```
#!/usr/bin/env python

# SDL_DS3231.py Python Driver Code
# SwitchDoc Labs 12/19/2014
# V 1.2
# only works in 24 hour mode
# now includes reading and writing the AT24C32 included on the SwitchDoc Labs
#       DS3231 / AT24C32 Module (www.switchdoc.com)
# please send patches to: https://github.com/switchdoclabs/RTC_SDL_DS3231


#encoding: utf-8

# Copyright (C) 2013 @XiErCh
#
# 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.

from __future__ import print_function

from datetime import datetime
import time

import smbus


SECONDS_PER_MINUTE = 60
MINUTES_PER_HOUR = 60
HOURS_PER_DAY = 24
DAYS_PER_WEEK = 7
MAX_DAYS_PER_MONTH = 31
MONTHS_PER_YEAR = 12
YEARS_PER_CENTURY = 100

OSCILLATOR_ON_MASK = 0b1<<7

def bcd_to_int(bcd, n=2):
    """Decode n least significant packed binary coded decimal digits to binary.
    Return binary result.
    n defaults to 2 (BCD digits).
    n=0 decodes all digits.
    """
    return int(('%x' % bcd)[-n:])


def int_to_bcd(x, n=2):
    """
    Encode the n least significant decimal digits of x
    to packed binary coded decimal (BCD).
    Return packed BCD value.
    n defaults to 2 (digits).
    n=0 encodes all digits.
    """
    return int(str(x)[-n:], 0x10)


class SDL_DS3231():
    (
        _REG_SECONDS,
        _REG_MINUTES,
```

```
            _REG_HOURS,
            _REG_DAY,
            _REG_DATE,
            _REG_MONTH,
            _REG_YEAR,
        ) = range(7)

        ############################
        # DS3231 Code
        # datasheet: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf
        ############################
        def __init__(self, twi=1, addr=0x68, at24c32_addr=0x56):
            self._bus = smbus.SMBus(twi)
            self._addr = addr
            self._at24c32_addr = at24c32_addr

        def _write(self, register, data):
            if False:
                print(
                    "addr =0x%x register = 0x%x data = 0x%x %i " %
                    (self._addr, register, data, bcd_to_int(data)))
            self._bus.write_byte_data(self._addr, register, data)

        def _read(self, register_address):
            data = self._bus.read_byte_data(self._addr, register_address)
            if False:
                print(
                    "addr = 0x%x register_address = 0x%x %i data = 0x%x %i "
                    % (
                        self._addr, register_address, register_address,
                        data, bcd_to_int(data)))
            return data

        def _incoherent_read_all(self):
            """Return tuple of year, month, date, day, hours, minutes, seconds.
            Since each value is read one byte at a time,
            it might not be coherent."""

            register_addresses = (
                self._REG_SECONDS,
                self._REG_MINUTES,
                self._REG_HOURS,
                self._REG_DAY,
                self._REG_DATE,
                self._REG_MONTH,
                self._REG_YEAR,
            )
            seconds, minutes, hours, day, date, month, year = (
                self._read(register_address)
                for register_address in register_addresses
            )
            seconds &= ~OSCILLATOR_ON_MASK
            if True:
                # This stuff is suspicious.
                if hours == 0x64:
                    hours = 0x40
                hours &= 0x3F
            return tuple(
                bcd_to_int(t)
                for t in (year, month, date, day, hours, minutes, seconds))

        def read_all(self):
            """Return tuple of year, month, date, day, hours, minutes, seconds.
            """

            """Read until one gets same result twice in a row.
            Then one knows the time is coherent."""

            old = self._incoherent_read_all()
            while True:
                new = self._incoherent_read_all()
                if old == new:
                    break
                old = new
            return new
```

```
151
152     def read_str(self):
153         """Return a string such as 'YY-DD-MMTHH-MM-SS'.
154         """
155         year, month, date, _, hours, minutes, seconds = self.read_all()
156         return (
157             '%02d-%02d-%02dT%02d:%02d:%02d' %
158             (year, month, date, hours, minutes, seconds)
159         )
160
161     def read_datetime(self, century=21, tzinfo=None):
162         """Return the datetime.datetime object.
