A Python Program for the PicoPak and PicoPlus Clock Measurement Modules

W.J. Riley
Hamilton Technical Services
Beaufort, SC 29907 USA
bill@wriley.com

• Introduction

This paper describes a Python terminal program to run a PicoPak or PicoPlus clock measurement module. The program can be used under Windows, Linux or Raspbian, and is especially useful with the latter and a Raspberry Pi computer. The program can be used with or without a PicoPak database, and can easily be adapted to a specific requirement or used as the basis of a GUI application. It uses a number of settings and flags to select the desired USB port, signal frequency, data format and storage media and other options, including use with a frequency error multiplier.

• Hardware Setup

The PicoPak or PicoPlus is connected to a Windows, Linux or Raspberry Pi computer via its rear USB cable which supplies both DC power and bidirectional communications, and +7 dBm nominal 10 MHz reference and 5 to 15 MHz signal inputs (10 MHz only for PicoPlus) are connected to the front SMA connectors. Either a monitor, keyboard and mouse are used or it can be operated "headless" via a network SSH connection. In that latter way a Raspberry Pi can serve as an inexpensive dedicated computer for the measuring system. A program like this for the RPi is particularly useful since Wine is not available for its ARM processor to run the Windows PicoPak user interface program.

• Program Description

The program is quite simple and readable thanks to the Python language and its pySerial library that supports serial port communications. It sets up the serial port, verifies that the PicoPak/Plus is connected by showing its S/N, checks that the input signals are present and that the unit is locked. Then the data stream is then activated and the measurements begin. They are scaled and displayed as phase readings in seconds, which, because of the exact 1-second measurement interval, are also fractional frequency values. Those readings are stored in a Stable32-compatible data file and/or PicoPak PostgreSQL database along with MJD timetags. The measurements continue indefinitely until the program is stopped with a Ctrl-C.

• Detailed Raspberry Pi Setup

A PicoPak or PicoPlus module combined with a Raspberry Pi computer forms a small and low cost but relatively high performance clock measurement system. The Python program described herein is particularly useful for operating a PicoPak/Plus with an RPi as shown in the photograph at the right, and the following are more detailed instructions for doing so,. They can be operated remotely via a WiFi connection using SSH and no additional hardware is needed except for the RPi power supply and a short USB cable.

1. Download and install the PySerial Python module needed for USB port access with this command:

   ```
   python -m pip install pyserial
   ```

2. Download and install the psycopg2 Python module needed for PostgreSQL database access (or eliminate the associated code from the picopak6.py program) with this command:

   ```
   python -m pip install psycopg2
   ```

3. Install the SSH service on the RPi with this command:

   ```
   sudo apt-get install ssh
   ```

4. Start the SSH daemon with this command:

   ```
   sudo /etc/init.d/ssh start
   ```

   and use `raspi-config` to activate SSH automatically whenever the RPi boots up:

5. Download and install the putty program on the Windows or Linux workstation computer that will be used to remotely control the PicoPak/Plus module from the following site:

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

6. Connect the PicoPak/Plus module to the RPi and to suitable reference and signal RF sources.

7. Set up the PicoPak Python program (picopak.py herein) on the RPi computer. Verify that the picopak.py program runs in response to the command python3 picopak6.py from the directory where it is installed. It may be necessary to make PYTHON PATH or other system changes depending on where pyserial and psycopg2 were installed.

8. Get the IP address of the RPi computer using the `ifconfig` or `hostname -I` command. It may be connected to the local network by either Ethernet or WiFi.

9. Connect to the RPi from the workstation using the putty application by entering the IP address of the RPi, selecting the SSH protocol, and entering the RPi login credentials (typically user pi and password raspberry).

10. Connect the desired reference and signal sources to the PicoPak/Plus module.

11. Edit or enter the desired picopak6.py program settings (e.g., to set source IDs and data filename or database usage).

12. Launch the PicoPak program remotely via SSH from the workstation and perform the desired measurement.

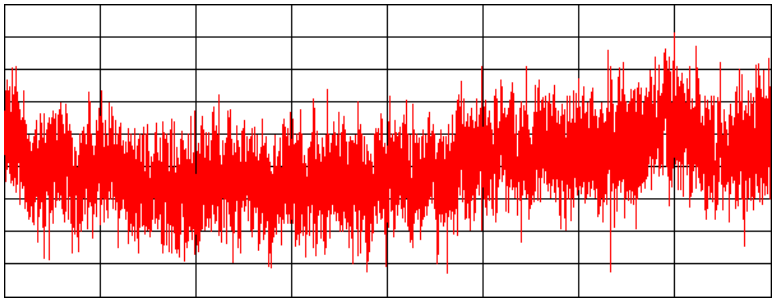
13. The PicoPak/Plus data can be accessed via the SSH connection and/or the PicoPak database. WinSCP is a good application for remote editing and downloading data from the RPi to a Windows computer.

