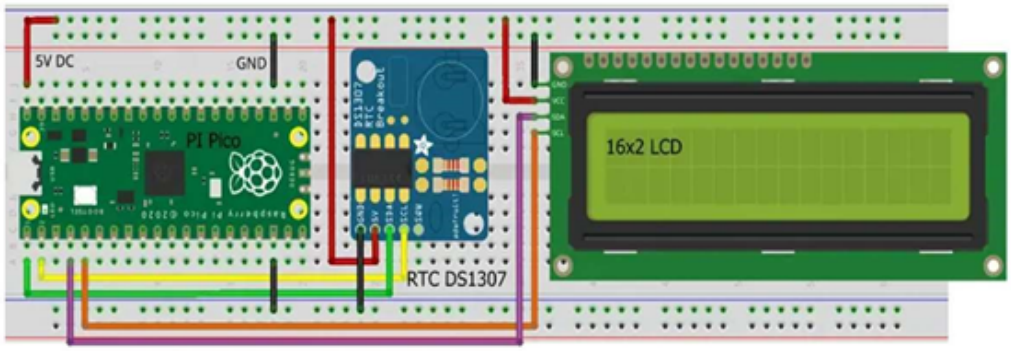
Project Name:Digital clock Using Raspberry Pi Pico	
Group No: 6	Roll No: 13,19,21,25
Software:	fritzing: 

Code:	#ds1302.py
	from machine import Pin
	
	DS1302_REG_SECOND = (0x80)
	DS1302_REG_MINUTE = (0x82)
	DS1302_REG_HOUR = (0x84)
	DS1302_REG_DAY = (0x86)
	DS1302_REG_MONTH = (0x88)
	DS1302_REG_WEEKDAY= (0x8A)
	DS1302_REG_YEAR = (0x8C)
	DS1302_REG_WP = (0x8E)
	DS1302_REG_CTRL = (0x90)
	DS1302_REG_RAM = (0xC0)
	
	class DS1302:
	def __init__(self, clk, dio, cs):
	self.clk = clk
	self.dio = dio
	self.cs = cs
	self.clk.init(Pin.OUT)
	self.cs.init(Pin.OUT)
	
	def _dec2hex(self, dat):
	return (dat//10) * 16 + (dat % 10)
	
	def _hex2dec(self, dat):
	return (dat//16) * 10 + (dat % 16)
	
