# factory\_test\_tool.py

import sys

import os

from factory\_test\_tool\_ui import \*

import time

import ConfigParser

import serial

import serial.tools.list\_ports

import threading

import datetime

import Queue

import win32api

import requests

import json

from memory\_profiler import profile

from my\_widget import \*

from io import StringIO

sys.path.append('../')

import RF\_test.esp\_test as esp\_test

sys.path.append('../')

import upload\_to\_server.upload\_to\_server as upload\_to\_server

from upload\_to\_server import \*

from PyQt4 import QtCore, QtGui

try:

\_fromUtf8 = QtCore.QString.fromUtf8

except AttributeError:

def \_fromUtf8(s):

return s

try:

\_encoding = QtGui.QApplication.UnicodeUTF8

def \_translate(context, text, disambig):

return QtGui.QApplication.translate(context, text, disambig, \_encoding)

except AttributeError:

def \_translate(context, text, disambig):

return QtGui.QApplication.translate(context, text, disambig)

class FactoryToolUI(Ui\_MainWindow, QtGui.QMainWindow):

\_SP\_SIGN = '$$'

SIGNAL\_PRINT = QtCore.pyqtSignal(QtGui.QTextBrowser, str)

DUT\_NUM = 4

DUT\_PORTS = []

DUT\_RATES = []

CHIP\_TYPE\_NUM = 0

class \_Print(StringIO):

'''

this is a inner class use for ui print to specify dut

'''

def \_\_init\_\_(self, factory\_tool, obj):

""" x.\_\_init\_\_(...) initializes x; see help(type(x)) for signature """

self.factory\_tool = factory\_tool

self.obj = obj

def write(self, log):

timestr=time.strftime('[%M:%S]',time.localtime(time.time()))

log=timestr+log

self.factory\_tool.SIGNAL\_PRINT.emit(self.obj, log)

def \_\_init\_\_(self, params={}, parent=None):

super(QtGui.QMainWindow, self).\_\_init\_\_(parent=parent)

self.setupUi(self) # general by pyqt designer

self.init\_ui() # add spicial control module and init some ui settings

self.init\_parameters() # initial the settings for production test

self.init\_signal() # add signal for need

tmp\_path = os.getcwd().replace('\\', '//')

self.logs\_path = tmp\_path[:tmp\_path.find(tmp\_path.split('//')[len(tmp\_path.split('//'))-1])] + 'logs//'

### init functions ###

def init\_ui(self):

self.twTestArea.setCurrentIndex(0)

for i in xrange(1,self.DUT\_NUM+1):

self.signal\_print\_log(eval('self.tbLog{}'.format(str(i))), '[state]idle')

for j in xrange(1,3):

self.DUT\_PORTS.append(eval('self.cbPort'+str(i)+'\_'+str(j)))

self.DUT\_RATES.append(eval('self.cbPortRate'+str(i)+'\_'+str(j)))

eval('self.lePortRate'+str(i)+'\_'+str(j)).setHidden(True)

def init\_signal(self):

QtCore.QObject.connect(self.trwTestFlow, QtCore.SIGNAL(\_fromUtf8("itemChanged(QTreeWidgetItem\*,int)")), self.\_signal\_testflow\_check)

QtCore.QObject.connect(self.pbTFSubmit, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_testflow\_submit)

QtCore.QObject.connect(self.pbTFReset, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_testflow\_reset)

QtCore.QObject.connect(self.pbCloudSync, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_cloud\_sync)

QtCore.QObject.connect(self.pbDutReset, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_dut\_reset)

QtCore.QObject.connect(self.pbDutSubmit, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_dut\_submit)

QtCore.QObject.connect(self.cbChipType, QtCore.SIGNAL(\_fromUtf8("currentIndexChanged(int)")), self.combobox\_chip\_type\_change)

QtCore.QObject.connect(self.cbTestFrom, QtCore.SIGNAL(\_fromUtf8("currentIndexChanged(int)")), self.combobox\_test\_from\_change)

QtCore.QObject.connect(self.pbBinPath, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_showFileDialog)

self.SIGNAL\_PRINT.connect(self.signal\_print\_log)

self.maCloud.changed.connect(self.button\_change\_position)

QtCore.QObject.connect(self.pbAllStart, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_all\_start)

QtCore.QObject.connect(self.pbAllStop, QtCore.SIGNAL(\_fromUtf8("clicked()")), self.button\_all\_stop)

self.pbStart1.clicked.connect(lambda :self.button\_single\_start(self.pbStart1))

self.pbStart2.clicked.connect(lambda :self.button\_single\_start(self.pbStart2))

self.pbStart3.clicked.connect(lambda :self.button\_single\_start(self.pbStart3))

self.pbStart4.clicked.connect(lambda :self.button\_single\_start(self.pbStart4))

self.pbStop1.clicked.connect(lambda :self.button\_single\_stop(self.pbStop1))

self.pbStop2.clicked.connect(lambda :self.button\_single\_stop(self.pbStop2))

self.pbStop3.clicked.connect(lambda :self.button\_single\_stop(self.pbStop3))

self.pbStop4.clicked.connect(lambda :self.button\_single\_stop(self.pbStop4))

for cb in self.DUT\_PORTS: cb.clicked.connect(self.combobox\_change\_port)

self.cbPortRate1\_1.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate1\_1))

self.cbPortRate1\_2.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate1\_2))

self.cbPortRate2\_1.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate2\_1))

self.cbPortRate2\_2.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate2\_2))

self.cbPortRate3\_1.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate3\_1))

self.cbPortRate3\_2.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate3\_2))

self.cbPortRate4\_1.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate4\_1))

self.cbPortRate4\_2.currentIndexChanged.connect(lambda :self.combobox\_change\_baud(self.cbPortRate4\_2))

self.maLog1.triggered.connect(lambda :self.button\_pop\_log(1))

self.maLog2.triggered.connect(lambda :self.button\_pop\_log(2))

self.maLog3.triggered.connect(lambda :self.button\_pop\_log(3))

self.maLog4.triggered.connect(lambda :self.button\_pop\_log(4))

