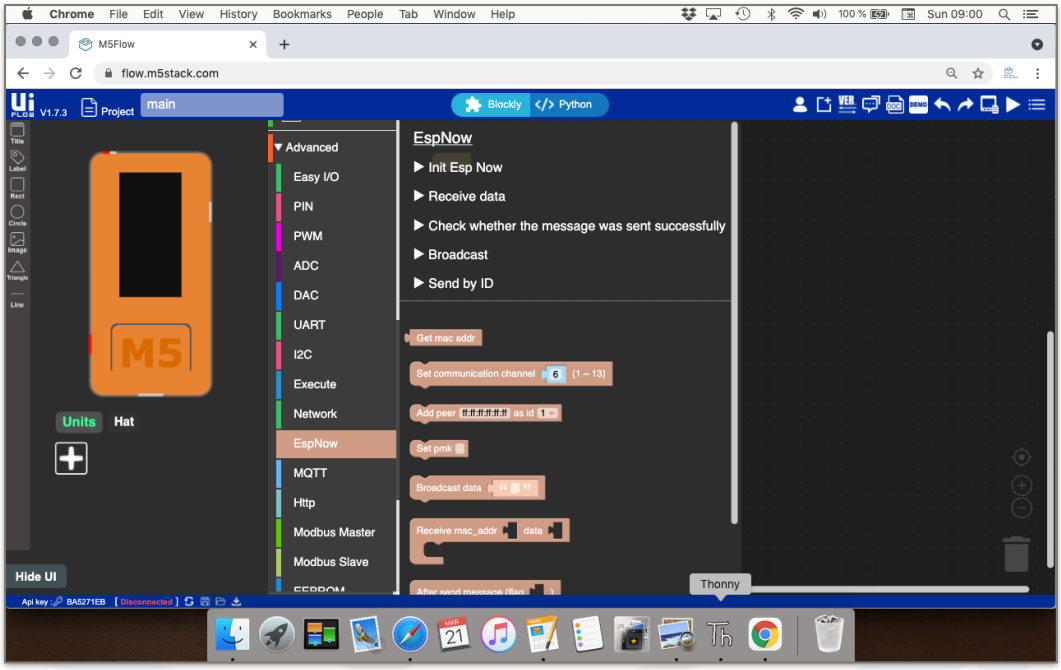
Software Setup

I have already shown you how to update firmware and setup the software in previous chapters and so I will assume that you have already read and understood that chapter.

In this chapter I will only be covering the UIFlow blocks and Micropython ESP-NOW functions that we have available to us.

Finding the Functions.

To find the ESP-NOW functions in UIFlow we first scroll down to the Advanced menu and after clicking on Advanced, scroll down and click on ESP-NOW to bring up the ESP-NOW menu.

