```python
import serial
import time

# You will have to change this to whatever COM port the pico is assigned when
# you plug it in.
# On Windows you can open device manager and look at the 'Ports (COM & LPT)' dropdown
# the pico will show up as 'USB Serial Device'
PICO_PORT = 'COM5'

MHZ = 1000000
```

```python
# helper for sending serial commands
# expects a string
def send(command, echo = True):
    # pico is expecting a newline to end every command
    if command[-1] != '\n':
        command += '\n'

    resp = ''
    try:
        conn = serial.Serial(PICO_PORT, baudrate = 152000, timeout = 0.1)
        conn.write(command.encode())
        if echo:
            resp = conn.readlines()
            resp = "".join([s.decode() for s in resp])

    except Exception as e:
        print("Encountered Error: ", e)

    finally:
        conn.close()

    return resp
```

Test Serial Communication with the Pico

```python
assert send('reset')    == 'ok\r\n'
assert send('status')   == '0\r\n'
assert send('version')  == '0.0.0\r\n'
print('Serial Communication Successful')
```

```
Serial Communication Successful
```

Test Register Readback

```python
# Helper for reading register values and putting them in a dictionary
def readregs(ref_clk = 125 * MHZ):
    ad9959 = {}
    regs = send('readregs')
    regs = regs.split('\r\n')

    regs = [''.join(r.split()[1:]) for r in regs]

    for i, reg in enumerate(regs):
        try:
            regs[i] = int(reg, 16)
        except ValueError:
            pass

    ad9959['pll_mult'] = (regs[1] & 0x7c0000) >> 18

    sys_clk = ref_clk * ad9959['pll_mult']

    for i in range(4):
        ad9959[i] = {}

        ftw = regs[5 + 9 * i]
        ad9959[i]['freq'] = ftw / 2**32 * sys_clk

        pow = regs[6 + 9 * i]
        ad9959[i]['phase'] = pow * 360 / 2**14

        acr = regs[7 + 9 * i]
        if acr & 0x001000:
            ad9959[i]['amp'] = (acr & 0x0003ff) / 1023
        else:
            ad9959[i]['amp'] = 1

    return ad9959


readregs()
```

```
{'pll_mult': 4,
 0: {'freq': 0.0, 'phase': 0.0, 'amp': 1},
 1: {'freq': 0.0, 'phase': 0.0, 'amp': 1},
 2: {'freq': 0.0, 'phase': 0.0, 'amp': 1},
 3: {'freq': 0.0, 'phase': 0.0, 'amp': 1}}
```

```
assert send('reset') == 'ok\r\n', 'Could not run "reset" command'
ad9959 = readregs()
for i in range(4):
    assert ad9959[i]['freq'] == 0
    assert ad9959[i]['phase'] == 0
    assert ad9959[i]['amp'] == 1

send('setfreq 0 100000000')
send('setphase 1 270')
send('setamp 2 0.5')

ad9959 = readregs()

assert abs(ad9959[0]['freq'] - 100 * MHZ) < 1
assert abs(ad9959[1]['phase'] - 270) < 1
assert abs(ad9959[2]['amp'] - 0.5) < 0.01

print('Register Readback Successful')
```

```
Register Readback Successful
```

Test Single Stepping Table Mode

Program a 2000 step table that single steps from 10 MHz to 100 MHz over the corse of 2 seconds. The resulting sweep can easily be seen with a spectrum analyzer. It is then automatically executed and checks that all 2000 triggers were processed successfully.

```
startPoint = 10 * MHZ
endPoint = 100 * MHZ
totalTime = 2 # sec

spacing = 1000 * 10**(-6) # us
steps = round(totalTime / spacing)
delta = (endPoint - startPoint) / steps

send('debug off')
send('mode 0 1')
send('setchannels 1')

for i in range(steps):
    send(f'set 0 {i} {startPoint + delta * i} 1 0 {spacing * 10**9 / 8}', echo=False)
assert send(f'set 4 {i + 1}') == "ok\r\n"

print("Table Programmed, Executing")
assert send('start') == 'ok\r\n', 'Buffered Execution did not start correctly'
time.sleep(totalTime)
assert send('numtriggers') == '2000\r\n', 'Wrong number of triggers processed'
print('Table Executed successfully')
```

```
Table Programmed, Executing
Table Executed successfully
```