QtCore.QMetaObject.connectSlotsByName(self)

def init\_parameters(self):

self.esp\_process={1:None, 2:None, 3:None, 4:None}

self.run\_queue=Queue.Queue(maxsize=4)

self.run\_flag = True

self.mutex = threading.Lock()

self.dut\_config = {}

self.CHIP\_TYPE\_NUM = self.cbChipType.count()

self.BAUD\_NUM = self.cbPortRate1\_1.count()

self.button\_dut\_reset(file\_path='./config/dutConfig', try\_login=True)

self.test\_flow = {}

self.init\_testflow()

self.init\_threshold()

self.init\_test\_thread()

def init\_test\_thread(self):

for i in range(1,5):

stdout\_ = self.\_Print(self, eval("self.tbLog"+str(i)))

self.dut\_config['common\_conf']['dut\_num'] = str(i)

self.esp\_process[id]=esp\_test.esp\_testThread(stdout\_, self.dut\_config,self.test\_flow)

def init\_threshold(self):

import openpyxl

wb = openpyxl.load\_workbook('config/ESP8266\_Threshold\_20180110\_hmj.xlsx')

sheet = wb.get\_sheet\_by\_name(wb.sheetnames[0])

for i in xrange(1, sheet.max\_row+1):

self.twThreshold.setRowCount(sheet.max\_row)

for j in xrange(1, sheet.max\_column+1):

twi = QtGui.QTableWidgetItem()

if sheet[str(chr(64+j))+str(i)].value != None:

twi.setText(str(sheet[str(chr(64+j))+str(i)].value))

self.twThreshold.setItem(i-1, j-1, twi)

def init\_testflow(self):

self.ui\_update\_testflow('./config/tmp\_testFlow')

if self.dut\_config['common\_conf']['position'] == 'cloud':

flow\_path = './config/cloudTestFlow'

else:

flow\_path = './config/testFlow'

try:

if self.button\_testflow\_reset(flow\_path) != 0:

print ('Test Flow file was broken, load with default config')

self.button\_testflow\_reset('./config/tmp\_testFlow')

except:

print ('Test Flow file was broken, load with default config')

self.button\_testflow\_reset('./config/tmp\_testFlow')

self.ui\_update\_testflow(flow\_path)

os.remove('./config/tmp\_testFlow')

### signal deal functions ###

def \_signal\_testflow\_check(self, item=None, index=None):

def updateParentItem(item):

try:

parent = item.parent()

except:

return

checkedCount = 0

checkableCount = 0

for i in xrange(parent.childCount()):

childItem = parent.child(i)

if childItem.flags()&QtCore.Qt.ItemIsUserCheckable:

checkableCount += 1

if childItem.checkState(0) == 2:

checkedCount += 1

if(checkedCount <= 0):

if parent.flags() & QtCore.Qt.ItemIsUserCheckable:

parent.setCheckState(0, QtCore.Qt.Unchecked)

elif checkedCount>0 and checkedCount<checkableCount:

if parent.flags() & QtCore.Qt.ItemIsUserCheckable:

parent.setCheckState(0, QtCore.Qt.PartiallyChecked)

elif checkedCount == checkableCount:

if parent.flags() & QtCore.Qt.ItemIsUserCheckable:

parent.setCheckState(0, QtCore.Qt.Checked)

if item.checkState(0) == QtCore.Qt.Unchecked:

checkalbeChildCount = 0

for i in xrange(item.childCount()):

if item.child(i).flags() & QtCore.Qt.ItemIsUserCheckable:

item.child(i).setCheckState(0,QtCore.Qt.Unchecked)

checkalbeChildCount += 1

if checkalbeChildCount <= 0:

updateParentItem(item)

elif item.checkState(0) == QtCore.Qt.Checked:

checkalbeChildCount = 0

for i in xrange(item.childCount()):

if item.child(i).flags() & QtCore.Qt.ItemIsUserCheckable:

item.child(i).setCheckState(0,QtCore.Qt.Checked)

checkalbeChildCount += 1

if checkalbeChildCount <= 0:

updateParentItem(item)

def signal\_print\_log(self, tb, log):

show\_flag = True

log=str(log)

state\_flag = True

dut\_num = str(tb.objectName()[-1])

if log.find('[state]') >= 0:

show\_flag = False

if log.lower().find('idle') >= 0:

state = 'IDLE'

eval("self.tbLog"+str(dut\_num)).clear()

style = "background-color: rgb(0, 170, 255);\n"

elif log.lower().find('sync') >= 0:

state = 'SYNC'

style = "background-color: rgb(0, 170, 255);\n"

elif log.lower().find('run') >= 0:

state = 'RUN'

style = "background-color: rgb(255, 255, 0);\n"