	def _write_byte(self, dat):
	self.dio.init(Pin.OUT)

```
    for i in range(8):
        self.dio.value((dat >> i) & 1)
        self.clk.value(1)
        self.clk.value(0)

def _read_byte(self):
    d = 0
    self.dio.init(Pin.IN)
    for i in range(8):
        d = d | (self.dio.value() << i)
        self.clk.value(1)
        self.clk.value(0)
    return d

def _get_reg(self, reg):
    self.cs.value(1)
    self._write_byte(reg)
    t = self._read_byte()
    self.cs.value(0)
    return t

def _set_reg(self, reg, dat):
    self.cs.value(1)
    self._write_byte(reg)
    self._write_byte(dat)
    self.cs.value(0)

def _wr(self, reg, dat):
    self._set_reg(DS1302_REG_WP, 0)
    self._set_reg(reg, dat)
    self._set_reg(DS1302_REG_WP, 0x80)

def start(self):
    t = self._get_reg(DS1302_REG_SECOND + 1)
    self._wr(DS1302_REG_SECOND, t & 0x7f)

def stop(self):
    t = self._get_reg(DS1302_REG_SECOND + 1)
    self._wr(DS1302_REG_SECOND, t | 0x80)

def second(self, second=None):
    if second == None:
        return self._hex2dec(self._get_reg(DS1302_REG_SECOND+1)) % 60
    else:
        self._wr(DS1302_REG_SECOND, self._dec2hex(second % 60))

def minute(self, minute=None):
    if minute == None:
        return self._hex2dec(self._get_reg(DS1302_REG_MINUTE+1))
    else:
        self._wr(DS1302_REG_MINUTE, self._dec2hex(minute % 60))

def hour(self, hour=None):
```

```
        if hour == None:
            return self._hex2dec(self._get_reg(DS1302_REG_HOUR+1))
        else:
            self._wr(DS1302_REG_HOUR, self._dec2hex(hour % 24))

    def weekday(self, weekday=None):
        if weekday == None:
            return self._hex2dec(self._get_reg(DS1302_REG_WEEKDAY+1))
        else:
            self._wr(DS1302_REG_WEEKDAY, self._dec2hex(weekday % 8))

    def day(self, day=None):
        if day == None:
            return self._hex2dec(self._get_reg(DS1302_REG_DAY+1))
        else:
            self._wr(DS1302_REG_DAY, self._dec2hex(day % 32))

    def month(self, month=None):
        if month == None:
            return self._hex2dec(self._get_reg(DS1302_REG_MONTH+1))
        else:
            self._wr(DS1302_REG_MONTH, self._dec2hex(month % 13))

    def year(self, year=None):
        if year == None:
            return self._hex2dec(self._get_reg(DS1302_REG_YEAR+1)) + 2000
        else:
            self._wr(DS1302_REG_YEAR, self._dec2hex(year % 100))

    def date_time(self, dat=None):
        if dat == None:
            return [self.year(), self.month(), self.day(), self.weekday(), self.hour(), self.minute(),
self.second()]
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])
            self.weekday(dat[3])
            self.hour(dat[4])
            self.minute(dat[5])
            self.second(dat[6])

    def ram(self, reg, dat=None):
        if dat == None:
            return self._get_reg(DS1302_REG_RAM + 1 + (reg % 31)*2)
        else:
            self._wr(DS1302_REG_RAM + (reg % 31)*2, dat)

#lcd.py
import time

class LcdApi:
```

```
# Implements the API for talking with HD44780 compatible character LCDs.
# This class only knows what commands to send to the LCD, and not how to get
# them to the LCD.
#
# It is expected that a derived class will implement the hal_xxx functions.
#
# The following constant names were lifted from the avrlib lcd.h header file,
# with bit numbers changed to bit masks.

# HD44780 LCD controller command set
LCD_CLR           = 0x01  # DB0: clear display
LCD_HOME          = 0x02  # DB1: return to home position

LCD_ENTRY_MODE    = 0x04  # DB2: set entry mode
LCD_ENTRY_INC     = 0x02  # DB1: increment
LCD_ENTRY_SHIFT   = 0x01  # DB0: shift

LCD_ON_CTRL       = 0x08  # DB3: turn lcd/cursor on
LCD_ON_DISPLAY    = 0x04  # DB2: turn display on
LCD_ON_CURSOR     = 0x02  # DB1: turn cursor on
LCD_ON_BLINK      = 0x01  # DB0: blinking cursor

LCD_MOVE          = 0x10  # DB4: move cursor/display
LCD_MOVE_DISP     = 0x08  # DB3: move display (0-> move cursor)
LCD_MOVE_RIGHT    = 0x04  # DB2: move right (0-> left)

LCD_FUNCTION      = 0x20  # DB5: function set
LCD_FUNCTION_8BIT  = 0x10  # DB4: set 8BIT mode (0->4BIT mode)
LCD_FUNCTION_2LINES = 0x08  # DB3: two lines (0->one line)
LCD_FUNCTION_10DOTS = 0x04  # DB2: 5x10 font (0->5x7 font)
LCD_FUNCTION_RESET  = 0x30  # See "Initializing by Instruction" section

LCD_CGRAM         = 0x40  # DB6: set CG RAM address
LCD_DDRAM         = 0x80  # DB7: set DD RAM address

LCD_RS_CMD        = 0