14. The tmux program can be used to allow the measurements to continue after the SSH connection is closed.

15. It is recommended that a PicoPak PostgreSQL database be used instead of a data file on the Raspberry Pi to reduce wear on its memory card, or that the data be stored in a tmpfs RAM-based filesystem, e.g., /dev/shm/picopak.dat. That shared memory RAM filesystem should already exist.

- **Example**

An example of 1 second coherent phase data from a PicoPlus clock measurement module is shown in the figure at the right. The vertical scale is 1 ps/div, the horizontal scale is about 15 min/div, the 1s ADEV is about 1.3×10^{-12} and the frequency offset is about 2×10^{-16}.



- **Program Listing**

The following pages are a listing of the `picopak.py` PicoPak/Plus Python program. Other versions of this program with user inputs coded in sub-functions, both without and with GUI interfaces, are available upon request. The plain text terminal version shown here is best for remote operation. The various flags and settings are used to configure it as desired, and they can be edited remotely from a Windows computer with WinSCP. Once set up, the only entries that are typically required are the source IDs and the measurement description.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Python 3 program for communicating with a PicoPak clock measurement module.

Program flow: get serial port #, open port and database connection,
get nominal frequency, calc DDS word, open data file, reset PicoPak,
load DDS word, get S/N, check inputs, check lock, calc scale factors,
put run into database, start data stream, capture, scale, correct and
print readings, store timetagged data to file and database until
measurements are interrupted.  Then close the file, serial port,
and database connection, and exit the program.

Support is provided for automatic lock acquisition and for use with a
higher-resolution PicoPlus module having a frequency error multiplier.

The program can be stopped by unplugging the PicoPak reference input
or USB cable, or by a Ctrl-C keyboard interrupt.

It is quite remarkable how simple and readable this Python code is compared
with equivalent C code for running a PicoPak clock measurement module.

Note: Use python -m serial.tools.list_ports -v to list available ports
and ID PicoPak/Plus Clock Measurement Modules.
Use python -m serial.tools.miniterm /dev/ttyUSB# 115200
(where # is USB port #) to open a mini terminal to exercise PicoPak commands.

Changes after 1st version:
01/23/18: Add try/except for port opening.
          Use sys.exit() if exception
          Add test for invalid frequency.
          Add try/except for data file opening.
          Add try/except for data communications.
          Add pySerial copyright notice
          Add serial tools information
01/24/18  Create picppak2.py
          Install psycopg2
          Add database code
01/25/18  Add signal and reference clock ID macros
01/25/18  Change to picopak3.py
          Add defaults to user inputs
          Add data file, database and entry flags
          Add database parameter entry
01/26/18  Add None filename option
          Add parameter printout option
          Add show sources option and printout
          Add auto acquisition option (not yet implemented)
          Add Windows/Linux option flag
01/27/18  Use signed hex to integer conversion
          Add FF and CC processing
          Add warning if no data storage
01/28/18  Add integrated phase
          Add choice and entry of screen outputs
          Add provisions for PicoPlus with FEM
01/30/18  Add checking for frequency measurement outliers
          Complete code for automatic lock acquisition
          Complete code for PicoPlus with FEM
02/01/18  Done
```

```
Created: Sat Jan 20 11:20 2018
Revised: Thu Feb 01 08:59 2018
Author: bill@wriley.com
URL: www.wriley.com

Copyrights and Licenses:
This program uses pySerial for its serial communications functions.
pySerial is (c) 2001-2017 Chris Liechti <cliechti@gmx.net>.
See: http://pyserial.readthedocs.io/en/latest/appendix.html#license

picopak.py is (c) 2018 W. J. Riley Hamilton Technical Services
Beaufort, SC 29907 USA. All Rights Reserved under MIT License
that allows unrestricted free use by anyone for any purpose
but without any warranty of any kind.
"""

# Import the pySerial module
import sys
import time
import serial
import psycopg2
import numpy as np
import matplotlib.pyplot as plt
import math