Test Non-Volatile Storage

Run this test after the previous one to test storing and retrieving table instructions in non-volatile memory that will survive a power cycle.

```
send('save')

# # destory current table
time.sleep(2)
for i in range(steps):
    send(f'set 0 {i} 0 0 0 0', echo=False)

# load and run table
send('load')
time.sleep(1)
send('mode 0 1')
send('setchannels 1')
send('start')

time.sleep(2)
assert send('numtriggers') == '2000\r\n', 'Something went wrong'
print('Table run from non-volatile memory successfully')
```

```
Table run from non-volatile memory successfully
```

Amplitude Sweep

Pico Start

```
send('debug off')
send("""mode 1 1
setchannels 1
setfreq 0 100000000
setfreq 1 100000000
setfreq 2 100000000
setfreq 3 100000000
set 0 0 1.0 0.0 0.001 1 2000
set 0 1 0.0 0.5 0.001 1 2000
set 0 2 0.5 1.0 0.001 1 2000
set 0 3 1.0 0.0 0.001 1 2000
set 0 4 0.0 1.0 0.001 1 2000
set 0 5 1.0 0.5 0.001 1 2000
set 0 6 0.5 0.0 0.001 1 2000
set 0 7 0.0 1.0 0.001 1 2000
set 4 8
start
""")
```

```
assert send('numtriggers') == '8\r\n'
print('Success')
```

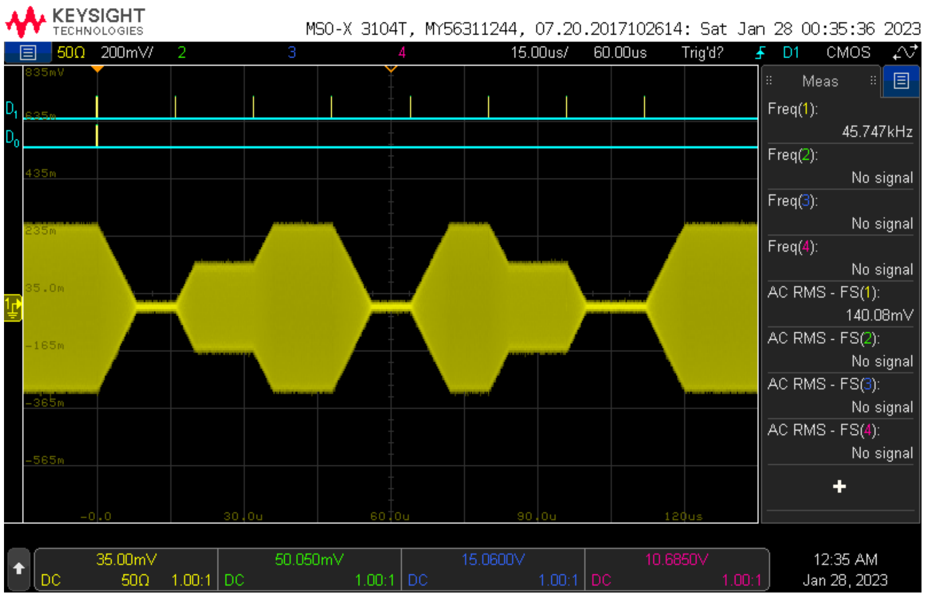
```
'ok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nc
```