# thread operation

if not self.run\_queue.full():

self.run\_queue.put(self.esp\_process[int(str(dut\_num))],block=False)

else:

print 'thread num error'

if self.run\_flag == True:

if not self.run\_queue.empty():

esp\_process=self.run\_queue.get(block=False)

esp\_process.SIGNAL\_RESUME.emit()

self.run\_flag = False

elif log.lower().find('passed') >= 0:

state = 'TESTED'

style = "background-color: rgb(0, 170, 0);\n"

if log.find('record') >= 0:

self.\_local\_count('pass', dut\_num)

elif log.lower().find('pass') >= 0:

state = 'PASS'

style = "background-color: rgb(0, 170, 0);\n"

if log.find('record') >= 0:

self.\_local\_count('pass', dut\_num)

elif log.lower().find('fail') >= 0:

state = 'FAIL'

style = "background-color: rgb(255, 0, 0);\n"

if log.find('record') >= 0:

self.\_local\_count('fail', dut\_num)

elif log.lower().find('upload-f') >= 0:

state = 'upload-f'

style = "background-color: rgb(255, 0, 0);\n"

elif log.lower().find('upload-p') >= 0:

state = 'upload-p'

style = "background-color: rgb(0, 0, 255);\n"

elif log.lower().find('finish') >= 0:

state\_flag = False

eval('self.pbStart{}'.format(dut\_num)).setDown(False)

eval('self.pbStart{}'.format(dut\_num)).setEnabled(True)

elif log.lower().find('rfmutex') >= 0:

print "switch:{}".format(time.time())

state\_flag = False

if not self.run\_queue.empty():

esp\_process=self.run\_queue.get(block=False)

esp\_process.SIGNAL\_RESUME.emit()

if self.run\_queue.empty():

self.run\_flag = True

else:

state\_flag = False

if state\_flag:

self.\_state\_change(dut\_num, state, style)

elif log.find('[upload]') >= 0:

self.lbTotalStatus.setText(log.split(']')[-1])

elif log.find('[mac]') >= 0:

eval("self.lbMAC{}".format(dut\_num)).setText(log[-12:])

if show\_flag:

tb.append(log)

def button\_testflow\_reset(self, file\_path='./config/testFlow'):

parent = 'root'

with open(file\_path, 'r') as fd:

rl = fd.readline()

while rl != '':

if not rl.startswith('['):

rl = fd.readline()

continue

rl = rl.strip().strip('[').strip(']')

if len(rl.split(self.\_SP\_SIGN)) < 5:

print 'config file is broken, please re-generate'

return -1

level\_index, childCount, checkable, editable, value = rl.split(self.\_SP\_SIGN)

checkable = int(checkable.strip())

editable = editable.strip()

value = value.strip()

level\_index = level\_index.strip().split('-')

if editable == '1' or checkable >= 0:

item = self.trwTestFlow.invisibleRootItem()

for i in xrange(1,len(level\_index)):

tmp = item.text(0)

item = item.child(int(level\_index[i]))

if editable == '1':

item.setText(0, value)

self.test\_flow[parent] = value

if checkable >= 0:

item.setCheckState(0, checkable)

parent = value

rl = fd.readline()

self.lbFWVer.setText(self.test\_flow['USER\_FW\_VER\_STR'])

return 0

def button\_testflow\_submit(self):

first\_flag = True

while(1):

tmp\_time = time.strftime('%Y-%m-%d-%H',time.localtime(time.time()))

y,m,d,h = map(lambda x:int(x), tmp\_time.split('-'))

print (tmp\_time, (y+(m+d+h))%10000)

if first\_flag:

verify,rst = QtGui.QInputDialog().getText(self, "Verify Box", "Verify:",

QtGui.QLineEdit().Normal, '')

else:

verify,rst = QtGui.QInputDialog().getText(self, "Verify Box", "Verify: (fail, retry!!)",

QtGui.QLineEdit().Normal, '')

first\_flag = False

if rst:

try:

print verify

if int(verify) == (y+(m+d+h))%10000:

self.ui\_update\_testflow(pop\_msg=True)

print ('verify pass')

break

else:

print ('verify fail')

except:

print ('please input 4 bytes Number')

else:

break

def button\_all\_start(self):

for id in (1,2,3,4):

try:

if self.esp\_process[id]==None or (not self.esp\_process[id].isRunning()):

stdout\_ = self.\_Print(self, eval("self.tbLog"+str(id)))

self.dut\_config['common\_conf']['dut\_num'] = str(id)

self.esp\_process[id]=esp\_test.esp\_testThread(stdout\_, self.dut\_config,self.test\_flow)

self.esp\_process[id].start()

eval('self.pbStart{}'.format(id)).setEnabled(False)

eval('self.pbStart{}'.format(id)).setDown(True)

time.sleep(0.1)

except:

pass

def button\_single\_start(self, btn):

btn.setEnabled(False)

btn.setDown(True)

id = int(btn.objectName()[len(btn.objectName())-1])

if id in (1,2,3,4):

stdout\_ = self.\_Print(self, eval("self.tbLog"+str(id)))

self.dut\_config['common\_conf']['dut\_num'] = str(id)

self.esp\_process[id]=esp\_test.esp\_testThread(stdout\_, self.dut\_config,self.test\_flow)

self.esp\_process[id].start()

else:

print('error: get strat btn err')

def button\_all\_stop(self):

for id in (1,2,3,4):

try:

if self.esp\_process[id].isRunning():

self.esp\_process[id].SIGNAL\_STOP.emit()

else:

self.signal\_print\_log(eval('self.tbLog{}'.format(id)), '[state]idle')

eval('self.lbMAC{}'.format(id)).setText('0x000000000000')

except:

self.signal\_print\_log(eval('self.tbLog{}'.format(id)), '[state]idle')

eval('self.lbMAC{}'.format(id)).setText('0x000000000000')

def button\_single\_stop(self, btn):

id = int(btn.objectName()[len(btn.objectName())-1])

if self.esp\_process.has\_key(id):

# if self.esp\_process[id].isRunning():

self.esp\_process[id].SIGNAL\_STOP.emit()

def button\_pop\_log(self, index):

log\_path = self.logs\_path

if index in (1,2,3,4):

if self.esp\_process[index] != None:

log\_path = self.esp\_process[index].logpath.split('//')[-1]

log\_path = self.logs\_path + log\_path

try:

os.startfile(log\_path)

except:

print 'error path'

else:

print('error: get log btn err')

def button\_cloud\_sync(self):

self.lbSyncState.setText('try sync cloud config by mpn')

self.pbCloudSync.setEnabled(False)

self.pbCloudSync.setDown(True)

conf = ConfigParser.ConfigParser()

conf.read('./config/dutConfig')

ip = conf.get('common\_conf', 'tmp\_server\_ip')

port = conf.get('common\_conf', 'tmp\_server\_port')