# Defaults
# Operating System
WIN = True
LINUX = False
# Serial Port #
PORT = 4
# Nominal Frequency, MHz
# Without FEM, signal frequency can be between 5 and 15 MHz (10 MHz default)
# With FEM, signal frequency always 10 MHz, FEM O/P 10.25 MHz into PicoPak
FREQ = 10.00 # Changed to 10.25 below if USE_FEM is set
# Data filename
FILENAME = "picopak.dat"
# PicoPak database credentials
HOST = "192.168.2.40"
DBNAME = "ppd"
USER = "postgres"
PASSWORD = "root"
# Source ID #s
SIG_ID = 0
REF_ID = 0
# Close tolerance for setting nominal frequency to 10 MHz
TOL = 1e-9

# Screen Output Selection
# 1=None, 2=Raw Data, 3=Phase Data, 4=Integrated Phase, 5=Frequency, 6=MJD
SCREEN = 3

# Flags
# No entries are required if GET flags are set False
# Minimal information is shown if all SHOW flags are set False
# Data is always shown on the screen
# Note: If USE_DB and USE_FILE both False, no data will be captured
USE_DB = True # Flag to use database
USE_FILE = True # Flag to save data to file
```

```
USE_FEM = False # Flag for frequency error multiplier (PicoPlus)
USE_CLOSE = True # Flag to use 10 MHz as nominal if estinated freq close
# Must have valid port #
GET_PORT = False # Flag to enter serial port #
# Must have valid nominal frequency or do automatic acquisition
GET_FREQ = False # Flag to enter signal frequency - N/A if FEM
# Database use is optional
# GET_DB S/B False if USE_DB is False
GET_DB = False # Flag to enter database parameters
# Previous filename is overwritten
GET_FILENAME = False # Flag to enter data filename
# Must manually enter a new source usinbg psql, etc.
GET_SOURCES = False # Flag to enter source ID #s
GET_SCREEN = False # Flag to enter screen display choice
# Parameter printout is mainly for testing
SHOW_PARAMS = False # Flag to show program parameters
# SHOW_SOURCES requires that USE_DB be True
SHOW_SOURCES = False # Flag to show source names
DO_AUTO = False # # Flag to do automatic acquisition
DO_PLOT = True # Flag to plot data when program closes
STORE_MJD = True # Flag to store MJD in data file

# Welcome message
if(USE_FEM):
    print("PicoPlus Terminal Program")# pass
    FREQ = 10.25
else:
    print("PicoPak Terminal Program")

# Print all program parameters
if(SHOW_PARAMS):
    print("Flags:")
    print("    WIN = " + str(WIN))
    print("    LINUX = " + str(LINUX))
    print("    USE_DB = " + str(USE_DB))
    print("    USE_FILE = " + str(USE_FILE))
    print("    USE_CLOSE = " + str(USE_CLOSE))
    print("    USE_FEM = " + str(USE_FEM))
    print("    GET_PORT = " + str(GET_PORT))
    print("    GET_FREQ = " + str(GET_FREQ))
    print("    GET_DB = " + str(GET_DB))
    print("    GET_FILENAME = " + str(GET_FILENAME))
    print("    GET_SOURCES = " + str(GET_SOURCES))
    print("    GET_SCREEN = " + str(GET_SCREEN))
    print("    SHOW_PARAMS = " + str(SHOW_PARAMS))
    print("    SHOW_SOURCES = " + str(SHOW_SOURCES))
    print("    STORE_MJD = " + str(STORE_MJD))
    print("    DO_AUTO = " + str(DO_AUTO))
    print("    DO_PLOT = " + str(DO_PLOT))
    if(LINUX): # For Linux and Raspbian
        print("USB Port:")
    if(WIN): # For Windows
        print("COM Port:")
    print("    PORT = " + str(PORT))
    print("Database:")
    print("    HOST = " + str(HOST))
    print("    DBNAME = " + str(DBNAME))
    print("    USER = " + str(USER))
    print("    PASSWORD = " + str(PASSWORD))
```

```
    print("Data File:")
    print("    FILENAME = " + str(FILENAME))
    print("Sources:")
    print("    SIG_ID = " + str(SIG_ID))
    print("    REF_ID = " + str(REF_ID))
    print(" Nominal Frequency, MHz:")
    print("    FREQ = " + str(FREQ))
    print("    TOL = " + str(TOL))
    print("Screen Format:")
    print("    SCREEN = " + str(SCREEN))

# Get USB or COM port #
usb = PORT
# Set default port
if(WIN):
    usb = 'COM' + str(usb)
if (LINUX):
    usb = '/dev/ttyUSB' + str(usb)

# Get port entry
if(GET_PORT):
    if(LINUX):
        usb = input("Enter USB Port # [" + str(PORT) + "]: ")
        usb = usb or PORT
        usb = '/dev/ttyUSB' + str(usb)

    if(WIN):
        usb = input("Enter COM Port # [" + str(PORT) + "]: ")
        usb = usb or PORT
        usb = 'COM' + str(usb)