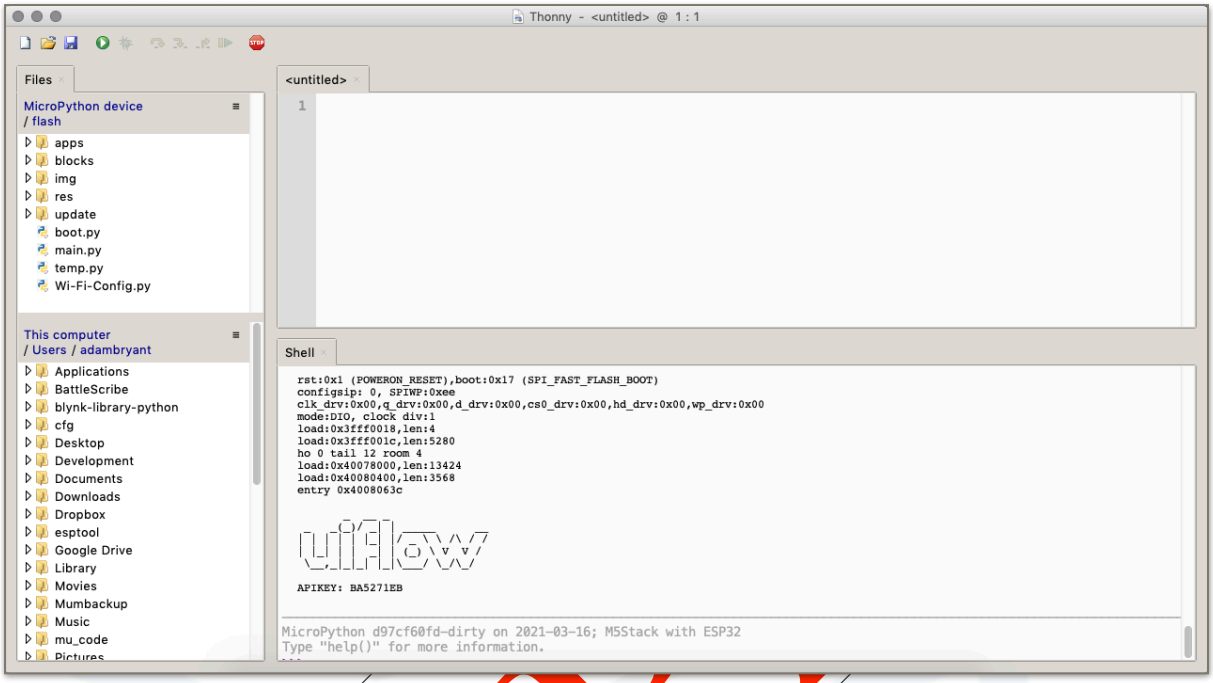


Screenshot showing the Advanced>ESP-NOW menu.

To find the Micropython commands we can add them in UIFlow and then switch to the </> Python screen or we can look them up as they are stored on the M5Stack devices.

Using Thonny to view the commands.

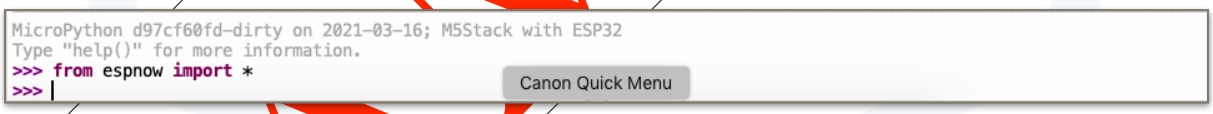
Start Thonny and then connect an M5Stack Controller (in these images I am using a Core2). Press the Power/reset button to restart the controller and then press the red "Stop" button at the top of the screen and you should end up with a shell output that looks like in the following image.



Thonny showing the M5Stack controller connected in the shell.

To access the Micropython commands we first need to import the ESP-NOW commands. This is done by typing the following command into the shell.

```
from ESP-NOW import *
```

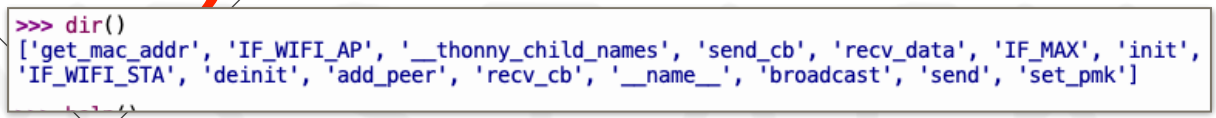


Screen shot of Thonny's shell after running the from ESP-NOW import * command.

Now that we have imported the commands we now need to list them.

This is done by running:

```
dir()
```



Screen shot showing the list of Micropython commands as reported by Thonny's shell.

Micropython API's and their matching UIFlow Blocks

As we can see from the screen shot in the previous section, the functions we have available to us are as follows:

- get_mac_addr,
- IF_WIFI_AP,
- send_cb,
- recv_data,
- IF_MAX,
- init,
- IF_WIFI_STA,
- deinit,
- add_peer,
- recv_cb,
- broadcast,
- send,
- set_pmk

In the following section I am describing the functions in the order that they are used.

In Micropython, all functions are preceded by esp. (in lower case).

Import ESP-NOW

Before we can start using ESP-NOW we first need to import the functions. We import the function by issuing the following Micropython command.

```
import espnow
```

However, some of the functions we need require the import of additional functions only found in the WIFI module.

```
import wifiCfg
```

Once these modules are imported we now have the ability to access the functions contained within.