#url = 'http://{}:{}/hp\_register.py?opration=config&user\_token={}'.format(ip,port,self.leMPNNo.text()) # test\_config\_002

url = 'https://{}:{}/mpn?mpnSid={}'.format(ip, port, str(self.leMPNNo.text())+'\_'+str(self.dut\_config["chip\_conf"]['chip\_type']))

try:

rsp = requests.get(url=url, verify=False,timeout=3).json()

with open('./config/cloudTestFlow', 'w') as fd:

fd.write(str(rsp['data']))

#print str(rsp['data'])

except:

self.lbSyncState.setText('sync fail!!! please check the mpn and chip type')

self.pbCloudSync.setEnabled(True)

self.pbCloudSync.setDown(False)

return

try:

self.button\_testflow\_reset(file\_path='./config/cloudTestFlow')

self.lbSyncState.setText('sync success')

except:

self.lbSyncState.setText('sync fail')

self.button\_testflow\_reset(file\_path='./config/testFlow')

self.ui\_update\_dut()

self.pbCloudSync.setEnabled(True)

self.pbCloudSync.setDown(False)

#url = 'https://{}:{}/mpn/'.format(ip, port)

#data = '{}"mpnSid":"{}", "data":"{}"{}'.format("{", "tt3", "88888888", "}")

#rsp = requests.post(url=url, verify=False).json()

#print rsp

return

def button\_dut\_reset(self, file\_path='./config/dutConfig', try\_login=False):

conf = ConfigParser.ConfigParser()

try:

conf.read(file\_path)

# set dut\_config

for i in conf.sections():

self.dut\_config[i] = dict(conf.items(i))

index = self.cbChipType.findText(self.dut\_config['chip\_conf']['chip\_type'])

if index >= 0:

self.cbChipType.setCurrentIndex(index)

self.leChipType.setHidden(True)

else:

self.cbChipType.setCurrentIndex(self.CHIP\_TYPE\_NUM-1)

self.leChipType.setText(self.dut\_config['chip\_conf']['chip\_type'])

self.leChipType.setHidden(False)

self.lePoNo.setText(self.dut\_config['common\_conf']['po\_no'])

self.leMPNNo.setText(self.dut\_config['common\_conf']['mpn\_no'])

self.leFacId.setText(self.dut\_config['common\_conf']['fac\_sid'])

self.leBatchId.setText(self.dut\_config['common\_conf']['batch\_sid'])

self.leFacPlan.setText(self.dut\_config['common\_conf']['fac\_plan'])

self.leBinPath.setText(self.dut\_config['common\_conf']['bin\_path'])

if self.dut\_config['common\_conf']['test\_from'] == 'RAM':

self.cbTestFrom.setCurrentIndex(0)

else:

self.cbTestFrom.setCurrentIndex(1)

self.cbFREQ.setCurrentIndex(self.cbFREQ.findText(self.dut\_config['chip\_conf']['freq']))

self.cbAutoStart.setChecked(True if self.dut\_config['common\_conf']['auto\_start'] == '1' else False)

self.cbEfuseMode.setCurrentIndex(int(self.dut\_config['chip\_conf']['efuse\_mode']))

self.lbChipType.setText(self.dut\_config['chip\_conf']['chip\_type'])

self.lbPoNo.setText(self.lePoNo.text())

self.lbMPNNo.setText(self.leMPNNo.text())

self.lbBatchId.setText(self.leBatchId.text())

self.lbFacPlan.setText(self.leFacPlan.text())

self.lbAutoStart.setText('ON' if self.dut\_config['common\_conf']['auto\_start'] == '1' else 'OFF')

self.lbTestMode.setText(self.dut\_config['common\_conf']['test\_from'])

for i in xrange(1, self.DUT\_NUM+1):

for j in xrange(1, 3):

eval('self.cbPort'+str(i)+'\_'+str(j)).setItemText(0, conf.get('DUT'+str(i), 'PORT'+str(j)))

eval('self.cbPort'+str(i)+'\_'+str(j)).setCurrentIndex(0)

index = eval('self.cbPortRate'+str(i)+'\_'+str(j)).findText(conf.get('DUT'+str(i), 'RATE'+str(j)))

if index >= 0:

eval('self.cbPortRate'+str(i)+'\_'+str(j)).setCurrentIndex(index)

eval('self.lePortRate'+str(i)+'\_'+str(j)).setHidden(True)

else:

eval('self.cbPortRate'+str(i)+'\_'+str(j)).setCurrentIndex(self.BAUD\_NUM-1)

eval('self.lePortRate'+str(i)+'\_'+str(j)).setText(conf.get('DUT'+str(i), 'RATE'+str(j)))

eval('self.lePortRate'+str(i)+'\_'+str(j)).setHidden(False)

if try\_login:

login\_flag=False

if self.dut\_config['common\_conf']['position'] == 'cloud':

token = self.dut\_config['common\_conf']['auth\_token']

try:

if self.\_try\_login(token)==0:

QtCore.QObject.disconnect(self.maCloud, QtCore.SIGNAL(\_fromUtf8("changed()")), self.button\_change\_position)

self.maCloud.setChecked(True)

self.maCloud.changed.connect(self.button\_change\_position)

self.wgCloudConfig.setHidden(False)

self.wgCloudShow.setHidden(False)

self.pbCloudSync.setEnabled(True)

self.twTestArea.widget(2).setEnabled(False)

self.lbPosition.setText('Cloud')

self.tePosition.setStyleSheet(\_fromUtf8("background-color: rgb(255, 255, 0);\n"