# Open the serial port
# Timeout of 2 seconds allows time for 1 second measurement interval
# ser = serial.Serial('/dev/ttyUSB2', 115200, timeout=2) # open serial port
try:
    ser = serial.Serial(usb, 115200, timeout=2) # open serial port
except:
    print("Unable to open port " + str(usb))
    print("Program closed")
    sys.exit("Invalid port #")

# Enter data filename
filename = FILENAME
if(GET_FILENAME):
    filename = input("Enter Data Filename [" + str(FILENAME) + "]: ")
    filename = filename or FILENAME
    if(filename == "None"):
        USE_FILE = False

# Enter database parameters
db_host = HOST
db_name = DBNAME
db_user = USER
db_pw = PASSWORD
if(GET_DB):
    db_host = input("Enter Database Host [" + str(HOST) + "]: ")
    db_host = db_host or HOST
    db_name = input("Enter Database Name [" + str(DBNAME) + "]: ")
    db_name = db_name or DBNAME
```

```
    db_user = input("Enter Database User [" + str(USER) + "]: ")
    db_user = db_user or USER
    db_pw = input("Enter Database Password [" + str(PASSWORD) + "]: ")
    db_pw = db_pw or PASSWORD

# Enter source ID #s
# See clock_names ppd database table for list of source IDs
sig_id = SIG_ID
ref_id = REF_ID
if(GET_SOURCES):
    sig_id = input("Enter Signal ID # [" + str(SIG_ID) + "]: ")
    sig_id = sig_id or SIG_ID
    ref_id = input("Enter Reference ID # [" + str(REF_ID) + "]: ")
    ref_id = ref_id or REF_ID

# Enter screen display choice
# 1=None, 2=Raw Data, 3=Phase Data, 4=Integrated Phase, 5=Frequency, 6=MJD
if(GET_SCREEN):
    screen = input("Enter Screen Display Choice # [" + str(SCREEN) + "]: ")
    SCREEN = screen or SCREEN

# Open the PicoPak database
# The database access credentials must be set above
if(USE_DB):
    try:
        conn = psycopg2.connect(host=db_host, dbname=db_name, \
        user=db_user, password=db_pw)
    except:
        USE_DB = False
        print("Unable to connect to database " + DBNAME + " at " + HOST)
        print("Program closed")
        sys.exit("No database connection")

    # Create database cursor
    cur = conn.cursor()

# Get nominal signal frequency
# Entry of zero frequency will enable automatic acquisition
# and use that absolute value as the nominal
# unless it is near 10 MHz and USE_CLOSE set
# If  Picoplus with FEM is used, it must be 10.25 MHz, no entry allowed
mhz = FREQ
if(GET_FREQ and not USE_FEM):
    mhz = input("Enter Signal Freq, MHz [" + str(FREQ) + \
    " (Auto=0)]: " )
    mhz = mhz or 10
    mhz = abs(float(mhz))
    if(mhz==0):
        DO_AUTO = True

# Test for valid frequency
if(not DO_AUTO):
    if((mhz < 5) or (mhz > 15)):
        print("Frequency must be between 5 and 15 MHz")
        print("Program closed")
        if(USE_DB):
            cur.close() # Close database cursor
            conn.close() # Close database connection
        sys.exit("Invalid frequency")
```

```
    # Calculate DDS word (if no auto acquisition)
    # DDS word = (MHz/120)*2^32 rounded to nearest int
    word = int(round((mhz/120.0) * 2**32, 0))
    # print(word) # For testing

    # Increment the DDS word for testing
    # With coherent 10 MHz inputs, this will change
    # PicoPak CCCC data stream from positive to negative
    # nominal 1 second phase increments to verify that
    # their processing is OK and we get nominally zero
    # frequency offset with both 15555555 and 155555556
    # word += 1

# Open data file for writing
if(USE_FILE):
    try:
        file = open(filename, 'w')
    except:
        print("Unable to open data file")
        print("Program closed")
        ser.close() # Close serial port
        if(USE_DB):
            cur.close() # Close database cursor
            conn.close() # Close database connection
        sys.exit("Invalid data file")

# Reset the PicoPak
ser.write(b'R') # write an R (reset)
# Swallow reset response
ser.readlines()

# Load the DDS word if not auto acquisition
# Send DDS word as 8 hex chars with F= command
if(not DO_AUTO):
    hexword = hex(word).upper()
    # print(hexword) # For testing
    hexword = str(hexword)
    hexword = hexword[2:10]
    print("DDS Word = " + str(hexword))  # Can omit
    cmd = 'F=' + hexword
    cmd = bytes(cmd, 'utf-8')
    ser.write(cmd)
    time.sleep(1.0) # Allow time for module to lock