Initialise ESP-NOW.

In order to start using ESP-NOW we first need to initialise ESP-NOW. In Micropython this is done by issuing the following command.

```
espnow.init()
```

However, in uiflow this command is automatically added in to the Setup phase of the program when we add an ESP-NOW block.

Deinitialise ESP-NOW.

If for some reason ESP-NOW needs to be terminated in order to run another function that it may interfere with, we can use the following following Micropython code.

```
espnow.deinit()
```

Unfortunately I haven't found a way in UIFLow for this to be added and so need to be issued using an execute block.

Set AP / STA Active.

By default AP mode or Access point mode is set to True in micropython but what does this mean?

Wifi devices can act in two mode, AP (Access Point) or STA (STAtion Mode)

In AP mode, devices act as WIFI hubs receiving and transmitting data to multiple devices connected to the network whereas, devices in STA mode can only send and receive data between itself and the hub. To control these modes we use the two following Micropython functions.

```
wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
```

These functions are actually part of the WIFICFG library but are required by the ESP-NOW functions.

For single device to device communications a transmitting device should be set as:

```
wifiCfg.wlan_ap.active(False)
wifiCfg.wlan_sta.active(True)
```

While the receiving "Hub" device should be set to:

```
wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(False)
```

If we want the receiving "Hub" to send the data over the network to other devices or to online services, both lines must be set as:

```
wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
```

If both AP and STA are set to false then no wifi or ESP-Now communication can happen. For a "Mesh" based network where all devices send and receive data to and from other connected devices, all devices must be set as:

```
wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
```

IF MAX

The IF MAX function is used to control how many stations can connect to an access point. If we want to restrict the amount of stations we use the following functions.

```
wlan_ap.config(max=3)
```

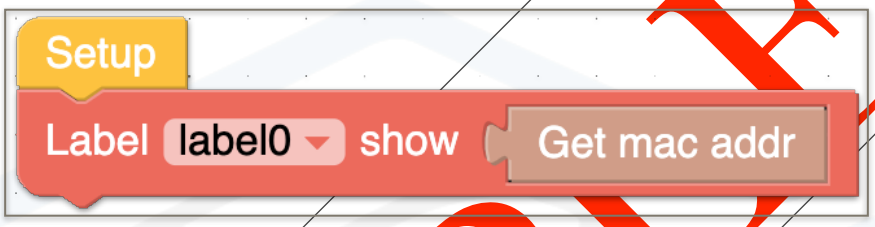
Get MAC Address

Once we have the ESP-NOW process started, we need to now pair two or more devices together. In order to pair devices, we first need to find their MAC addresses. To find out the MAC address we use the following Micropython code

```
espnow.get_mac_addr()
```

Or UIFlow block:

The Get Mac Address function is used to return the 6 byte MAC (Media Access Control address) of the M5Stack controller that the function is run on. On its own its not very useful and needs another function to run. In the following example I am using a Label block in order to show the MAC address on the Core2's screen.



Get MAC address example made in UIFlow

In UIFlow the program looks simple but in Micropython code for this example is as follows.