"border-color: rgb(0, 255, 255);"))

login\_flag=True

else:

print ('login fail')

except:

print ('login fail')

if not login\_flag:

self.maCloud.setChecked(False)

self.wgCloudConfig.setHidden(True)

self.wgCloudShow.setHidden(True)

self.pbCloudSync.setEnabled(False)

self.lbPosition.setText('Local')

except:

print ('load to config file fail')

def button\_dut\_submit(self):

first\_flag = True

while(1):

tmp\_time = time.strftime('%Y-%m-%d-%H',time.localtime(time.time()))

y,m,d,h = map(lambda x:int(x), tmp\_time.split('-'))

print (tmp\_time, (y+(m+d+h))%10000)

if first\_flag:

verify,rst = QtGui.QInputDialog().getText(self, "Verify Box", "Verify:",

QtGui.QLineEdit().Normal, '')

else:

verify,rst = QtGui.QInputDialog().getText(self, "Verify Box", "Verify: (fail, retry!!)",

QtGui.QLineEdit().Normal, '')

first\_flag = False

if rst:

try:

print verify

if int(verify) == (y+(m+d+h))%10000:

print ('verify pass')

self.ui\_update\_dut()

break

else:

print ('verify fail')

except:

print ('please input 4 bytes Number')

else:

break

def \_try\_login(self, token):

ip = self.dut\_config['common\_conf']['tmp\_server\_ip']

port = self.dut\_config['common\_conf']['tmp\_server\_port']

url = 'https://{}:{}/factorys'.format(ip, port)

headers = {'token':token}

try:

rsp = requests.get(url=url, headers=headers, verify=False, timeout=3).json()

print rsp

if rsp['status'] == 200:

return 0

else:

print 'login fail'

return 1

except:

print 'login fail'

return 1

def button\_change\_position(self):

QtCore.QObject.disconnect(self.maCloud, QtCore.SIGNAL(\_fromUtf8("changed()")), self.button\_change\_position)

if self.maCloud.isChecked(): # local -> cloud

first\_flag = True

while(1):

if first\_flag:

first\_flag = False

token,rst = QtGui.QInputDialog().getText(self, "Login", "Verify:",

QtGui.QLineEdit().Normal, '')

else:

token,rst = QtGui.QInputDialog().getText(self, "Login", "Verify: (fail, retry!!)",

QtGui.QLineEdit().Normal, '')

if rst:

try:

if self.\_try\_login(token)==0:

self.maCloud.setChecked(True)

self.maCloud.changed.connect(self.button\_change\_position)

self.wgCloudConfig.setHidden(False)

self.wgCloudShow.setHidden(False)

self.pbCloudSync.setEnabled(True)

self.twTestArea.widget(2).setEnabled(False)

self.lbPosition.setText('Cloud')

self.tePosition.setStyleSheet(\_fromUtf8("background-color: rgb(255, 255, 0);\n"

"border-color: rgb(0, 255, 255);"))

conf = ConfigParser.ConfigParser()

conf.read('./config/dutConfig')

conf.set('common\_conf', 'auth\_token', str(token))

conf.write(open('./config/dutConfig', 'w'))

self.ui\_update\_dut()

return

else:

print ('login fail')

except:

print ('login fail')

else:

break

self.maCloud.setChecked(False)

self.wgCloudConfig.setHidden(True)

self.wgCloudShow.setHidden(True)

self.maCloud.changed.connect(self.button\_change\_position)

self.pbCloudSync.setEnabled(False)

self.twTestArea.widget(2).setEnabled(True)

self.lbPosition.setText('Local')

self.tePosition.setStyleSheet(\_fromUtf8("background-color: rgb(255, 170, 127);\n"

"border-color: rgb(0, 255, 255);"))

def button\_showFileDialog(self):

filename = QtGui.QFileDialog.getOpenFileName(None, 'Open file', './bin/', filter='firmware(\*.bin);;all(\*.\*)', selectedFilter='firmware(\*.bin)')

self.leBinPath.setText(filename)

self.dut\_config['common\_conf']['bin\_path'] = filename

def combobox\_change\_port(self, cb\_port):

port\_list = list(serial.tools.list\_ports.comports())

cb\_port.clear()

for port in port\_list:

print port[0]

cb\_port.addItem(\_fromUtf8(port[0]))

cb\_port.showPopup()

def combobox\_change\_baud(self, cb):

if cb.currentText() == 'custom':

eval('self.lePortRate{}'.format(cb.objectName()[-3:])).setHidden(False)

else:

eval('self.lePortRate{}'.format(cb.objectName()[-3:])).setHidden(True)

def combobox\_chip\_type\_change(self, index):

if index == self.CHIP\_TYPE\_NUM - 1:

self.leChipType.setHidden(False)

else:

self.leChipType.setHidden(True)

def combobox\_test\_from\_change(self, index):

self.dut\_config['common\_conf']['test\_from'] = self.cbTestFrom.currentText()

if self.cbTestFrom.currentText() == 'RAM':

self.pbBinPath.setEnabled(True)

self.leBinPath.setEnabled(True)

else:

self.pbBinPath.setEnabled(False)

self.leBinPath.setEnabled(False)