# Do automatic lock acquisition
# Make low-resolution counter frequency measurement
# Set up a 100 Hz beat note
# Make higher-resolution period measurement
# Set DDS accordingly
if(DO_AUTO and not USE_FEM):
    print("Performing automatic lock acquisition")
    # Doing this preliminary coarse freq meas doesn't seem necessary
    if(False): # Flag to do preliminary coarse frequency measurement
        # Send H? command to PicoPak
        ser.write(b'H?') # write an H? (meas freq)
        # Wait for the 1s measurement
        time.sleep(1.1)
        # Read approximate signal frequency (8 hex bytes plus \r\n)
```

```
    line = ser.readline()
    # print("1st meas=" + str(line)) # For testing
# Do coarse frequency measurement
ser.write(b'H?') # write an H? (meas freq)
time.sleep(1.1)
line = ser.readline()
# print("2nd meas=" + str(line)) # For testing
fmeas = str(line)
fmeas = fmeas[2:10]
fmeas = int(fmeas, 16) # Approx sig freq, Hz
print("Approximate Signal Frequency = " + str(fmeas) + " Hz") # Can omit
# Add 100 Hz for beat note
fmeas += 100
# Calc DDS word
word = int(round((fmeas/120e6) * 2**32, 0))
# Send offset DDS word to PicoPak
hexword = hex(word)
hexword = str(hexword)
hexword = hexword[2:10]
cmd = 'F=' + hexword
cmd = bytes(cmd, 'utf-8')
ser.write(cmd)
# Measure beat period 10 times
fbeat=np.zeros(10)
for i in np.arange(10):
    # Send B? to measure beat period
    ser.write(b'B?') # write an B? (meas beat period)
    # Wait for measurement
    time.sleep(0.1)
    # Read beatnote period (4 hex chars plus \r\n)
    line = ser.readline()
    period = str(line)
    period = '0x' + period[2:6]
    period = int(period, 16) # Beat period in units of 200 ns
    # Calculate beat frequency
    fbeat[i] =  1.0 / (200e-9 * period)
# Calculate accurate signal frequency
# Find median, check for and reject > 1% outliers, then find average
fmed = np.median(fbeat)
for i in np.arange(10):
    if( fbeat[i]<0.99*fmed or fbeat[i]>1.01*fmed):
        fbeat[i] = fmed # Replace outlier with median
favg = np.average(fbeat)
# print("Estimated Beat Frequency = " + str(favg) + " Hz") # For testing
# Calculate estimated signal frequency in Hz
fest = fmeas - favg
print("Estimated Signal Frequency = " + str(fest) + " Hz") # Can omit
# Set nominal frequency in MHz to estimate
# or optionally to 10 MHz if close
mhz = fest / 1e6
# Use exact 10 MHz as the nominal frequemcy if the estimated signal
# frequency is close to it
# An option would be to explicitly enter nominal frequency
# The default relative tolerance is 1e-9
# which can be changed with the 3rd argument (TOL)
# The 4th argument absolute tolerance should remain its default = 0
if(USE_CLOSE): # Flag to using 10 MHz if estimate is close
    if(math.isclose(mhz, 10.0, rel_tol=TOL)):
        mhz = 10.0
```

```
    print("Nominal Frequency = " + str(mhz) + " MHz") # Can omit
    # Calc DDS word
    word = int(round((fest/120e6) * 2**32, 0))
    # print("DDS Word = " + str(word)) # For testing
    # Set DDS to accurate frequency
    hexword = hex(word)
    hexword = str(hexword)
    hexword = hexword[2:10]
    print("DDS Word = " + str(hexword))
    cmd = 'F=' + hexword
    cmd = bytes(cmd, 'utf-8')
    ser.write(cmd)
    # Acquire lock (happens by itself, allow 1s, checked below)
    time.sleep(1.0) # Allow time for module to lock

# Get PicoPak S/N
# N entry needs ? for plain response or CR for verbose
ser.write(b'N?') # write an N? (module info)
# Swallow model #
ser.readline()
# Get S/N as 4 hex digits
line = ser.readline()
sn = str(line)
hexsn = sn[4:6]
# Windows and Ubuntu
sn = sn[2:6]
# Raspbian (?) Results have differed - Perhaps different Python version
sn = int(sn, 16)
if(USE_FEM):
    print("PicoPlus S/N = " + str(sn))
else:
    print("PicoPak S/N = " + str(sn))

