Three actuator linear slider

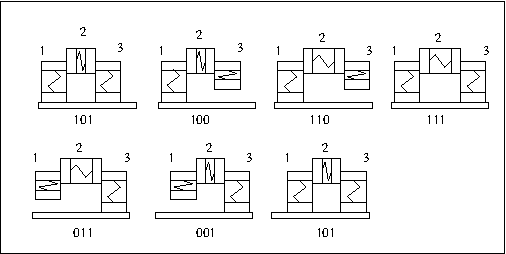
# Objective

To facilitate motion along linear path with three linear actuators which can be electromagnetic or piezo type.

# Physical principle

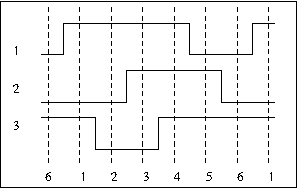
If two of the actuators provide, depending on their state, friction or free sliding against the surface, while the third one is capable of changing the distance between the first two, it is possible, by alternating contractions and extensions of thtee actuators, to achieve linear movements.

If extended actuator is coded as `1’ and contracted as `0’, the sequence for six phases is shown below. The seventh phase is the same as the first one. Free sliding is depicted as raised element, cohesion as lowered.



Note that only one of three binary positions changes from one phase to the next making this sequence similar to binary Gray code.

Three control signals, provided high level extends the actuator and low level contracts it, would look like below



# Implementation

The implementation consists of

* Signal Generator
* High Voltage (HV) side
* Mechanical part

# Signal Generator

## Overview

Signal generator is implemented on [ESP32 WROOM 32D](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf) System on Module (SoM) from Espressif Systems. Even though it’s NRND (Not Recommended for New Designs), its low cost and upward compatibility with Espressif newer products makes it a feasible choice for prototyping.



Another advantage of ESP32 SoM is the presence of WiFi module which allows control over WiFi from smartphone or tablet.

SoM runs [MicroPython](https://micropython.org/) – a powerful and versatile development tool ideal for fast and efficient prototype building.

## MicroPython

### Installation

To install MicroPython on SoM you will need a Windows, Linux or MacOS host with the following software installed:

* python version 3.7 or later
* esptool
* ampy

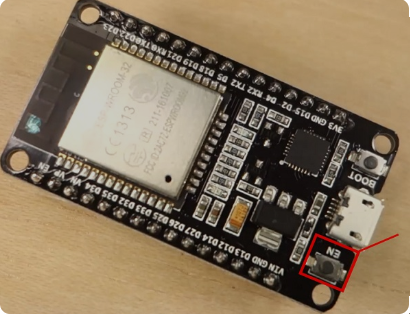
For esptool and ampy utilities installation instructions please refer to ["Esptool Installation"](https://docs.espressif.com/projects/esptool/en/latest/esp32/installation.html) and ["Adafruit Ampy Installation"](https://learn.adafruit.com/micropython-basics-load-files-and-run-code/install-ampy).

After these are in place, refer to [MicroPython Installation page](https://micropython.org/download/ESP32_GENERIC/) for instructions

NB:

1. If the Virtual COM port on ESP32 USB connection (COM12 or /dev/ttyUSB0) was used by the host program, like minicom or putty, the program must exit and free the COM port before the port can be used by esptool of ampy utility
2. The firmware must be written to ESP32 flash with offset 0x1000

After the firmware has been flashed, please start a terminal emulator ( minicom or putty) on your host with baudrate 115200 8 bits no parity and press RESET button on ESP32.



You should see something like this:

MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32

Type "help()" for more information.

>>>

This is MicroPython console, also called REPL (Read-Eval-Print Loop) and it makes all MicroPython functionality available.

### REPL programming example

For example, to get the desired signal pattern on pins 12,13 and 14, you can enter on REPL console:

>>> from time import sleep

>>> from machine import Pin

>>>

>>> p1 = Pin(12, Pin.OUT)

>>> p2 = Pin(13, Pin.OUT)

>>> p3 = Pin(14, Pin.OUT)

>>>

>>> delay=.25

>>>

>>> while (True):

... p1.on()

... sleep(delay)

... p3.off()

... sleep(delay)

... p2.on()

... sleep(delay)

... p3.on()

... sleep(delay)

... p1.off()

... sleep(delay)

... p2.off()

... sleep(delay)

...

