Getting Started Guide

RP2040 – AD9833 Dev board

DIY Waveform Generator

Table of Contents

1. Introduction

The purpose of this document is to serve as a getting started guide for the DIY waveform generator circuit. As such, it will provide a few examples that will demonstrate how to program the board using Thonny, blink the LEDs on the board, and generate a waveform output from the AD9833.

The circuit is basically a copy of the Raspberry Pi Pico board with the added circuity to support the AD9833 waveform generator IC. It is USB powered and is programmed exactly the same way a Raspberry Pi Pico board would be. The pinout of the Raspberry Pi Pico board from the datasheet is shown below for reference.

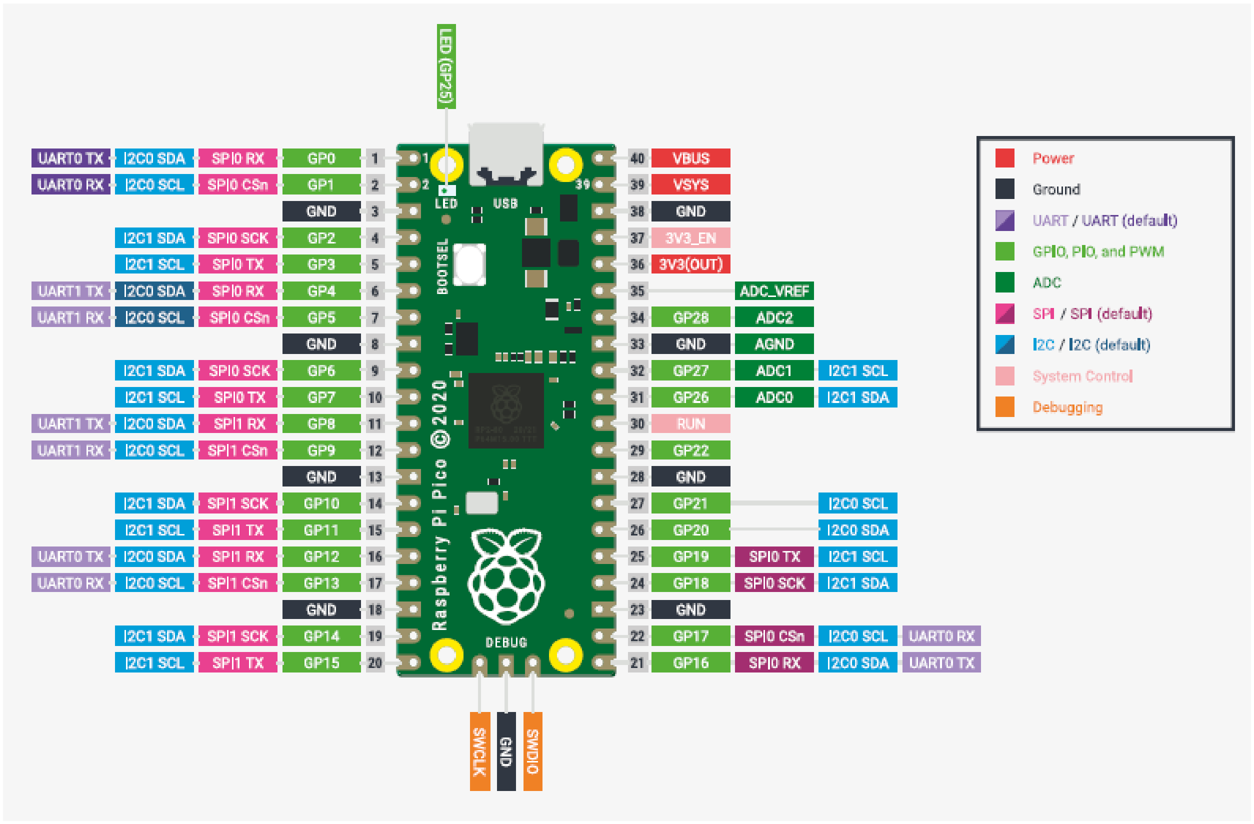


Figure 1. Raspberry Pi Pico board pinout

On the DIY waveform generator board there are 8 LEDs that are connected to GPIO pins 0 through 7 on the RP2040. In addition, the AD9833 SPI interface is connected to the GPIO pins 17, 18, and 19 on the RP2040, which is the default SPI interface as shown in the above pinout diagram. The headers on the board have additional pins broken out that include GPIO 23, 24, 25, 26, 27, 28, 29, and the SWCLK and SWD pins. These were broken out to headers for additional testing and prototyping possibilities.