# Swallow firmware version
ser.readline()

# Check PicoPak inputs
ser.write(b'E?') # write an E? (ref & sig check)
line = ser.readline()
if int(line) == 0:
    print("No Reference or Signal Inputs")
    print("Program Closed")
    ser.close() # Close serial port
    if(USE_FILE):
        file.close() # Close file
    if(USE_DB):
        cur.close() # Close database cursor
        conn.close() # Close database connection
    sys.exit("No Reference or Signal")
if int(line) == 1:
    print("No Signal Input")
    print("Program Closed")
    ser.close() # Close port
    if(USE_FILE):
        file.close() # Close file
    if(USE_DB):
        cur.close() # Close database cursor
        conn.close() # Close database connection
    sys.exit("No Signal")
```

```
if int(line) == 2:
    print("No Reference Input")
    print("Program Closed")
    ser.close() # Close port
    if(USE_FILE):
        file.close() # Close file
    if(USE_DB):
        cur.close() # Close database cursor
        conn.close() # Close database connection
    sys.exit("No Reference")
if int(line) == 3:
    print("Inputs OK")

# Show signal and reference clock names
if(SHOW_SOURCES and not USE_DB):
    print("Signal Clock ID = " + str(sig_id))
    print("Reference Clock ID = " + str(ref_id))

if(SHOW_SOURCES and USE_DB):
    if(int(sig_id)>0):
        try:
            cur.execute("SELECT description FROM clock_names \
            WHERE clock_id=%s", sig_id)
            res = cur.fetchone()
            print("Signal Clock: ID = " + str(sig_id) + ", " + res[0])
        except:
            print("Database  error #9 getting signal clock description")

    elif(int(sig_id)==0):
        print("Generic Signal Clock")

    if(int(ref_id)>0):
        try:
            cur.execute("SELECT description FROM clock_names \
            WHERE clock_id=%s", ref_id)
            res = cur.fetchone()
            print("Reference Clock: ID = " + str(ref_id) + ", " + res[0])
        except:
            print("Database error #10 getting reference clock description")

    elif(int(ref_id)==0):
        print("Generic Reference Clock")

# Check lock
ser.write(b'=?') # Write an =? (status check)
line = ser.readline()
# Response has 11 hex chars, lock bit is 5th char from left
lock = line[6:7]
if int(lock) == 1:
    if(USE_FEM):
        print("PicoPlus Locked")
    else:
        print("PicoPak Locked")
if int(lock) == 0:
    print("Unlocked")
    print("Program Closed")
    ser.close() # Close port
    if(USE_FILE):
        file.close() # Close file
```

```
    if(USE_DB):
        cur.close() # Close database cursor
        conn.close() # Clo se database connection
    if(USE_FEM):
        sys.exit("PicoPlus Unlocked")
    else:
        sys.exit("PicoPak Unlocked")

# For a 10 MHz signal frequency w/o FEM the closest 32-bit DDS tuning word
# is 15555555  hex or 357913941 and the corresponding DDS frequency
# is 120 MHz * DDS word / 2^32 =  9999999.990686774 Hz.  That
# corresponds to a fractional frequency offset of -9.313225746154785e-10.
# The PicoPak measurement interval is exactly 1 second as determined
# by its 10 MHz reference.  Thus the phase ramp that appears on its
# phase data is about -931.3 ps  or between 152 and 153 counts or
# 98 and 99 hex per each point, with the readings varying slightly around
# that nominal value because of noise.
# Witn FEM, the closest DDS word is 15DDDDDE
# Here, we use DDS word (word) to calc DDS freq (dds)
# and its fractional frequency offset from nominal frequency (mhs & hz)
# print("DDS word = " + str(word) + " = " + str(hexword) + " hex")
dds = 120e6 * word / 2**32
# print("DDS frequency = " + str(dds) + " Hz")
hz = mhz * 1e6
# print("Nominal Frequency = " + str(hz) + " Hz")
offset = (dds - hz) / hz
# print("DDS fractional frequency offset = " + str(offset))

# The scale factor for the phase data is equal to the
# resolution of the 14-bit PicoPak AD9951 DDS phase control word
# e.g., 100 ns / 2^14 = about 6.10 ps at 10 MHz
scale = 1.0 / (hz * 2**14)

# The scale factor is enhanced by x10.625 with the PicoPlus FEM
# to about 0.574 ps
if(USE_FEM):
    scale /= 10.625
    offset /= 10.625