### class functions ###

def ui\_update\_testflow(self, file\_path='./config/testFlow', pop\_msg=False):

with open(file\_path, 'w') as fd:

fd.write("- level-index $$ childCount$$ checkable$$ editable$$ value -\n")

self.\_testflow\_general(fd, self.trwTestFlow.invisibleRootItem(), '0')

self.lbFWVer.setText(self.test\_flow['USER\_FW\_VER\_STR'])

if pop\_msg:

msg = QtGui.QMessageBox(QtGui.QMessageBox.NoIcon, '!!!','The Test Flow update succ!! ')

msg.exec\_()

def \_testflow\_general(self, fd, root, level\_index):

if(root.flags()&QtCore.Qt.ItemIsEditable):

self.test\_flow[str(root.parent().text(0))] = str(root.text(0))

else:

self.test\_flow[str(root.text(0))] = root.checkState(0)

fd.write("[%-12s%s%2d%s%2d%s%2d%s %s]\n"%(level\_index, self.\_SP\_SIGN, int(root.childCount()), self.\_SP\_SIGN,

int(root.checkState(0) if(root.flags()&QtCore.Qt.ItemIsUserCheckable) else -1), self.\_SP\_SIGN,

int(1 if(root.flags()&QtCore.Qt.ItemIsEditable) else 0), self.\_SP\_SIGN, str(root.text(0))))

for i in xrange(root.childCount()):

self.\_testflow\_general(fd, root.child(i), level\_index+'-'+str(i))

def ui\_update\_dut(self, file\_path='./config/dutConfig'):

conf = ConfigParser.ConfigParser()

conf.read(file\_path)

if not conf.has\_section('common\_conf'):

conf.add\_section('common\_conf')

if not conf.has\_section('chip\_conf'):

conf.add\_section('chip\_conf')

conf.set('common\_conf', 'POSITION', 'cloud' if self.maCloud.isChecked() else 'local')

conf.set('common\_conf', 'TEST\_FROM', self.cbTestFrom.currentText())

conf.set('chip\_conf', 'FREQ', self.cbFREQ.currentText())

conf.set('chip\_conf', 'efuse\_mode', self.cbEfuseMode.currentIndex())

conf.set('common\_conf', 'auto\_start', '1' if self.cbAutoStart.checkState() else '0')

if self.cbChipType.currentIndex() < self.CHIP\_TYPE\_NUM - 1:

conf.set('chip\_conf', 'CHIP\_TYPE', self.cbChipType.currentText())

else:

conf.set('chip\_conf', 'CHIP\_TYPE', self.leChipType.text())

conf.set('common\_conf', 'PO\_NO', self.lePoNo.text())

conf.set('common\_conf', 'MPN\_NO', self.leMPNNo.text())

conf.set('common\_conf', 'FAC\_SID', self.leFacId.text())

conf.set('common\_conf', 'BATCH\_SID', self.leBatchId.text())

conf.set('common\_conf', 'FAC\_PlAN', self.leFacPlan.text())

conf.set('common\_conf', 'BIN\_PATH', self.leBinPath.text())

for i in xrange(1, self.DUT\_NUM+1):

if not conf.has\_section('DUT'+str(i)):

conf.add\_section('DUT'+str(i))

for j in xrange(1, 3):

conf.set('DUT'+str(i), 'PORT'+str(j), str(eval('self.cbPort'+str(i)+'\_'+str(j)).currentText()))

if eval('self.cbPortRate'+str(i)+'\_'+str(j)).currentIndex() < self.BAUD\_NUM-1:

conf.set('DUT'+str(i), 'RATE'+str(j), str(eval('self.cbPortRate'+str(i)+'\_'+str(j)).currentText()))

else:

conf.set('DUT'+str(i), 'RATE'+str(j), str(eval('self.lePortRate'+str(i)+'\_'+str(j)).text()))

conf.write(open(file\_path, 'w'))

self.button\_dut\_reset(file\_path)

if file\_path=='./config/dutConfig':

msg = QtGui.QMessageBox(QtGui.QMessageBox.NoIcon, '!!!','The DUT config update succ!! ')

msg.exec\_()

def \_state\_change(self, dut\_num, state, style):

leState = eval("self.leStatus{}".format(dut\_num))

leState.setText(state)

leState.setStyleSheet(\_fromUtf8(style+"color: rgb(255, 255, 255);"))

def \_local\_count(self, rst, dut\_num):

self.mutex.acquire()

# total, pass, fail, mac, time

datas = [0,0,0,0,0]

mac = eval("self.lbMAC{}".format(dut\_num)).text()

with open('./config/localCount.txt', 'a+') as fd:

rls = fd.readlines()

if len(rls) > 0:

for i in xrange(1, len(rls)+1 if len(rls)<3 else 4):

if len(rls[-1 \* i].split(',')) >= 4:

datas = rls[-1].split(',')

break

try:

total = int(datas[0])

pass\_num = int(datas[1])

fail\_num = int(datas[2])

except:

total = 0

pass\_num = 0

fail\_num = 0

now\_time = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')[5:]

if rst == 'pass':

pass\_num += 1

elif rst == 'fail':

fail\_num += 1

total += 1

fd.write(str(total)+','+str(pass\_num)+','+str(fail\_num)+','+str(mac)+','+now\_time+"\n")

with open('./mac\_list/'+self.dut\_config['common\_conf']['fac\_plan']+'\_'+rst+'.csv', 'a') as fd:

fd.write(mac+'\n')

self.mutex.release()

self.lbWorkStat.setText('pass:{}/ fail:{}'.format(pass\_num, fail\_num))

def main():

app = QtGui.QApplication(sys.argv)

ui = FactoryToolUI()

ui.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

main()

# read\_log\_to\_csv\_noprint.py

import sys

import os

#import xlwt

import xlrd

csv\_list=[ #item whose failed data will be put into a .csv

'dco\_sweep\_test\_ADC\_STEP',

'RX\_PATH\_GAIN',

'RX\_SWITCH\_GAIN',

'TXIQ',

'TXBB\_TXIQ',

'TXDC',

'FREQ\_OFFSET',

'fb\_rx\_num\_max',

'RXIQ\_TEST\_5M\_diff',

'pass'