LCD_RS_DATA       = 1

LCD_RW_WRITE      = 0
LCD_RW_READ       = 1

def __init__(self, num_lines, num_columns):
   self.num_lines = num_lines
   if self.num_lines > 4:
      self.num_lines = 4
   self.num_columns = num_columns
   if self.num_columns > 40:
      self.num_columns = 40
   self.cursor_x = 0
   self.cursor_y = 0
   self.implied_newline = False
   self.backlight = True
   self.display_off()
```

```
        self.backlight_on()
        self.clear()
        self.hal_write_command(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)
        self.hide_cursor()
        self.display_on()

    def clear(self):
        # Clears the LCD display and moves the cursor to the top left corner
        self.hal_write_command(self.LCD_CLR)
        self.hal_write_command(self.LCD_HOME)
        self.cursor_x = 0
        self.cursor_y = 0

    def show_cursor(self):
        # Causes the cursor to be made visible
        self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                               self.LCD_ON_CURSOR)

    def hide_cursor(self):
        # Causes the cursor to be hidden
        self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def blink_cursor_on(self):
        # Turns on the cursor, and makes it blink
        self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                               self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

    def blink_cursor_off(self):
        # Turns on the cursor, and makes it no blink (i.e. be solid)
        self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                               self.LCD_ON_CURSOR)

    def display_on(self):
        # Turns on (i.e. unblanks) the LCD
        self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def display_off(self):
        # Turns off (i.e. blanks) the LCD
        self.hal_write_command(self.LCD_ON_CTRL)

    def backlight_on(self):
        # Turns the backlight on.

        # This isn't really an LCD command, but some modules have backlight
        # controls, so this allows the hal to pass through the command.
        self.backlight = True
        self.hal_backlight_on()

    def backlight_off(self):
        # Turns the backlight off.

        # This isn't really an LCD command, but some modules have backlight
        # controls, so this allows the hal to pass through the command.
```

```
    self.backlight = False
    self.hal_backlight_off()

  def move_to(self, cursor_x, cursor_y):
    # Moves the cursor position to the indicated position. The cursor
    # position is zero based (i.e. cursor_x == 0 indicates first column).
    self.cursor_x = cursor_x
    self.cursor_y = cursor_y
    addr = cursor_x & 0x3f
    if cursor_y & 1:
      addr += 0x40    # Lines 1 & 3 add 0x40
    if cursor_y & 2:    # Lines 2 & 3 add number of columns
      addr += self.num_columns
    self.hal_write_command(self.LCD_DDRAM | addr)

  def putchar(self, char):
    # Writes the indicated character to the LCD at the current cursor
    # position, and advances the cursor by one position.
    if char == '\n':
      if self.implied_newline:
        # self.implied_newline means we advanced due to a wraparound,
        # so if we get a newline right after that we ignore it.
        pass
      else:
        self.cursor_x = self.num_columns
    else:
      self.hal_write_data(ord(char))
      self.cursor_x += 1
    if self.cursor_x >= self.num_columns:
      self.cursor_x = 0
      self.cursor_y += 1
      self.implied_newline = (char != '\n')
    if self.cursor_y >= self.num_lines:
      self.cursor_y = 0
    self.move_to(self.cursor_x, self.cursor_y)

  def putstr(self, string):
    # Write the indicated string to the LCD at the current cursor
    # position and advances the cursor position appropriately.
    for char in string:
      self.putchar(char)

  def custom_char(self, location, charmap):
    # Write a character to one of the 8 CGRAM locations, available
    # as chr(0) through chr(7).
    location &= 0x7
    self.hal_write_command(self.LCD_CGRAM | (location << 3))
    self.hal_sleep_us(40)
    for i in range(8):
      self.hal_write_data(charmap[i])
      self.hal_sleep_us(40)
    self.move_to(self.cursor_x, self.cursor_y)
```

```
    def hal_backlight_on(self):
        # Allows the hal layer to turn the backlight on.
        # If desired, a derived HAL class will implement this function.
        pass

    def hal_backlight_off(self):
