```
# SignalCore RF Source DLL driver
#

import sys
import time
import ctypes
import types
import numpy as np
import logging

SUCCESS = 0
NO_DEVICE = 0


# 64 bit API
LB_DLL = 'C:\\some_install_dir\\SignalCore\\x64\\sc5511a.dll'
try:
    lb_dll = ctypes.windll.LoadLibrary(LB_DLL)
    #lb_dll = ctypes.cdll.LoadLibrary(LB_DLL)

except Exception, e:
    s = 'Unable to load SignalCore DLL, please put sc5511a.dll in instrumentserver directory ' + str(e)
    raise ValueError(s)

class device_rf_params_t(ctypes.Structure):
    _fields_ =  [('rf1_freq', ctypes.c_ulonglong),
                 ('start_freq', ctypes.c_ulonglong),
                 ('stop_freq', ctypes.c_ulonglong),
                 ('step_freq', ctypes.c_ulonglong),
                 ('sweep_dwell_time', ctypes.c_uint),
                 ('sweep_cycles', ctypes.c_uint),
                 ('buffer_points', ctypes.c_uint),
                 ('rf_level', ctypes.c_float),
                 ('rf2_freq', ctypes.c_ushort)]

    def print_params(self):
        print('\ndevice rf params:\n'
              + 'rf1_freq = ' + str(self.rf1_freq) + '\n'
              + 'start_freq = ' + str(self.start_freq) + '\n'
              + 'stop_freq = ' + str(self.stop_freq) + '\n'
              + 'step_freq = ' + str(self.step_freq) + '\n'
              + 'sweep_dwell_time = ' + str(self.sweep_dwell_time) + '\n'
              + 'sweep_cycles = ' + str(self.sweep_cycles) + '\n'
              + 'buffer_points = ' + str(self.buffer_points) + '\n'
              + 'rf_level = ' + str(self.rf_level) + '\n'
              + 'rf2_freq = ' + str(self.rf2_freq))
```

```
class list_mode_t(ctypes.Structure):
    _fields_ =  [('sss_mode', ctypes.c_ubyte),
                 ('sweep_dir', ctypes.c_ubyte),
                 ('tri_waveform', ctypes.c_ubyte),
                 ('hw_trigger', ctypes.c_ubyte),
                 ('step_on_hw_trig', ctypes.c_ubyte),
                 ('return_to_start', ctypes.c_ubyte),
                 ('trig_out_enable', ctypes.c_ubyte),
                 ('trig_out_on_cycle', ctypes.c_ubyte)]

    def print_params(self):
        print('\nlist mode params:\n'
              + 'sss_mode = ' + str(self.sss_mode) + '\n'
              + 'sweep_dir = ' + str(self.sweep_dir) + '\n'
              + 'tri_waveform = ' + str(self.tri_waveform) + '\n'
              + 'hw_trigger = ' + str(self.hw_trigger) + '\n'
              + 'step_on_hw_trig = ' + str(self.step_on_hw_trig) + '\n'
              + 'return_to_start = ' + str(self.return_to_start) + '\n'
              + 'trig_out_enable = ' + str(self.trig_out_enable) + '\n'
              + 'trig_out_on_cycle = ' + str(self.trig_out_on_cycle))

class pll_status_t(ctypes.Structure):
    _fields_ =  [('sum_pll_ld', ctypes.c_ubyte),
                 ('crs_pll_ld', ctypes.c_ubyte),
                 ('fine_pll_ld', ctypes.c_ubyte),
                 ('crs_ref_pll_ld', ctypes.c_ubyte),
                 ('crs_aux_pll_ld', ctypes.c_ubyte),
                 ('ref_100_pll_ld', ctypes.c_ubyte),
                 ('ref_10_pll_ld', ctypes.c_ubyte),
                 ('rf2_pll_ld', ctypes.c_ubyte)]

    def print_params(self):
        print('\npll status params:\n'
              + 'sum_pll_ld = ' + str(self.sum_pll_ld) + '\n'
              + 'crs_pll_ld = ' + str(self.crs_pll_ld) + '\n'
              + 'fine_pll_ld = ' + str(self.fine_pll_ld) + '\n'
              + 'crs_ref_pll_ld = ' + str(self.crs_ref_pll_ld) + '\n'
              + 'crs_aux_pll_ld = ' + str(self.crs_aux_pll_ld) + '\n'
              + 'ref_100_pll_ld = ' + str(self.ref_100_pll_ld) + '\n'
              + 'ref_10_pll_ld = ' + str(self.ref_10_pll_ld) + '\n'
              + 'rf2_pll_ld = ' + str(self.rf2_pll_ld))


class operate_status_t(ctypes.Structure):
    _fields_ =  [('rf1_lock_mode', ctypes.c_ubyte),
                 ('rf1_loop_gain', ctypes.c_ubyte),
                 ('device_access', ctypes.c_ubyte),
                 ('rf2_standby', ctypes.c_ubyte),
                 ('rf1_standby', ctypes.c_ubyte),
                 ('auto_pwr_disable', ctypes.c_ubyte),
                 ('alc_mode', ctypes.c_ubyte),
                 ('rf1_out_enable', ctypes.c_ubyte),
                 ('ext_ref_lock_enable', ctypes.c_ubyte),
                 ('ext_ref_detect', ctypes.c_ubyte),
                 ('ref_out_select', ctypes.c_ubyte),
                 ('list_mode_running', ctypes.c_ubyte),