]

judge\_list=[ # check list: item in this list will be checked in the script

#the upper the more prioritized

'pass',

'dco\_sweep\_test\_ADC\_STEP',

'RX\_PATH\_GAIN',

'RX\_SWITCH\_GAIN',

'TXIQ',

'TXBB\_TXIQ',

'TXDC',

'FREQ\_OFFSET',

'fb\_rx\_num\_max',

'RXIQ\_TEST\_5M\_diff',

'VDD33',

'FREQ\_OFFSET',

'rc\_cal\_dout',

'dco\_sweep\_test\_ADC\_STEP',

#'dco\_sweep\_test\_DCO',

'TXIQ',

'TXBB\_TXIQ',

'TXDC',

#'TXBB\_TXDC',##############################

'TXCAP\_TMX2G\_CCT\_LOAD',

'TXCAP\_PA2G\_CCT\_STG1',

'TXCAP\_PA2G\_CCT\_STG2',

'TX\_PWRCTRL\_ATTEN',

'TX\_PWCTRL\_CHAN\_OFFSET',

'RX\_PATH\_GAIN',

'RX\_PATH\_SNR',

'ADC\_DAC\_SNR',

'RX\_SWITCH\_GAIN',

#'TXBB\_TXDC',##############################

##=========================================================

'RX\_GAIN\_CHECK',

'BBRX2\_RXIQ',

'RX\_NOISEFLOOR',

'RXIQ',

'RXDC',

'RTC\_freq\_170khz',

'RTC\_freq\_70khz',

'RXIQ\_TEST\_5M\_diff',

'rxiq\_cover\_fail\_num',

'rombist\_rslt',

'timeout\_fail',

'site\_num',

'RXIQ\_REMAIN',

'fb\_rxrssi',

'dut\_rxrssi',

'fb\_rx\_num\_max',

'fb\_rx\_num\_sum',

'io\_test\_result',

'wifi\_init\_time',

]

threshold={ #threshold for checking items

# 'name':[[lowlist,],[highlist,]]

'dco\_sweep\_test\_ADC\_STEP':[[0],[5]],

'RX\_PATH\_GAIN':[[40],[48]],

'RX\_SWITCH\_GAIN':[[1,-9,3],[4,-6,6]],

'TXIQ':[[-12,-25],[12,25]],

'TXBB\_TXIQ':[[-6],[6]],

'TXDC':[[3],[124]],

'FREQ\_OFFSET':[[0],[32]],

'fb\_rx\_num\_max':[[15],[16]],

'RXIQ\_TEST\_5M\_diff':[[-8],[8]] ,

'VDD33':[[3200,0,-100],[3600,180,0]],

'rc\_cal\_dout':[[3],[60]],

'TXCAP\_TMX2G\_CCT\_LOAD':[[0],[15]],

'TXCAP\_PA2G\_CCT\_STG1':[[0],[12]],

'TXCAP\_PA2G\_CCT\_STG2':[[0],[6]],

'TX\_PWRCTRL\_ATTEN':[[-4,8,18,25,30,42],[26,30,38,47,53,63]],

'TX\_PWCTRL\_CHAN\_OFFSET':[[-10],[10]],

'RX\_PATH\_GAIN':[[40],[48]],

'RX\_PATH\_SNR':[[25],[5000]],

'ADC\_DAC\_SNR':[[34],[5000]],

'RX\_SWITCH\_GAIN':[[1,-9,3],[4,-6,6]],

#'TXBB\_TXDC',##############################

#======================================================

'RX\_GAIN\_CHECK':[[16,-25,15,-4,-6,-4,5,-2,-2,-2,-2,-2,-2,-2],[24,-17,20,1,0,1,14,2,2,2,2,2,2,2]],

'BBRX2\_RXIQ':[[-3],[3]],

'RX\_NOISEFLOOR':[[-390],[-370]],

'RXIQ':[[-13,-27],[13,27]],

'RXDC':[[128],[384]],

'RTC\_freq\_170khz':[[140],[210]],

'RTC\_freq\_70khz':[[60],[90]],

'RXIQ\_TEST\_5M\_diff':[[-8],[8]],

'rxiq\_cover\_fail\_num':[[0],[0]],

'rombist\_rslt':[[0],[0]],

'timeout\_fail':[[0,0,0],[1,1,0]],

'site\_num':[[0,1,1,0],[100,1,100,96]],

'RXIQ\_REMAIN':[[-200],[-30]],

'fb\_rxrssi':[[0,0,0,0,0,40],[100,100,100,100,100,50]],

'dut\_rxrssi':[[0,0,0,0,0,50],[100,100,100,100,100,60]],

'fb\_rx\_num\_max':[[15],[16]],

'fb\_rx\_num\_sum':[[64,0],[96,2]],

'io\_test\_result':[[0],[0]],

'wifi\_init\_time':[[370000],[470000]],

}

fail\_dict\_1st\_order={ }

for litem in judge\_list:

fail\_dict\_1st\_order[litem]=[0,[]]

def read\_log\_data(file\_path,mode,block\_num):

value\_list=[]

end\_flg=''

values={ 'rc\_cal\_dout': [[],[]],

'rx\_para\_cal':[['\_1','\_2','\_3'],[]],

'CHIP\_ID': [[''],[]],

'CHIP\_VERSION':[[''],[]],

'TEST\_NUM':[[],[]],

'vdd33':[[],[]],

'VDD33':[[],[]],

'TX\_VDD33':[[],[]],#add for new version

'TX\_VDD33\_DIFF':[[''],[]],#add for new version

'txp\_result':[[],[]],#add for new version

'TOUT':[[],[]],#ADD FOR TOUT TEST

'cal\_rf\_ana\_gain':[[],[]],

'TXBB\_TXIQ':[['\_gain','\_phase'],[ ]],

'TXBB\_TXDC':[['\_i','\_q'],[]],##############################

'RX\_GAIN\_CHECK':[['\_CH1','\_CH6','\_CH11'],[]],

'RX\_GAIN\_CHECK\_POWER\_hdb':[['\_1','\_2'],[]],

'BBRX2\_RXIQ':[['\_gain','\_phase'],[]],

'RX\_NOISEFLOOR':[['\_CH1','\_CH6','\_CH11'],[]],

'TXCAP\_TMX2G\_CCT\_LOAD':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG1':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG2':[[''],[]],