HWStart

```
send('debug off')
send("""abort
mode 1 1
setchannels 1
setfreq 0 100000000
setfreq 1 100000000
setfreq 2 100000000
setfreq 3 100000000
set 0 0 1.0 0.0 0.001 1 2000
set 0 1 0.0 0.5 0.001 1 2000
set 0 2 0.5 1.0 0.001 1 2000
set 0 3 1.0 0.0 0.001 1 2000
set 0 4 0.0 1.0 0.001 1 2000
set 0 5 1.0 0.5 0.001 1 2000
set 0 6 0.5 0.0 0.001 1 2000
set 0 7 0.0 1.0 0.001 1 2000
set 4 8
hwstart
""")
```

```
'ok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nc
```

This produces the following scope trace:



D_1 is the IO_UPDATE line between the pico and the AD9959.
D_0 is the external trigger line into the pico.
The yellow trace is any of the 4 channel outputs from the AD9959

Frequency Sweep

```
f53 = 53000000
f56 = 56000000
f59 = 59000000

d = 2000
t = 3000

send(
f"""abort
mode 2 1
setchannels 1
set 0 0 {f53} {f59} {d} 1 {t}
set 0 1 {f59} {f56} {d} 1 {t}
set 0 2 {f56} {f53} {d} 1 {t}
set 0 3 {f53} {f56} {d} 1 {t}
set 0 4 {f56} {f59} {d} 1 {t}
set 0 5 {f59} {f53} {d} 1 {t}
set 4 6
start
"""
)
```

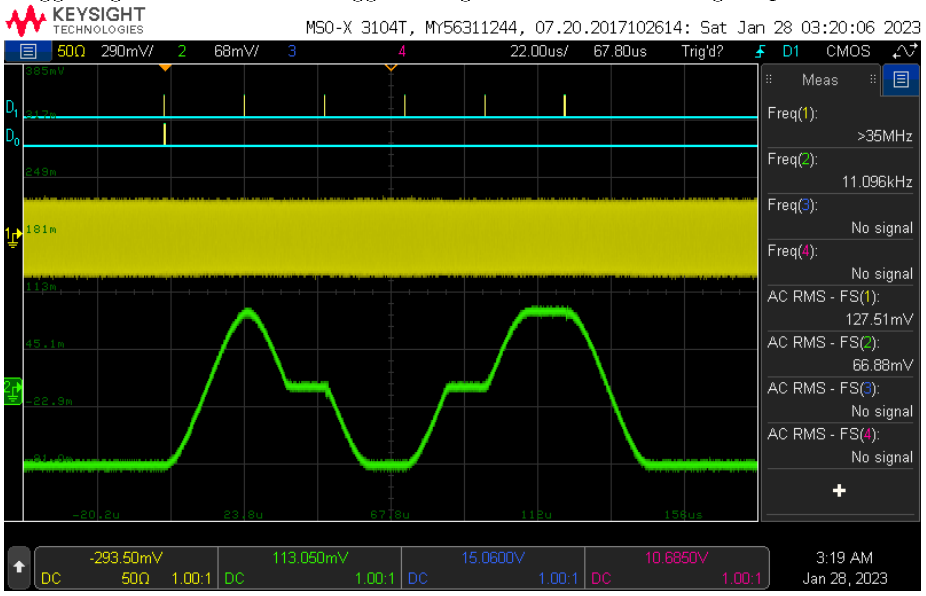
```
assert send('numtriggers') == '6\r\n'
print('Success')
```

```
Success
```

```
send(
f"""abort
mode 2 1
setchannels 1
set 0 0 {f53} {f59} {d} 1 {t}
set 0 1 {f59} {f56} {d} 1 {t}
set 0 2 {f56} {f53} {d} 1 {t}
set 0 3 {f53} {f56} {d} 1 {t}
set 0 4 {f56} {f59} {d} 1 {t}
set 0 5 {f59} {f53} {d} 1 {t}
set 4 6
hwstart
"""
)
```

```
'ok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\nok\r\n'
```

Triggering of the hardware trigger line generates the following scope trace:



D_1 is the IO_UPDATE line between the pico and the AD9959.
D_0 is the external trigger line into the pico.
The yellow trace is any of the 4 channel outputs from the AD9959
The green trace is the output from an interferometer