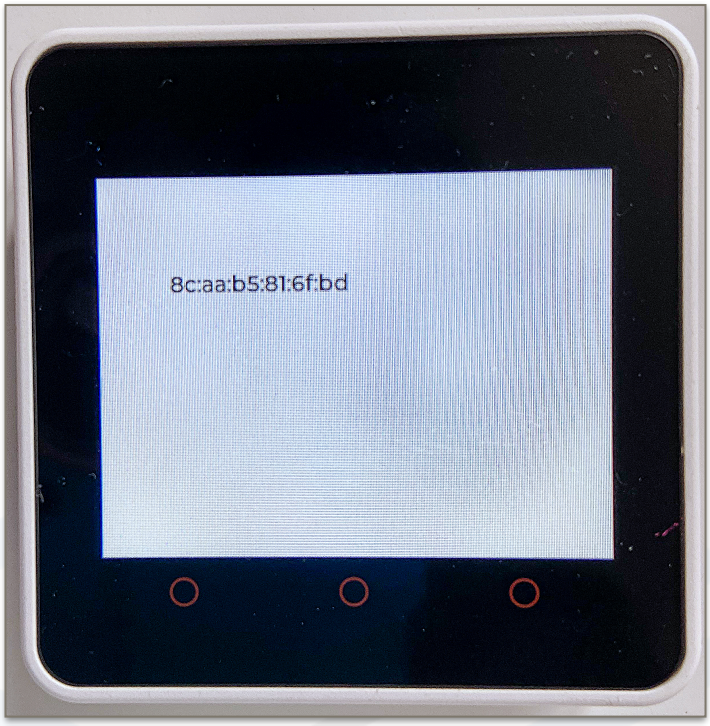
```
from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

setScreenColor(0x111111)

wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
ESP-NOW.init()
label0 = M5TextBox(3, 6, "Text", lcd.FONT_Default, 0xFFFFFF, rotate=0)

label0.setText(str(espnow.get_mac_addr()))
```

The import line to note here is

```
label0.setText(str(espnow.get_mac_addr()))
```

This uses the MAC address returned by the function as the label text.

When run, you will see this displayed on the screen of M5Stack and M5Stick devices with screens.



Screenshot of M5Stack Core2 showing its MAC address.

Run this on all Controllers that have a screen and make a note somewhere safe as you will be constantly looking up these numbers for network based projects.

On the M5Stack Atom Lite and Atom Matrix we can't run this code as they have no screen. In order to get their MAC address we have to use Thonny and print out the address to the shell.

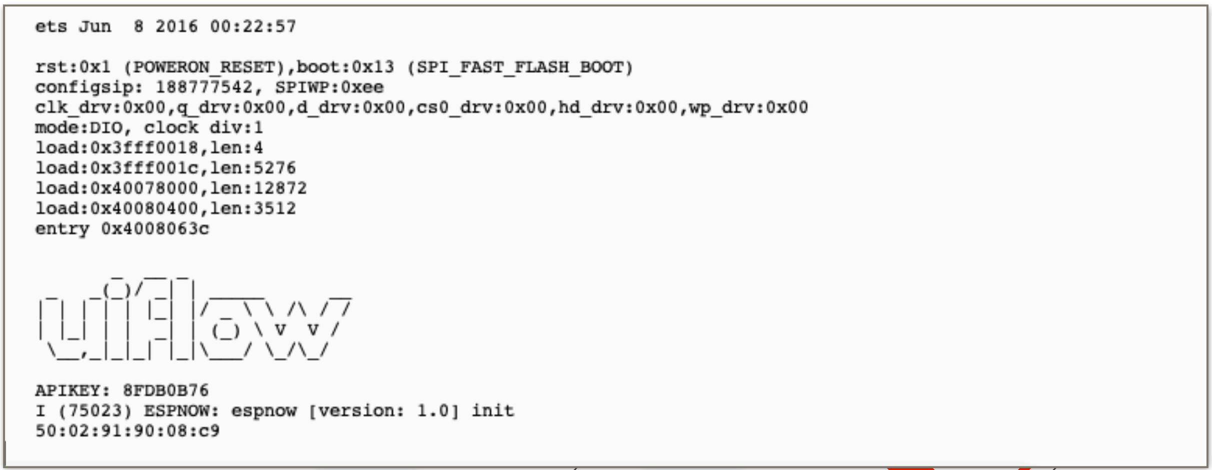
In order for the code to work work on the Atom Lite and Atom Matrix and show us the MAC address over the shell in Thonny, we need to change:

```
label0.setText(str(espnow.get_mac_addr()))
```

To

```
print(str(espnow.get_mac_addr()))
```

Once the line has been changed, run the code in UIFlow and look at the shell in Thonny. At the bottom you will see the MAC address displayed.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 188777542, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:5276
load:0x40078000,len:12872
load:0x40080400,len:3512
entry 0x4008063c