# Put PicoPak module into database
# Entering a new module or updating an existing entry
# requires reading and displaying existing entries and accepting
# a new one - requires rather complex user interface without GUI
# Alternative is to print content of clock_names table
# manually using psql query: SELECT * from clock_names;
# and use as paper reference to clock_id numbers
# NOT IMPLEMENTED - MODULE MUST ALREADY BE IN measurement_modules TABLE

# Put measurement run info into database
# Get current MJD
if(USE_DB):
    now = time.time()
    mjd = 40587.0 + now/86400.0
    # Get next meas_id
    try:
        cur.execute("SELECT meas_id FROM measurement_list ORDER BY meas_id \
        DESC LIMIT 1")
    except:
        print("Database error #1 getting next meas_id")
```

```
    res = cur.fetchone()

    # Next meas_id is res[0] (an int) +1
    meas = res[0] + 1
    print("Measurement ID = " + str(meas))

    # Put run into measurements_list
    try:
        cur.execute("INSERT INTO measurement_list(meas_id, sn, sig_id, ref_id, \
        frequency, description, begin_mjd, tau) \
        VALUES(%s, %s, %s, %s, %s, %s, %s, %s)", (meas, sn, sig_id, ref_id, \
        hz, "PicoPak Measurement", mjd, 1))
    except:
        print("Database error #2 putting run into measurement_list")

    # Set measurement_module active
    try:
        cur.execute("UPDATE measurement_modules SET active=TRUE \
        WHERE sn=%s", (sn, ))
    except:
        print("Database error #3 setting measurement module active")

    # Do not commit these transactions here
    # Wait until data are stored to avoid an empty measurement entry

# Issue warning if data are not being stored
if(not USE_FILE and not USE_DB):
    print("Data not being stored")

# Start data stream #3
# Format is PPPPFFCC, where PPPP is phase increment, FF is frequency
# adjustment and CC is phase correction
# Phase increment includes DDS frequency offset
# No support for other data streams
ser.write(b'S=03') # Write an S=03 (stream #3)
print("Measurements Started")

# Initialize integrated phase to zero
phase=0.0

# Show screen output format
if(SCREEN==1): # None
    pass
elif(SCREEN==2): # Raw Data
    print("Raw data (hex)")
elif(SCREEN==3): # Phase
    print("Phase, seconds")
elif(SCREEN==4): # Integreated Phase
    print("Integrated Phase, seconds")
elif(SCREEN==5): # Frequency
    print("Frequency, Hz")
elif(SCREEN==6): # MJD
    print("MJD")

# Read data stream
# The 1st reading likely partial result - swallow it
ser.readline()
while(True):
    try:
```

```
    line = ser.readline() # read a '\n' terminated line
except:
    # Get this exception if USB cable disconnected
    print("No signal received")
    print("Program closed")
    ser.close() # Close port
    if(USE_FILE):
        file.close() # Close file
    if(USE_DB):
        # Put end MJD into measurement_list
        try:
            cur.execute("UPDATE measurement_list SET end_mjd=%s \
            WHERE meas_id=%s", (mjd, meas))
        except:
            print("Database error #5 putting end mjd into measurement_list")
        # Set measurement_module inactive
        try:
            cur.execute("UPDATE measurement_modules SET active=FALSE \
            WHERE sn=%s", (sn, ))
        except:
            print("Database error #6 setting measurement module inactive")
        # Commit final database entries
        conn.commit()
        # Close database
        cur.close() # Close database cursor
        conn.close() # Close database connection
    # Exit program
    sys.exit("Comunications lost")

# Check that data were received
# Expect 10 chars (8 data hex chars plus CR/LF)
if(len(line) < 10):
    break
reading = str(line)
# print("PicoPak/Plus data reading = ", reading) # For testing
# reading is of form b'PPPPFFCC\r\n'
# Windows and Ubuntu
value = '0x' + reading[2:6]
# Raspbian results have differed - Python 2 vs 3?
# Hex string to signed int conversion
point = int(value, 16)
if(point>0x7FFF): # Check if negative
    point -= 0x10000
# print("Data point = " + str(point)) # For testing
point *= scale
# print("Scaled data point = " + str(point)) # For testing
point += offset
# print("Offset-corrected data point = " + str(point)) # For testing

# Process CC phase correction
# We apply phase correction before adjusting DDS frequency
# Hex string to signed integer
corr = '0x' + reading[8:10]
corr = int(corr, 16)
if(corr>0x7F):
    corr -= 0x100