The complete schematic of the circuit is shown below for reference:

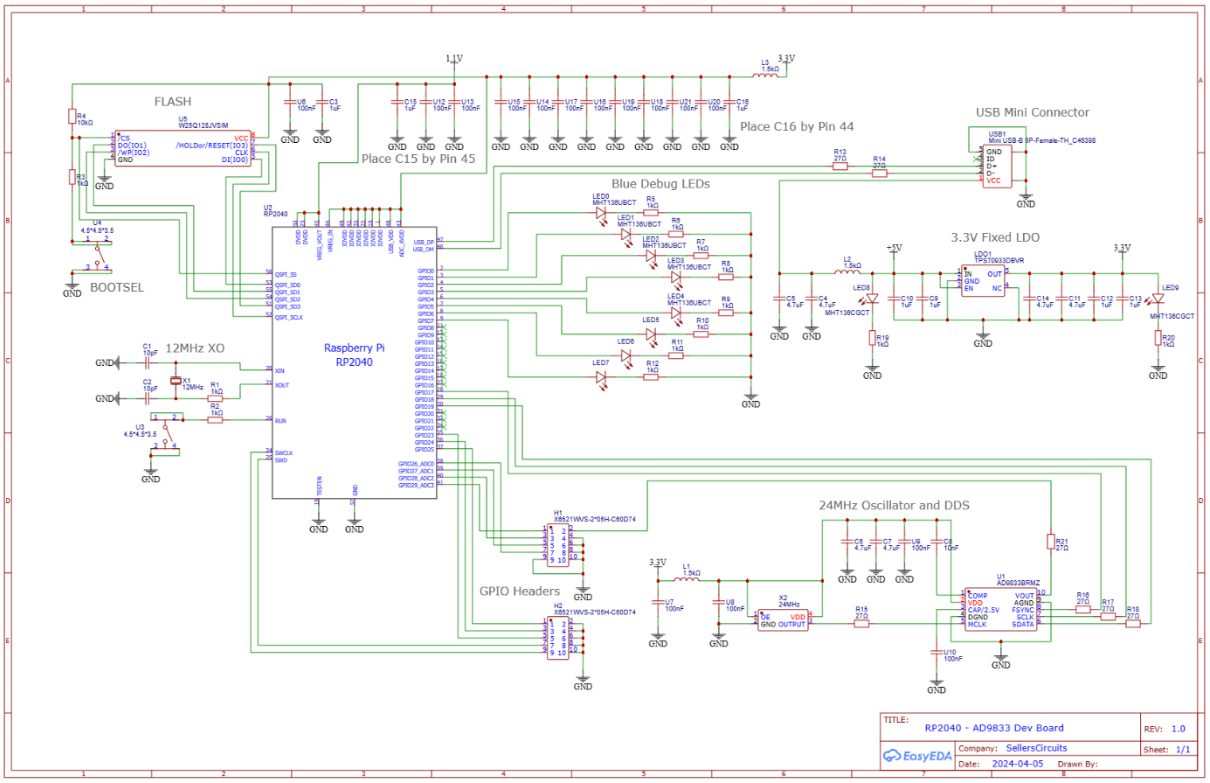


