Hardware Competition Timer: Wiring and Code

February 2024

Contributors: Steven Butler Steele
Razvan Voicu

Contents

List of Figures

1 Wiring Diagram

The foot pedal switch is in place of the standard button.

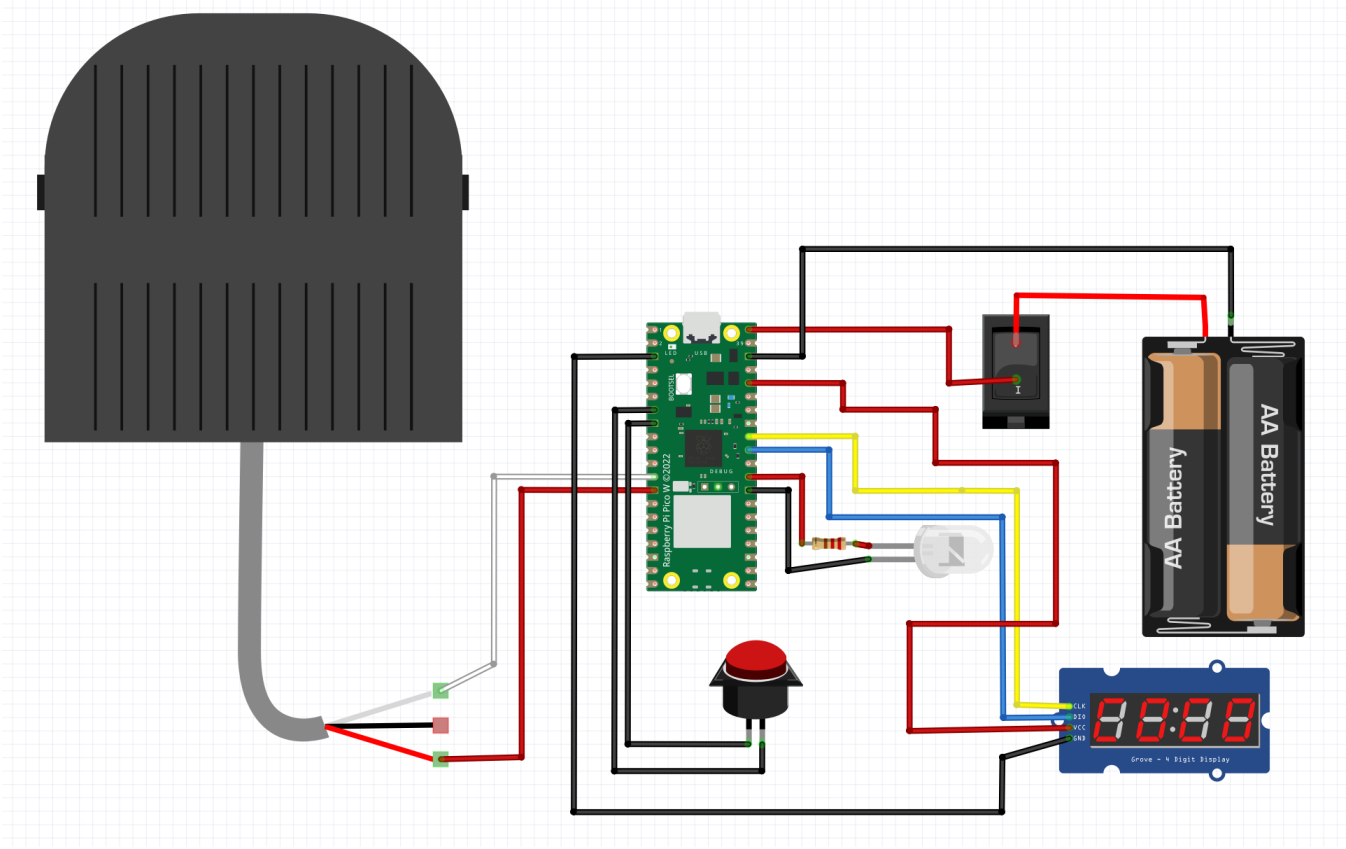

Figure 1: Wiring diagram

2 Code

Main code used with the Rasperry Pi Pico W. To upload code to your microcontroller, you must first install Micropython on the device.