                 ('rf1_mode', ctypes.c_ubyte),
                 ('over_temp', ctypes.c_ubyte),
                 ('harmonic_ss', ctypes.c_ubyte)]

    def print_params(self):
        print('\noperate mode status: \n'
              + 'rf1_lock_mode = ' + str(self.rf1_lock_mode) + '\n'
              + 'rf1_loop_gain = ' + str(self.rf1_loop_gain) + '\n'
              + 'device_access = ' + str(self.device_access) + '\n'
              + 'rf2_standby = ' + str(self.rf2_standby) + '\n'
              + 'rf1_standby = ' + str(self.rf1_standby) + '\n'
              + 'auto_pwr_disable = ' + str(self.auto_pwr_disable) + '\n'
              + 'alc_mode = ' + str(self.alc_mode) + '\n'
              + 'rf1_out_enable = ' + str(self.rf1_out_enable) + '\n'
              + 'ext_ref_lock_enable = ' + str(self.ext_ref_lock_enable) + '\n'
              + 'ext_ref_detect = ' + str(self.ext_ref_detect) + '\n'
              + 'ref_out_select = ' + str(self.ref_out_select) + '\n'
              + 'list_mode_running = ' + str(self.list_mode_running) + '\n'
              + 'rf1_mode = ' + str(self.rf1_mode) + '\n'
              + 'over_temp = ' + str(self.over_temp) + '\n'
              + 'harmonic_ss = ' + str(self.harmonic_ss))


class device_status_t(ctypes.Structure):
    _fields_ =  [('list_mode_t', list_mode_t),
                 ('operate_status_t', operate_status_t),
                 ('pll_status_t', pll_status_t)]

    def print_params(self):
        self.list_mode_t.print_params()
        self.operate_status_t.print_params()
        self.pll_status_t.print_params()



class date(ctypes.Structure):
    _fields_ =  [('year', ctypes.c_ubyte),
                 ('month', ctypes.c_ubyte),
                 ('day', ctypes.c_ubyte),
                 ('hour', ctypes.c_ubyte)]

class device_info_t(ctypes.Structure):
    _fields_ =  [('product_serial_number', ctypes.c_ubyte),
              ('hardware_revision', ctypes.c_float),
              ('firmware_revision', ctypes.c_float),
              ('man_date', date)]

NUM_MAX_DEVICES = 5
ID_BUFFER_SIZE = 8
```

Next, we initialize the device, get the pointers and device handle and things like that correct:

```
 class SC5511A(Instrument):

    def __init__(self, name, devid=None, serial=None):
        super(SC5511A, self).__init__(name)

        if devid is None:
            raise Exception('SignalCore driver needs devid or serial as parameter')

        string_buffers = [ctypes.create_string_buffer(ID_BUFFER_SIZE) for i in range(NUM_MAX_DEVICES)]
        pointers = (ctypes.c_char_p*NUM_MAX_DEVICES)(*map(ctypes.addressof, string_buffers))
        results = [s.value for s in string_buffers]
        self.dev_num = ctypes.c_char_p(devid)
        lb_dll.sc5511a_open_device.restype = ctypes.POINTER(ctypes.c_int)
        self._handle = lb_dll.sc5511a_open_device(self.dev_num)

        device_rf_params = device_rf_params_t()
        device_status = device_status_t()

        device_status_temp = ctypes.pointer(device_status)
        lb_dll.sc5511a_get_device_status(self._handle, device_status_temp.contents)
        lb_dll.sc5511a_get_rf_parameters(self._handle, device_rf_params)


        lb_dll.sc5511a_set_rf_mode(self._handle, 0)
        lb_dll.sc5511a_set_output(self._handle, 1)
```

Then finally we use the functions defined in the API to make our own "get" and "set" functions to control the device settings or read them out.

```
     def do_get_frequency(self):
        device_rf_params = device_rf_params_t()
        lb_dll.sc5511a_get_rf_parameters(self._handle, device_rf_params)
        return float(device_rf_params.rf1_freq)

    def do_set_frequency(self, freq_Hz):
```

```
    return lb_dll.sc5511a_set_freq(self._handle, ctypes.c_ulonglong(int(freq_Hz)))

def do_get_power(self):
    device_rf_params = device_rf_params_t()
    lb_dll.sc5511a_get_rf_parameters(self._handle, device_rf_params)
    return device_rf_params.rf_level

def do_set_power(self, power):
    return lb_dll.sc5511a_set_level(self._handle, ctypes.c_float(power))

def do_get_rf_on(self):
    device_status = device_status_t()
    lb_dll.sc5511a_get_device_status(self._handle, device_status)
    return device_status.operate_status_t.rf1_out_enable == 1

def do_set_rf_on(self, val):
    if(val):
        return lb_dll.sc5511a_set_output(self._handle, 1)
    else:
        return lb_dll.sc5511a_set_output(self._handle, 0)
```