Figure 2. Schematic for the DIY waveform generator board

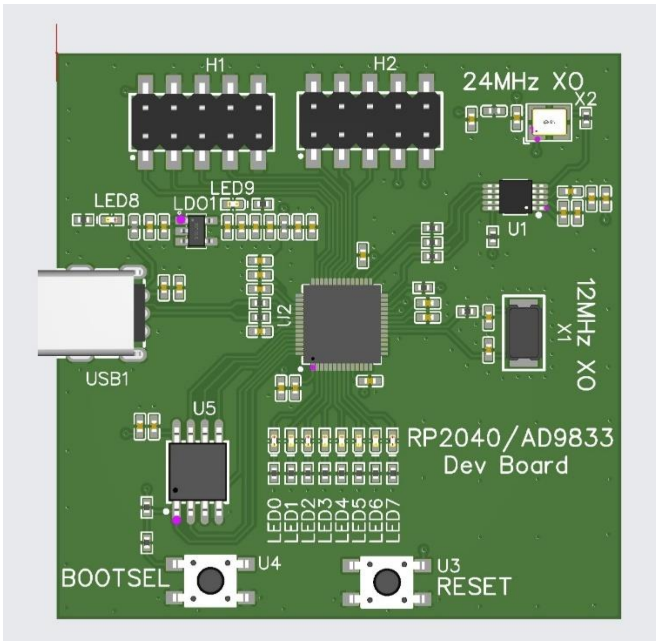
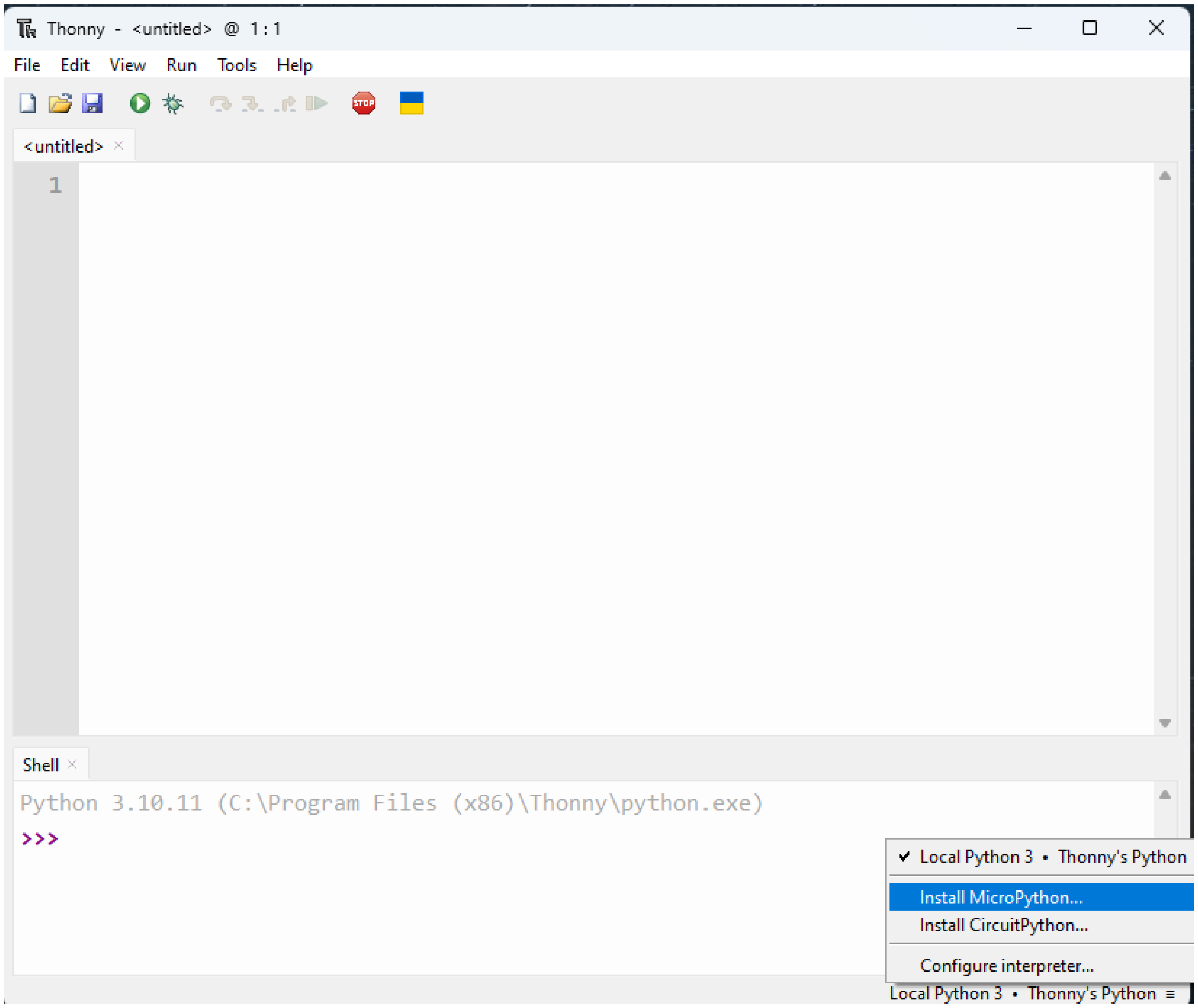