Phase Sweep

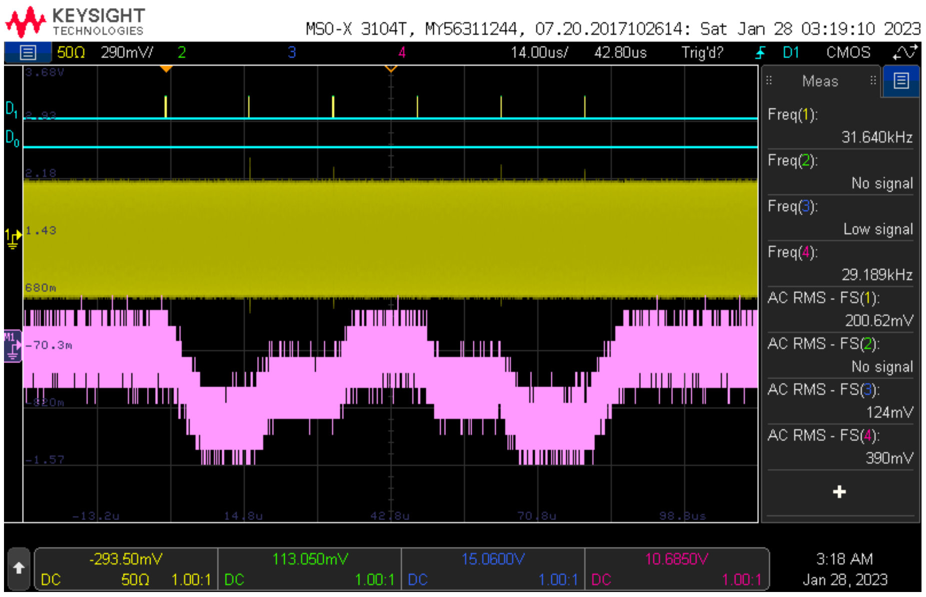
```
t = 2000
d = 0.2

send(f"""abort
debug off
setfreq 0 100000000
setfreq 1 100000000
setfreq 2 100000000
setfreq 3 100000000
setphase 0 0
setphase 1 0
setphase 2 0
setphase 3 0
mode 3 1
setchannels 2
set 0 0 0 0 0 0 {t}
set 0 1 0 0 0 0 {t}
set 0 2 0 0 0 0 {t}
set 0 3 0 0 0 0 {t}
set 0 4 0 0 0 0 {t}
set 0 5 0 0 0 0 {t}
set 1 0 0 180 {d} 1 {t}
set 1 1 180 90 {d} 1 {t}
set 1 2 90 0 {d} 1 {t}
set 1 3 0 90 {d} 1 {t}
set 1 4 90 180 {d} 1 {t}
set 1 5 180 0 {d} 1 {t}
set 4 6
start
""")

assert send('numtriggers') == '6\r\n'
print('Success')
```

```
Success
```

This produces the following scope trace:



```
MSO-X 3104T, MY56311244, 07.20.2017102614: Sat Jan 28 03:19:10 2023
50Ω  290mV/   2   3   4   14.00us/  42.80us  Trig'd?  D1  CMOS

Freq(1):            31.640kHz
Freq(2):            No signal
Freq(3):            Low signal
Freq(4):            29.189kHz
AC RMS - FS(1):     200.62mV
AC RMS - FS(2):     No signal
AC RMS - FS(3):     124mV
AC RMS - FS(4):     390mV

-293.50mV            113.050mV       15.0600V        10.6850V
DC   50Ω   1.00:1    DC   1.00:1     DC   1.00:1     DC   1.00:1

3:18 AM
Jan 28, 2023
```

D_1 is the IO_UPDATE line between the pico and the AD9959.
D_0 is the external trigger line into the pico.
The yellow trace is channel 1 from the AD9959
The pink trace is the output of a phase frequency detector between channels 0 and 1 of the AD9959