2.1 main.py

```
import time
from time import sleep
import tm1637
from machine import Pin
tm = tm1637.TM1637(clk=Pin(27), dio=Pin(26))
led = Pin(22, Pin.OUT)

tm.brightness(1)

interrupt_flag=0
debounce_time=0
```

```
stop = Pin(5, Pin.IN, Pin.PULL_UP)

def callback(stop):
    global interrupt_flag, debounce_time
    if (time.ticks_ms()-debounce_time) > 500:
        interrupt_flag= 1
        debounce_time=time.ticks_ms()

stop.irq(trigger=Pin.IRQ_FALLING, handler=callback)

start = Pin(9, Pin.IN, Pin.PULL_UP)
min = 0
sec = 0
led.value(0)
is_counting = True
while True:
    tm.numbers(min, sec)
    if start.value() == 0:
        is_counting = True
        min = 0
        sec = 0
        tm.numbers(min, sec)
        led.value(1)

        for i in range(1,4):
            for x in range(1, 60):
                sleep(1)
                if interrupt_flag is 1:
                    interrupt_flag=0
                    print("Interrupt Detected")
                    is_counting = False
                    led.value(0)
                    break
                sec = x
                tm.numbers(min, sec)

            if not is_counting:
                break

            sec = 0
            min = i
            tm.numbers(min, sec)
            if start.value ==1 and not is_counting:
                min = 0
                sec = 0

        led.value(0)
```