Figure 3. 3D rendering of the PCB layout and design

2. Programming the Board for the First Time

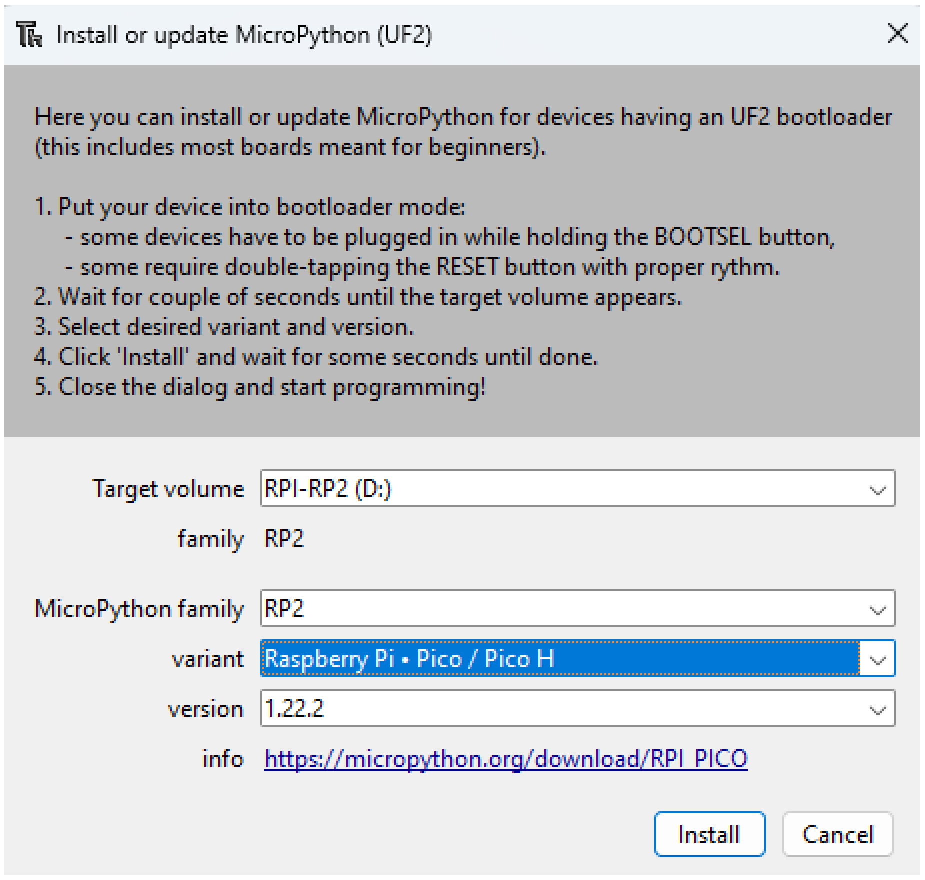
Upon powering up the boards for the first time, the “Boot select” button should be held down while plugging it into a computer. This puts the device into USB mass storage mode and allows us to drag and drop a file onto the board to program it. With that said we are going to hold the boot select button down, plug the device into a computer, and open up the Thonny IDE to install micropython on the device. With those steps completed, you should be able to click on the bottom right corner of the Thonny IDE to choose “Install MicroPython” as shown below:



```
Python 3.10.11 (C:\Program Files (x86)\Thonny\python.exe)
>>>
```

Figure 4. MicroPython can be installed by clicking on the bottom right area of the IDE

An additional window will pop up asking for the microython family, variant of board and version. I chose the latest version (which should populate automatically) and the RP2/Raspberry Pi Pico variant, since I have essentially copied the Pico board schematic.



Here you can install or update MicroPython for devices having an UF2 bootloader (this includes most boards meant for beginners).

1. Put your device into bootloader mode:
   - some devices have to be plugged in while holding the BOOTSEL button,
   - some require double-tapping the RESET button with proper rythm.
2. Wait for couple of seconds until the target volume appears.
3. Select desired variant and version.
4. Click 'Install' and wait for some seconds until done.
5. Close the dialog and start programming!