APIKEY: 8FDB0B76
I (75023) ESPNOW: espnow [version: 1.0] init
50:02:91:90:08:c9
```

Screen shot showing how the MAC address appears in Thonny after the amended code is run in UIFlow.

Add Peer

Now that we have addresses of each device we can connect them together. We do this using the Add Peer function in Micropython:

```
espnow.add_peer('ff:ff:ff:ff:ff:ff', id=1)
```

.Or the Add Peer block in UIFlow:

The Add Peer block is used on the transmitter and is used to hold the MAC address of the controller we wish to send data to. By clicking on the dropdown box next to the ID number, we can add up to ten different receivers with different ID's.

The addresses are written in Hexadecimal and are case sensitive (0,o, and O are all different values). When adding the transmitting device MAC address you must make to copy it exactly as it is shown on the screen on shell output.

This example shows the receiving controllers MAC address and adds the transmitting device's MAC address as device 1.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5stack_ui import *
from uiflow import *
import ESP-NOW
import wifiCfg

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)

wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
ESP-NOW.init()

label0.set_text(str(ESP-NOW.get_mac_addr()))
ESP-NOW.add_peer('50:02:91:90:08:c9', id=1)
```

Set Communication Channel

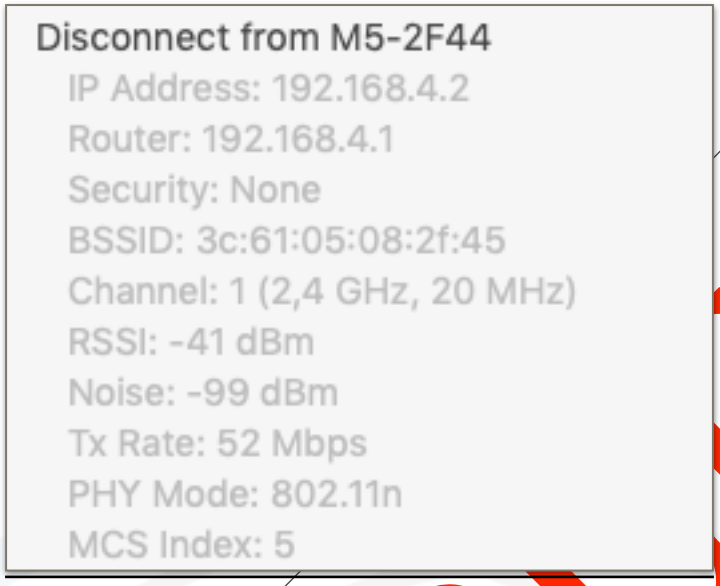
The set communication channel function is used to control which WIFI channel devices communicate on while in AP (Access Point) mode. If this function is not set devices will attempt to communicate on the default WIFI channel but this may interrupt normal local WIFI communication that is also using the default channel. The Micropython function for this is as follows:

```
wifiCfg.wlan_ap.config(channel=6)
```

With the matching UIFlow block as follows:

This function only works on WIFI which uses the 2.4GHz band and does not work on the 5GHz band that most modern WIFI networks now use.

To find out which channel an M5Stack is board casting on in STA mode you need to view the WIF setting in the host operating system's WIFI manager.



This screen shot (taken in OSX) shows that the device has presented itself to the WIFI network on channel 1 of the 2.4GHz network and so the AP mode channel must be set to channel 2 through to channel 13.

Set PMK

The Set PMK function is designed to allow a private key to be issued in order to secure the communications between two devices. In Micropython this is done by issuing the following command.

```
espnow.set_pmk('')
```

In Uiflow we have the Block shown above to set the PMK. If no key has been set then ESP-NOW will use the default base key other wise a 16 byte key must be set using hexadecimal