        # Allows the hal layer to turn the backlight off.
        # If desired, a derived HAL class will implement this function.
        pass

    def hal_write_command(self, cmd):
        # Write a command to the LCD.
        # It is expected that a derived HAL class will implement this function.
        raise NotImplementedError

    def hal_write_data(self, data):
        # Write data to the LCD.
        # It is expected that a derived HAL class will implement this function.
        raise NotImplementedError

    def hal_sleep_us(self, usecs):
        # Sleep for some time (given in microseconds)
        time.sleep_us(usecs)
#lcd_api.py
import utime
import gc

from lcd_api import LcdApi
from machine import I2C

# PCF8574 pin definitions
MASK_RS = 0x01       # P0
MASK_RW = 0x02       # P1
MASK_E  = 0x04       # P2

SHIFT_BACKLIGHT = 3  # P3
SHIFT_DATA      = 4  # P4-P7

class I2cLcd(LcdApi):

    #Implements a HD44780 character LCD connected via PCF8574 on I2C

    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.i2c.writeto(self.i2c_addr, bytes([0]))
        utime.sleep_ms(20)   # Allow LCD time to powerup
        # Send reset 3 times
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(5)    # Need to delay at least 4.1 msec
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
```

```
    utime.sleep_ms(1)
    # Put LCD into 4-bit mode
    self.hal_write_init_nibble(self.LCD_FUNCTION)
    utime.sleep_ms(1)
    LcdApi.__init__(self, num_lines, num_columns)
    cmd = self.LCD_FUNCTION
    if num_lines > 1:
      cmd |= self.LCD_FUNCTION_2LINES
    self.hal_write_command(cmd)
    gc.collect()

  def hal_write_init_nibble(self, nibble):
    # Writes an initialization nibble to the LCD.
    # This particular function is only used during initialization.
    byte = ((nibble >> 4) & 0x0f) << SHIFT_DATA
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    gc.collect()

  def hal_backlight_on(self):
    # Allows the hal layer to turn the backlight on
    self.i2c.writeto(self.i2c_addr, bytes([1 << SHIFT_BACKLIGHT]))
    gc.collect()

  def hal_backlight_off(self):
    #Allows the hal layer to turn the backlight off
    self.i2c.writeto(self.i2c_addr, bytes([0]))
    gc.collect()

  def hal_write_command(self, cmd):
    # Write a command to the LCD. Data is latched on the falling edge of E.
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
          (((cmd >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
          ((cmd & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    if cmd <= 3:
      # The home and clear commands require a worst case delay of 4.1 msec
      utime.sleep_ms(5)
    gc.collect()

  def hal_write_data(self, data):
    # Write data to the LCD. Data is latched on the falling edge of E.
    byte = (MASK_RS |
          (self.backlight << SHIFT_BACKLIGHT) |
          (((data >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    byte = (MASK_RS |
          (self.backlight << SHIFT_BACKLIGHT) |
```

```
            ((data & 0x0f) << SHIFT_DATA))
        self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytes([byte]))
        gc.collect()
#main.py

"""

DS1302 real-time clock module

Special thanks to yunline for the DS1302  Library:
https://github.com/omarbenhamid/micropython-ds1302-rtc
and T-622 for the I2C LCD library: https://github.com/T-622/RPI-PICO-I2C-LCD

Remember to check out more tutorials on NerdCave - https://www.youtube.com/c/NerdCaveYT

Project Pinout
VCC - VSYS (PIN39)
GND - GND (Any ground on Pico)
CLK - GP18 (PIN24)
DAT - GP17 (PIN22)
RST  - GP16 (PIN21)

"""


from machine import I2C, Pin
from ds1302 import DS1302
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 63
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

ds = DS1302(Pin(18),Pin(17),Pin(16))

ds.date_time() # returns the current datetime.

#ds.date_time([2023, 3, 2, 0, 8, 17, 50, 0]) # set datetime.

#ds.hour() # returns hour.
#print(ds.date_time())


while True:
  (Y,M,D,day,hr,m,s)=ds.date_time()
  if s < 10:
    s = "0" + str(s)
  if m < 10:
    m = "0" + str(m)
```

	if hr < 10:
	  hr = "0" + str(hr)
	if D < 10:
	  D = "0" + str(D)
	if M < 10:
	  M = "0" + str(M)
	
	lcd.move_to(0,0)
	lcd.putstr("Time:")
	lcd.move_to(6,0)
	lcd.putstr(str(hr) + ":" + str(m) + ":" + str(s))
	lcd.move_to(0,1)
	lcd.putstr("Date:")
	lcd.move_to(6,1)
	lcd.putstr(str(D) + "/" + str(M) + "/" + str(Y))

Reference	1. https://www.instructables.com/Raspberry-Pi-Pico-Alarm-Clock/