Target volume	RPI-RP2 (D:)
family	RP2
MicroPython family	RP2
variant	Raspberry Pi • Pico / Pico H
version	1.22.2
info	https://micropython.org/download/RPI_PICO

Figure 5. window with microPython family and variant options

After choosing install we can now start programming the device. Overall, the entire process should be identical to the process done on a Raspberry Pi Pico board.

3. Example – Blinking LEDS

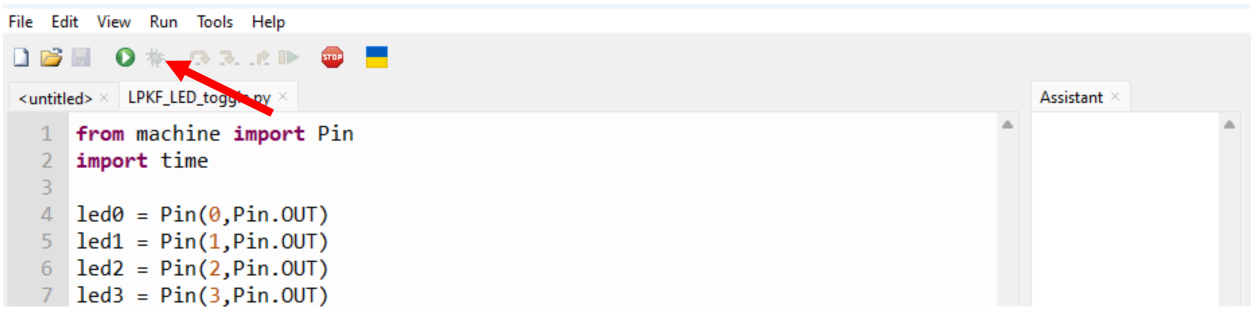
The following example will blink all eight LEDs on the board that are connected to the GPIO pins 0 through 7. This example will show how to set the pins to outputs and toggle them high and low. It also uses the time module in Python to add delay between turning the LEDs on and off. The code for this example is shown below:

```
from machine import Pin
import time

led0 = Pin(0,Pin.OUT)
led1 = Pin(1,Pin.OUT)
led2 = Pin(2,Pin.OUT)
led3 = Pin(3,Pin.OUT)
led4 = Pin(4,Pin.OUT)
led5 = Pin(5,Pin.OUT)
led6 = Pin(6,Pin.OUT)
led7 = Pin(7,Pin.OUT)
delay = 0.2

while(True):
    led1.value(1)
    time.sleep(delay)
    led1.value(0)
    led2.value(1)
    time.sleep(delay)
    led2.value(0)
    led3.value(1)
    time.sleep(delay)
    led3.value(0)
    led4.value(1)
    time.sleep(delay)
    led4.value(0)
    led5.value(1)
    time.sleep(delay)
    led5.value(0)
    led6.value(1)
    time.sleep(delay)
    led6.value(0)
    led7.value(1)
    time.sleep(delay)
    led7.value(0)
```

After copying or typing the code into Thonny, we should now be able to hit the green "run" button at the top of the Thonny IDE to run the code on the dev board.



```
from machine import Pin
import time

led0 = Pin(0,Pin.OUT)
led1 = Pin(1,Pin.OUT)
led2 = Pin(2,Pin.OUT)
led3 = Pin(3,Pin.OUT)
```

Figure 6. The green button at the top of the IDE is needed to load the code onto the RP2040 platform

The LEDs should now be blinking in a chasing pattern. That is one turns on at a time in sequential order.