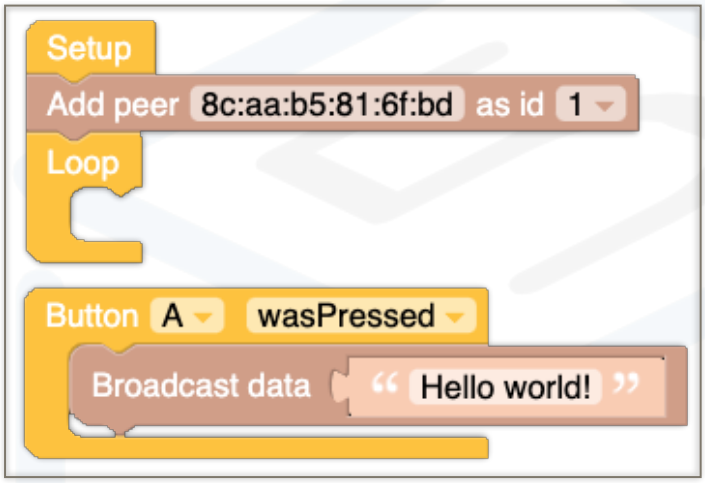
Broadcast Data

The Broadcast Data block is used to send information from the transmitter to any receiver connected, this can be a string (as in "Hello World!") Variables or numbers. The Micropython function is:

```
espnow.broadcast(data=str('Hello world!'))
```

with the UIFlow block:

When run on an Atom Matrix, this examples sends "Hello World" to the listed peer every time the button hidden under the LEDs is pressed.



The Micropython code for this transmission program is as follows:

```
from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

rgb.set_screen([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])

wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
espnow.init()

def buttonA_wasPressed():
  # global params
  espnow.broadcast(data=str('Hello world!'))
  pass
btnA.wasPressed(buttonA_wasPressed)

espnow.add_peer('8c:aa:b5:81:6f:bd', id=1)
```

```
while True:
  wait_ms(2)
```

Send Message ID

The Send Message ID Data function is used to send data to specific connected devices whereas Broadcast data function send the data to all connected devices.

The Micropython function is as follows:

```
espnow.send(id=1, data=str())
```

And the UIFLow block presents itself as:

WIFI AP Mode.

The WIFI AP mode function is used to set the device into AP (Access Point) or STA (Station Point) mode.

The Micropython code for AP Mode is:

```
wifiCfg.wlan_ap.active(True)
```

and for station mode it is:

```
wifiCfg.wlan_sta.active(True)
```

Setting them to True makes them active and False in active. If you wish to use both, you have to make sure that AP channel is set to a different channel then the STA channel.(see set channel function)

Callback Functions.

Callback functions are triggered upon sending or receiving data. The following callback functions are currently available in Micropython and UIFlow.

Receive MAC Address and Data

The Receive MAC Address and data block is a loop that returns the MAC address and data sent from a transmitting controller and runs any code placed inside it.

Using this loop block we can trigger events and action only when triggered by a message completely separate from the main program loop. The Micropython Code for this function is:

```
def recv_cb(_):
  # global params
  _, _, _ = ESP-NOW.recv_data(encoder='str')

  pass
espnow.recv_cb(recv_cb)
```

This consists of the parts, first is where the functions we wish to be triggered are defined

```
def recv_cb(_):
  # global params
  _, _, _ = ESP-NOW.recv_data(encoder='str')

  pass
```

and second is the function.

```
espnow.recv_cb(recv_cb)
```

In UIFlow this is presented as the following loop block.

After Send Message Flag.

The After Send Message Flag function is used to trigger an action once data has been sent over ESP-NOW. To trigger this code in Micropython we use the following function:

```
def send_cb(flag):
  # global params
  _ = flag

  pass
espnow.send_cb(send_cb)
```

The function consists of two parts, the first is where we define the code we wish to trigger

```
def send_cb(flag):
  # global params
  _ = flag

  pass
```

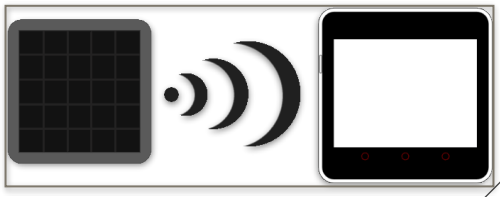
And the second is where we want to function to be triggered.

```
espnow.send_cb(send_cb)
```

In UIFlow this function presents itself as a separate Loop block.

ESP-NOW Projects

Project 01 - One Way Communication.