2.2 tm1637.py

This code is the library that the timer will use to drive the seven segment display. Please keep in mind that the code wraps around. For example _SEGMENTS is a single line

```
"""
MicroPython TM1637 quad 7-segment LED display driver
https://github.com/mcauser/micropython-tm1637

MIT License
Copyright (c) 2016-2023 Mike Causer

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""

__version__ = '1.3.0'

from micropython import const
from machine import Pin
from time import sleep_us, sleep_ms

TM1637_CMD1 = const(64)  # 0x40 data command
TM1637_CMD2 = const(192) # 0xC0 address command
TM1637_CMD3 = const(128) # 0x80 display control command
TM1637_DSP_ON = const(8) # 0x08 display on
TM1637_DELAY = const(10) # 10us delay between clk/dio pulses
TM1637_MSB = const(128)  # msb is the decimal point or the colon depending on your display

# 0-9, a-z, blank, dash, star
_SEGMENTS = bytearray(b'\x3F\x06\x5B\x4F\x66\x6D\x7D\x07\x7F\x6F
\x77\x7C\x39\x5E\x79\x71\x3D\x76\x06\x1E\x76\x38\x55\x54\x3F
```

```
\x73\x67\x50\x6D\x78\x3E\x1C\x2A\x76\x6E\x5B\x00\x40\x63')

class TM1637(object):
    """Library for quad 7-segment LED modules based on the TM1637 LED driver."""
    def __init__(self, clk, dio, brightness=7):
        self.clk = clk
        self.dio = dio

        if not 0 <= brightness <= 7:
            raise ValueError("Brightness out of range")
        self._brightness = brightness

        self.clk.init(Pin.OUT, value=0)
        self.dio.init(Pin.OUT, value=0)
        sleep_us(TM1637_DELAY)

        self._write_data_cmd()
        self._write_dsp_ctrl()

    def _start(self):
        self.dio(0)
        sleep_us(TM1637_DELAY)
        self.clk(0)
        sleep_us(TM1637_DELAY)

    def _stop(self):
        self.dio(0)
        sleep_us(TM1637_DELAY)
        self.clk(1)
        sleep_us(TM1637_DELAY)
        self.dio(1)

    def _write_data_cmd(self):
        # automatic address increment, normal mode
        self._start()
        self._write_byte(TM1637_CMD1)
        self._stop()

    def _write_dsp_ctrl(self):
        # display on, set brightness
        self._start()
        self._write_byte(TM1637_CMD3 | TM1637_DSP_ON | self._brightness)
        self._stop()

    def _write_byte(self, b):
        for i in range(8):
            self.dio((b >> i) & 1)
            sleep_us(TM1637_DELAY)
```

```
            self.clk(1)
            sleep_us(TM1637_DELAY)
            self.clk(0)
            sleep_us(TM1637_DELAY)
        self.clk(0)
        sleep_us(TM1637_DELAY)
        self.clk(1)
        sleep_us(TM1637_DELAY)
        self.clk(0)
        sleep_us(TM1637_DELAY)

    def brightness(self, val=None):
        """Set the display brightness 0-7."""
        # brightness 0 = 1/16th pulse width
        # brightness 7 = 14/16th pulse width
        if val is None:
            return self._brightness
        if not 0 <= val <= 7:
            raise ValueError("Brightness out of range")

        self._brightness = val
        self._write_data_cmd()
        self._write_dsp_ctrl()

    def write(self, segments, pos=0):
        """Display up to 6 segments moving right from a given position.
        The MSB in the 2nd segment controls the colon between the 2nd
        and 3rd segments."""
        if not 0 <= pos <= 5:
            raise ValueError("Position out of range")
        self._write_data_cmd()
        self._start()

        self._write_byte(TM1637_CMD2 | pos)
        for seg in segments:
            self._write_byte(seg)
        self._stop()
        self._write_dsp_ctrl()

    def encode_digit(self, digit):
        """Convert a character 0-9, a-f to a segment."""
        return _SEGMENTS[digit & 0x0f]

    def encode_string(self, string):
        """Convert an up to 4 character length string containing 0-9, a-z,
        space, dash, star to an array of segments, matching the length of the
        source string."""
        segments = bytearray(len(string))
```

```
        for i in range(len(string)):
            segments[i] = self.encode_char(string[i])
        return segments

    def encode_char(self, char):
        """Convert a character 0-9, a-z, space, dash or star to a segment."""
        o = ord(char)
        if o == 32:
            return _SEGMENTS[36] # space
        if o == 42:
            return _SEGMENTS[38] # star/degrees
        if o == 45:
            return _SEGMENTS[37] # dash
        if o >= 65 and o <= 90:
            return _SEGMENTS[o-55] # uppercase A-Z
        if o >= 97 and o <= 122:
            return _SEGMENTS[o-87] # lowercase a-z
        if o >= 48 and o <= 57:
            return _SEGMENTS[o-48] # 0-9
        raise ValueError("Character out of range: {:d} '{:s}'".format(o, chr(o)))

    def hex(self, val):
        """Display a hex value 0x0000 through 0xffff, right aligned."""
        string = '{:04x}'.format(val & 0xffff)
        self.write(self.encode_string(string))

    def number(self, num):
        """Display a numeric value -999 through 9999, right aligned."""
        # limit to range -999 to 9999
        num = max(-999, min(num, 9999))
        string = '{0: >4d}'.format(num)
        self.write(self.encode_string(string))

    def numbers(self, num1, num2, colon=True):
        """Display two numeric values -9 through 99, with leading zeros
        and separated by a colon."""
        num1 = max(-9, min(num1, 99))
        num2 = max(-9, min(num2, 99))
        segments = self.encode_string('{0:0>2d}{1:0>2d}'.format(num1, num2))
        if colon:
            segments[1] |= 0x80 # colon on
        self.write(segments)

    def temperature(self, num):
        if num < -9:
            self.show('lo') # low
        elif num > 99:
            self.show('hi') # high
```

```
        else:
            string = '{0: >2d}'.format(num)
            self.write(self.encode_string(string))
        self.write([_SEGMENTS[38], _SEGMENTS[12]], 2) # degrees C

    def show(self, string, colon=False):
        segments = self.encode_string(string)
        if len(segments) > 1 and colon:
            segments[1] |= 128
        self.write(segments[:4])

    def scroll(self, string, delay=250):
        segments = string if isinstance(string, list) else self.encode_string(string)
        data = [0] * 8
        data[4:0] = list(segments)
        for i in range(len(segments) + 5):
            self.write(data[0+i:4+i])
            sleep_ms(delay)


class TM1637Decimal(TM1637):
    """Library for quad 7-segment LED modules based on the TM1637 LED driver.

    This class is meant to be used with decimal display modules (modules
    that have a decimal point after each 7-segment LED).
    """

    def encode_string(self, string):
        """Convert a string to LED segments.
        Convert an up to 4 character length string containing 0-9, a-z,
        space, dash, star and '.' to an array of segments, matching the length of
        the source string."""
        segments = bytearray(len(string.replace('.','')))
        j = 0
        for i in range(len(string)):
            if string[i] == '.' and j > 0:
                segments[j-1] |= TM1637_MSB
                continue
            segments[j] = self.encode_char(string[i])
            j += 1
        return segments
```