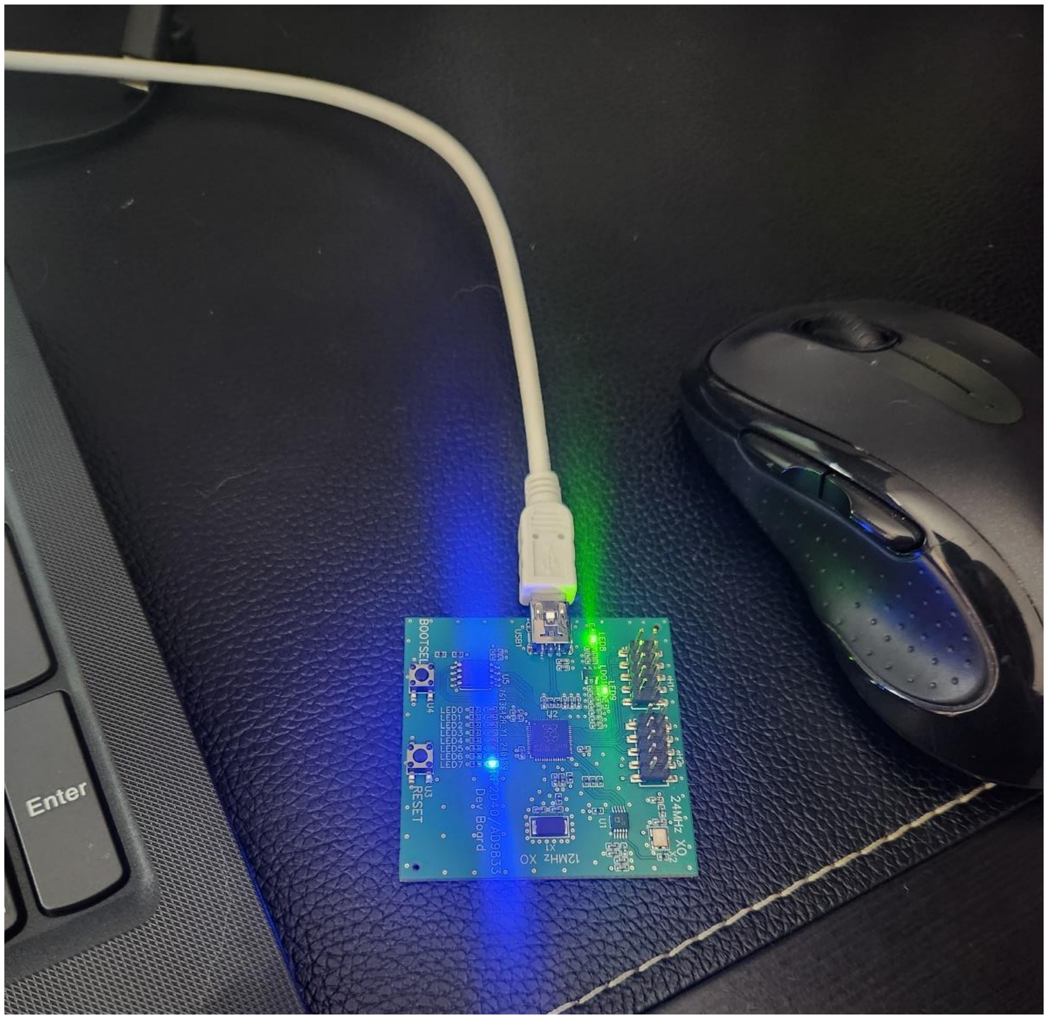


Figure 7. LED blinking in action

4. Example – Generating a Sinusoidal Waveform

This example will exercise the AD9833 waveform generator. The code below can be used to generate any frequency up to about 12MHz in sinusoidal or square wave outputs, as well as triangular waveforms. Furthermore, the code below is provided as a foundation to demonstrate how to use the waveform generator. Functionality can be added to improve the device as a tool such as a CLI interface. It is also important to note that the waveform generator output is connected to pin 2 of the H1 header. This is the pin in the upper left corner of the PCB.