163         """
164         year, month, date, _, hours, minutes, seconds = self.read_all()
165         year = 100 * (century - 1) + year
166         return datetime(
167             year, month, date, hours, minutes, seconds,
168             0, tzinfo=tzinfo)
169
170     def write_all(self, seconds=None, minutes=None, hours=None, day=None,
171             date=None, month=None, year=None, save_as_24h=True):
172         """Direct write un-none value.
173         Range: seconds [0,59], minutes [0,59], hours [0,23],
174                day [0,7], date [1-31], month [1-12], year [0-99].
175         """
176         if seconds is not None:
177             if not 0 <= seconds < SECONDS_PER_MINUTE:
178                 raise ValueError('Seconds is out of range [0,59].')
179             seconds_reg = int_to_bcd(seconds)
180             self._write(self._REG_SECONDS, seconds_reg)
181
182         if minutes is not None:
183             if not 0 <= minutes < MINUTES_PER_HOUR:
184                 raise ValueError('Minutes is out of range [0,59].')
185             self._write(self._REG_MINUTES, int_to_bcd(minutes))
186
187         if hours is not None:
188             if not 0 <= hours < HOURS_PER_DAY:
189                 raise ValueError('Hours is out of range [0,23].')
190             self._write(self._REG_HOURS, int_to_bcd(hours) ) # not  | 0x40 according to
                datasheet
191
192         if year is not None:
193             if not 0 <= year < YEARS_PER_CENTURY:
194                 raise ValueError('Years is out of range [0,99].')
195             self._write(self._REG_YEAR, int_to_bcd(year))
196
197         if month is not None:
198             if not 1 <= month <= MONTHS_PER_YEAR:
199                 raise ValueError('Month is out of range [1,12].')
200             self._write(self._REG_MONTH, int_to_bcd(month))
201
202         if date is not None:
203             # How about a more sophisticated check?
204             if not 1 <= date <= MAX_DAYS_PER_MONTH:
205                 raise ValueError('Date is out of range [1,31].')
206             self._write(self._REG_DATE, int_to_bcd(date))
207
208         if day is not None:
209             if not 1 <= day <= DAYS_PER_WEEK:
210                 raise ValueError('Day is out of range [1,7].')
211             self._write(self._REG_DAY, int_to_bcd(day))
212
213     def write_datetime(self, dt):
214         """Write from a datetime.datetime object.
215         """
216         self.write_all(dt.second, dt.minute, dt.hour,
217                 dt.isoweekday(), dt.day, dt.month, dt.year % 100)
218
219     def write_now(self):
220         """Equal to DS3231.write_datetime(datetime.datetime.now()).
221         """
222         self.write_datetime(datetime.now())
223
224     def getTemp(self):
```

```
225             byte_tmsb = self._bus.read_byte_data(self._addr,0x11)
226             byte_tlsb = bin(self._bus.read_byte_data(self._addr,0x12))[2:].zfill(8)
227             return byte_tmsb+int(byte_tlsb[0])*2**(-1)+int(byte_tlsb[1])*2**(-2)
228
229         ############################
230         # AT24C32 Code
231         # datasheet: atmel.com/Images/doc0336.pdf
232         ############################
233
234         def set_current_AT24C32_address(self,address):
235             a1, a0 = divmod(address, 1<<8)
236             self._bus.write_i2c_block_data(self._at24c32_addr,a1,[a0])
237
238         def read_AT24C32_byte(self, address):
239             if False:
240                 print(
241                     "i2c_address =0x%x eepromaddress = 0x%x  " %
242                     (self._at24c32_addr, address))
243
244             self.set_current_AT24C32_address(address)
245             return self._bus.read_byte(self._at24c32_addr)
246
247         def write_AT24C32_byte(self, address, value):
248             if False:
249                 print(
250                     "i2c_address =0x%x eepromaddress = 0x%x value = 0x%x %i " %
251                     (self._at24c32_addr, address, value, value))
252             a1, a0 = divmod(address, 1<<8)
253             self._bus.write_i2c_block_data(self._at24c32_addr,a1,[a0, value])
254             time.sleep(0.20)
255
```