...

...

REPL console’s primary purpose is bringup and debugging. Normal operation and development should rely on other MicroPython features like filesystem and network connections

### WiFi connectivity and WebREPL

One of the most handy ESP32 features is embedded WiFi controller.

WebREPL is a REPL console available via http connection. To get WiFi network and WebREPL up and running the several files have to be copied to ESP32 filesystem. Host file copy command looks like:

ampy -p /dev/ttyUSB0 put <local\_file> <esp32\_file>

The files are:

SSID  
WiFiPWD  
boot.py  
webrepl\_cfg.py  
wifinet.py

File SSID should contain a string with WiFi Access Point SSID. Make sure there’s no trailing newline.

SSID:

MyAccessPoint

File WiFiPWD should contain security password for WiFi Access Point. No trailing newline.

WiFiPWD:

Passwd123

File boot.py is a startup script executed on every SoM boot. For WiFi connectivity and REPL it should look like this:

boot.py:

# This file is executed on every boot (including wake-boot from deepsleep)

#import esp

#esp.osdebug(None)

import webrepl

webrepl.start()

import wifinet

wifinet.start()

File webrepl\_cfg.py has a WebREPL password that’s required for WebREPL startup

webrepl\_cfg.py:

PASS = '8266'

And wifinet.py has methods for getting ESP32 online and offline

wifinet.py:

import sys

import network

from time import sleep

def start():

st = network.WLAN(network.STA\_IF)

st.active(True)

SsidFile = 'SSID'

WifiPwdFile = 'WiFiPWD'

if (not st.active()):

print('WiFi not available\n')

sys.exit()

try:

with open(SsidFile, 'r') as f:

ssid = f.read()

f.close()

except OSError:

print("Could not read file: ", SsidFile)

sys.exit()

try:

with open(WifiPwdFile, 'r') as f:

wifipwd = f.read()

f.close()

except OSError:

print("Could not read file: ", WifiPwdFile)

sys.exit()

st.connect(ssid, wifipwd)

timeout = 10

while (timeout):

if (not st.isconnected()):

timeout = timeout - 1

sleep(1)

else:

timeout = 0

if (not st.isconnected()):

print("Could not connect to ", ssid)

sys.exit()

st.ifconfig()

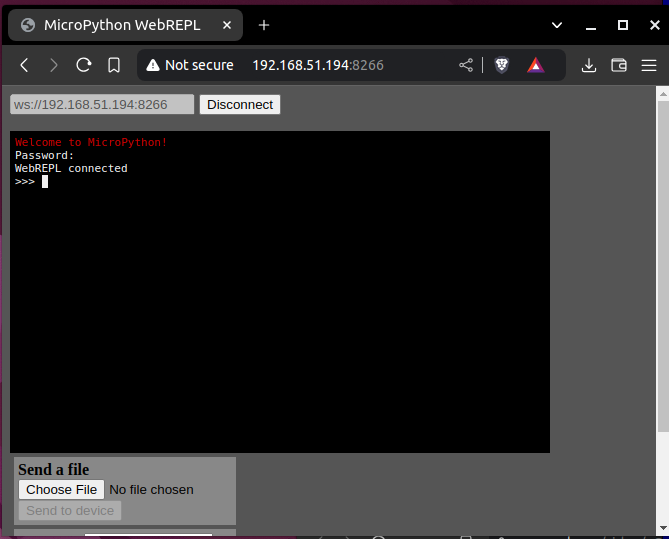
def stop():

st = network.WLAN(network.STA\_IF)

st.disconnect()

st.active(False)

In this configuration, ESP32 on startup will be automatically connected to the access point listed in SSID file and will allow WebREPL access on port 8266. Password is the same as in WiFiPWD file.



To find out ESP32 address, you can

1. Check your Access Point connections, ESP32 will be listed as `mpy-esp32’  
     
   or
2. On serial line REPL type:

>>> import network  
>>> st = network.WLAN(network.STA\_IF)  
>>> st.ifconfig()  
('192.168.51.194', '255.255.255.0', '192.168.51.1', '192.168.51.1')

The first address in st.ifconfig() output is ESP32 IP address. Note that once it’s assigned it will stay the same for subsequent connections for most access points.

Note that once WebREPL connection is established, it also allows file uploads to ESP32 (low left corner of the page) in lieu of ampy utility