The below code will generate a 1KHz sine wave output when loaded onto the dev board.

```
import machine
import utime

# AD9833 Register Address
REG_FREQ0 = 0x4000
REG_FREQ1 = 0x8000
REG_PHASE0 = 0xC000
REG_PHASE1 = 0xE000
REG_CONTROL = 0x2000

# AD9833 Control Bits
BIT_RESET = 0x0100
BIT_B28 = 0x2000
BIT_HLB = 0x8000

# SPI Configuration
spi = machine.SPI(0, baudrate=1000000, polarity=1, phase=0)
cs = machine.Pin(17, machine.Pin.OUT)

# Function to send data to AD9833
def send_data(data):
    cs.value(0)             # Select AD9833
    spi.write(data.to_bytes(2, 'big'))
    cs.value(1)             # Deselect AD9833

# Function to set the frequency of AD9833
def set_frequency(freq):
    freq_reg = int((freq * (2**28)) / 24000000)
    print('Freq_reg value:'+str(freq_reg))
    print('freq00 register value: '+str(REG_FREQ0 | (freq_reg & 0x3FFF)))
    print('freq01 register value: '+str(REG_FREQ0 | ((freq_reg >> 14) & 
0x3FFF)))
    send_data(REG_FREQ0 | (freq_reg & 0x3FFF))
    send_data(REG_FREQ0 | ((freq_reg >> 14) & 0x3FFF))

# Function to set the phase of AD9833
def set_phase(phase):
    phase_reg = int(phase * (2**12 / 360))
    send_data(REG_PHASE0 | (phase_reg & 0x0FFF))
    send_data(REG_PHASE1 | ((phase_reg >> 14) & 0x0FFF))
```

```
# Function to reset AD9833
def reset():
    send_data(REG_CONTROL | BIT_RESET)
    utime.sleep_us(10)

def wakeUp():
    send_data(REG_CONTROL | 0x0000)

def setSine():
    send_data(REG_CONTROL | 0x0000)

def setTriangle():
    send_data(REG_CONTROL | 0x0002)

def setSquare():
    send_data(REG_CONTROL | 0x0028)

# Initialize AD9833
reset()
send_data(REG_CONTROL | BIT_HLB)  # Set control bits for AD9833

# Set frequency and phase
set_frequency(1000)  # Set frequency to 1000 Hz
set_phase(0)        # Set phase to 0 degrees
wakeUp()
```

If an oscilloscope is on-hand you should be able to probe the output pin of the AD9833 or connect it to pin 2 of the H1 header and measure a 1KHz sinusoidal waveform. This is shown below:

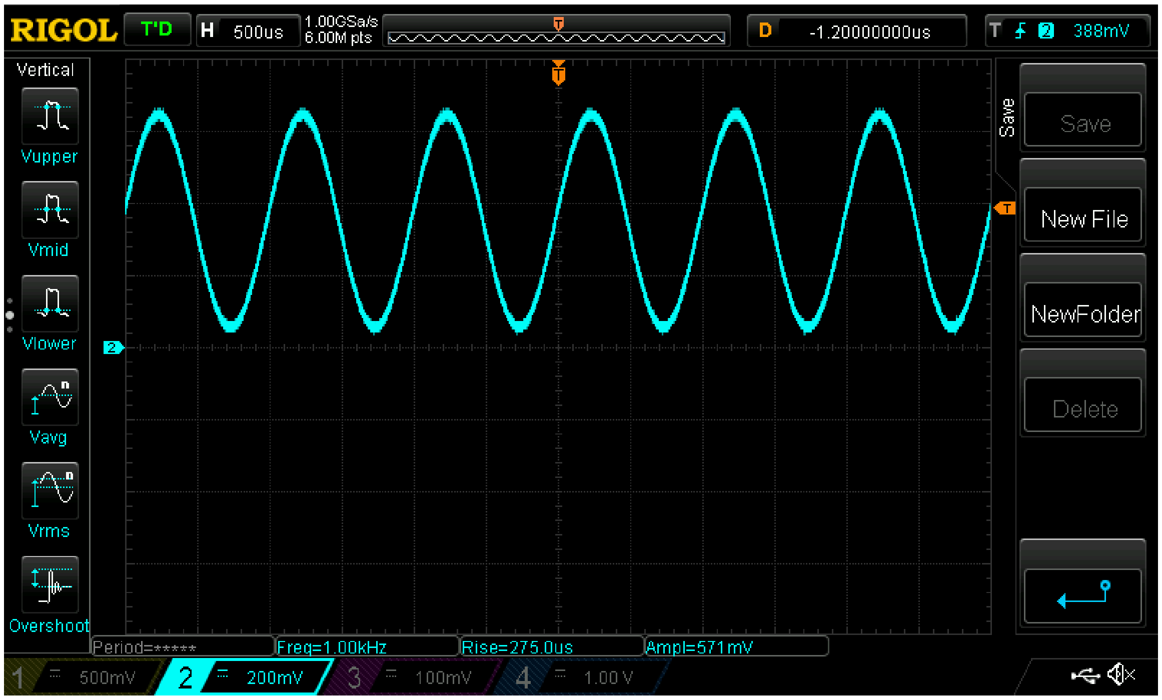


Figure 8. 1KHz Sine wave from the AD9833