Introduction

So far I have shown you the ESP-NOW functions needed for simple communication between two M5Stack controllers. In this guide I will show you an example of one way communication between two controllers using EPS-NOW by making a simple wireless doorbell.

Parts

For this project you will need the following parts:

- Atom Matrix or Atom Lite,
- Atom Tailbat
- Core2, Core, Fire, or Core2 AWS,
- 3d Printed case for the Atom Matrix/Lite+Tailbat.

Code

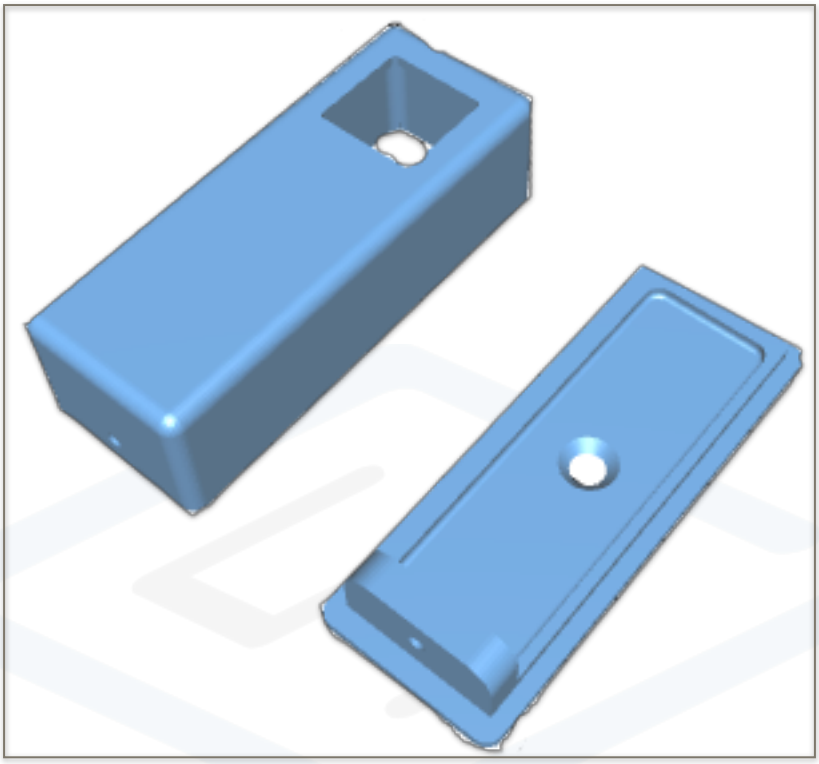
The code used in this basic example is separated into two parts, the transmitter code and the receiver code. The transmitter code is downloaded to the Atom Matrix or Atom Lite while the receiver code is downloaded to the M5Stack Core or Core2.

3D Printed Files

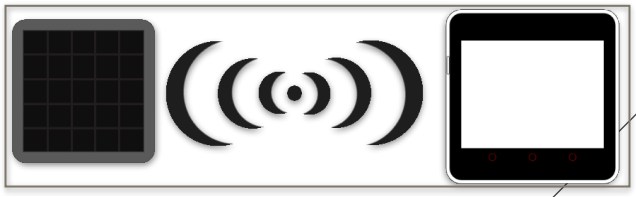
The 3D printed files can be downloaded from this GITHUB link. The files are only provided as a basic design and can be modified but I ask that you credit me and post links to the original files.

3D Printable Case

3D render of the prototype doorbell case.



Project 02 - Two Way Communication.



Introduction

In the previous project I showed you a simple example of one way ESP-NOW communication in order to make a simple wireless doorbell. Is this project I will show you how to expand the doorbell in order for the receiver to request to check that the doorbell is still active. At present here is no battery monitor on the Atom Matrix or Tailbat and so all we can do is send a hello to the the Atom Matrix and see if it responds. We wont need any additional 3d printed parts as we have already made them in Project 1.

Parts

For this project you will need the following parts:

- Atom Matrix or Atom Lite,
- Atom Tailbat
- Core2, Core, Fire, or Core2 AWS.

Code

Further Reading.