# Is a phase correction required?
# Phase correction may be needed because of a frequency adjustment
```

```
# If corr is between -54 and +54 (dec), it is a phase correction
# equal to corr * step * adj
# where step = (12e7 / (2^32 * hz) * 0.1
step = (12e7 / ((2**32) * hz)) * 0.1
# If corr is -64 or +64, it is a tracking warning,
# and the phase correction is -55 or +55
# If corr is +96, it is an alarm that tracking has been lost
# Detect tracking lost
if(corr==96):
    print("TRACKING LOST")
    # MAY WANT TO ABORT
    corr = 0
# Trap corr values <-54 or >54
if(corr==-64):
    print("TRACKING WARNING")
    corr = -55
if(corr==64):
    print("TRACKING WARNING")
    corr = 55
# Apply non-zero phase correction
if(corr != 0):
    # Apply phase correction
    # PicoPak frequency adjustment step size defaults to 1
    size = 1
    point += corr * step * size

# Get current MJD
now = time.time()
mjd = 40587.0 + now/86400.0

# point is the current phase value in seconds
# It is also the current fractional frequency
# Calculate current frequency in Hz
# Note that signal frequency not FEM O/P is reported with FEM
freq = FREQ * 1e6 * (1 + point)
if(USE_FEM):
    # Refer frequency value to 10 MHz
    freq = 10e6 * (1 + point)
# Integrate phase
phase += point

# Print selected output
if(SCREEN==1): # None
    pass
elif(SCREEN==2): # Raw Data
    print('0x' + reading[2:10])
elif(SCREEN==3): # Phase
    print(point)
elif(SCREEN==4): # Integreated Phase
    print(phase)
elif(SCREEN==5): # Frequency
    print(freq)
elif(SCREEN==6): # MJD
    print(mjd)

# Process FF frequency adjustment
# Hex string to signed integer
adj = '0x' + reading[6:8]
adj = int(adj, 16)
```

```
    if(adj>0x7F):
        adj -= 0x100

    # Apply FF frequency adjustment
    # Is frequency adjustment required?
    if(adj != 0):
        # Adjust DDS word
        word += adj
        # Send DDS word to PicoPak
        hexword = hex(word)
        hexword = str(hexword)
        hexword = hexword[2:10]
        # print("DDS Word = ", + str(hexword))  # For testing
        cmd = 'F=' + hexword
        cmd = bytes(cmd, 'utf-8')
        ser.write(cmd)
        # Recalculate DDS frequency and offset correction
        dds = 120e6 * word / 2**32
        hz = mhz * 1e6
        offset = (dds - hz) / hz
        # Update scale factor
        scale = 1.0 / (hz * 2**14)

    # Store timetagged data into file
    if(USE_FILE and STORE_MJD):
        out = str(mjd) + ' ' + str(phase) + '\n'
        file.write(out)
        file.flush()

    # Store plain data into file
    if(USE_FILE and not STORE_MJD):
        out = str(phase) + '\n'
        file.write(out)
        file.flush()

    # Store timetagged data into database
    if(USE_DB):
        try:
            cur.execute("INSERT INTO measurements(sn, mjd, meas) \
            VALUES(%s, %s, %s)", (sn, mjd, phase))
        except:
            print("Database error #4 entering data")
        # Commit database entry or entries
        conn.commit()
# End of measurement while loop

if(USE_FILE and DO_PLOT):
    # Get the phase data
    # This works in IPython window but not in ordinary terminal
    # Can monitor measurement via database with PicoMon program
    # or the PicoPak web monitor application
    # Can read data file into Stable32 "on-the fly"
    data = np.loadtxt(filename)
    # Plot the phase data (it is also the fractional frequency)
    if(STORE_MJD):
        plt.plot(data[:,1])
    else: # 1 column
        plt.plot(data)
    plt.xlabel("Data Point")
```

```
    plt.ylabel("Phase")
    plt.grid()

# Reference signal removed to stop program
print("Signal lost")
# Close the serial port
ser.close()
if(USE_FILE):
    # Close data file
    file.close()
if(USE_DB):
    # Put end MJD into measurement_list
    try:
        cur.execute("UPDATE measurement_list SET end_mjd=%s \
        WHERE meas_id=%s", (mjd, meas))
    except:
        print("Database error #7 putting end mjd into measurement_list")
    # Set measurement_module inactive
    try:
        cur.execute("UPDATE measurement_modules SET active=FALSE \
        WHERE sn=%s", (sn, ))
    except:
        print("Database error #8 setting measurement module inactive")
    # Commit final database entries
    conn.commit()
    # Close database cursor and connection
    cur.close()
    conn.close()

# Close program
print("Program Closed")
```