'TX\_POWER\_BACKOFF':[[''],[]],#add for new version

'TX\_PWRCTRL\_ATTEN':[[''],[]],

'TX\_PWCTRL\_CHAN\_OFFSET':[[''],[]],

'TXIQ':[['\_gain','\_phase'],[]],

'BT\_TXIQ':[['\_gain','\_phase'],[]],

'TXDC':[['\_i','\_q'],[]],

'BT\_TXDC':[['\_i','\_q'],[]],

'RXIQ':[['\_gain','\_phase'],[]],

'RXDC':[['\_c\_i','\_c\_q','\_f\_i','\_f\_q'],[]],

'RXDC\_RFRX\_BT':[[],[]],

'RXDC\_RFRX\_WIFI':[[],[]],

'RXDC\_RXBB\_BT':[[],[]],

'freq\_offset\_cal':[[],[]],

'RX\_PATH\_GAIN':[[''],[]],

'RXIQ\_tot\_power':[[''],[]],

'FREQ\_OFFSET':[[''],[]],

'RX\_PATH\_SNR':[[''],[]],

'adc\_dac\_snr\_2tone':[[],[]],

'ADC\_DAC\_SNR':[[''],[]],

'rx\_switch\_gain\_check':[['\_bbrx1','\_bbrx2','\_total\_pwr\_db','\_sig\_pwr\_db','\_sw\_g'],[]],#####################################

'RX\_SWITCH\_GAIN':[[''],[]],#######################################

'dco\_sweep\_test\_ADC\_STEP':[[],[]],

'dco\_sweep\_test\_DCO':[[],[]],

'wi\_pad 0 and ri\_pad 4':[[],[]],

'wi\_pad 3 and ri\_pad 5':[[],[]],

'RTC\_freq\_170khz':[[''],[]],

'RTC\_freq\_70khz':[[''],[]],

'DVDD\_testV1':[[],[]],

'VDD\_RTC\_testV1':[[],[]],

'DVDD\_testV2' :[[],[]],

'VDD\_RTC\_testV2':[[],[]],

'LightSleep\_IDD\_VBAT' :[[],[]],

'LightSleep\_IDD\_DVDD\_IO':[[],[]],

'DeepSleep\_IDD\_VBAT':[[],[]],

'Chip\_PD\_IDD\_VBAT':[[],[]],

'Chip\_PD\_IDD\_DVDD\_IO':[[],[]],

'DeepSleep\_IDD\_DVDD\_IO' :[[],[]],

'AnaWorkIDD\_VBAT':[[],[]],

'AnaWorkIDD\_DVDD\_IO':[[],[]],

'rssi':[[''],[]],

'rx\_suc\_num':[[''],[]] ,

'RXIQ\_TEST\_-5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M\_diff':[['\_gain','\_phase'],[]],

'rxiq\_cover\_fail\_num':[[],[]],

'rxiq\_compute\_num':[[''],[]],

'rombist\_rslt':[[],[]],

'timeout\_fail':[[],[]],

'site\_num':[[],[]],

'RXIQ\_REMAIN':[[''],[]], # maintain IQ OR DC , print according to the name str in value[item] [0]

'txp\_pwctrl\_atten':[[''],[]],

'fb\_rxrssi':[[''],[]],

'dut\_rxrssi':[[''],[]],

'fb\_rx\_num':[[''],[]],

'fb\_rx\_num\_max':[[''],[]],

'dut\_rx\_num':[[''],[]],

'fb\_rx\_num\_sum':[[],[]],

'txp\_state':[[''],[]],

'rxsdut\_cnt':[[''],[]],

'rxsdut\_max\_rssi':[[''],[]],

'txp\_result':[[''],[]],

'txreq\_start\_time':[[],[]],

'check\_result\_t':[[],[]],

'io\_test\_result':[[],[]],

'wifi\_init\_time':[[''],[]],

'WIFI\_INIT\_ITEM':[[''],[]],

'SVN\_Version':[[''],[]],

'rx\_para\_cal\_tone':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_1':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_2':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_3':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_4':[[''],[]],

'filepath':'',

'timer expire':''

}

v\_tmp={ 'rc\_cal\_dout': [[],[]],

'rx\_para\_cal':[['\_1','\_2','\_3'],[]],

'CHIP\_ID': [[''],[]],

'CHIP\_VERSION':[[''],[]],

'TEST\_NUM':[[],[]],

'vdd33':[[],[]],

'VDD33':[[],[]],

'TX\_VDD33':[[],[]],#add for new version

'TX\_VDD33\_DIFF':[[''],[]],#add for new version

'txp\_result':[[],[]],#add for new version

'TOUT':[[],[]],#ADD FOR TOUT TEST

'cal\_rf\_ana\_gain':[[],[]],

'TXBB\_TXIQ':[['\_gain','\_phase'],[ ]],

'TXBB\_TXDC':[['\_i','\_q'],[]],##############################

'RX\_GAIN\_CHECK':[['\_CH1','\_CH6','\_CH11'],[]],

'RX\_GAIN\_CHECK\_POWER\_hdb':[['\_ '],[]],

'BBRX2\_RXIQ':[['\_gain','\_phase'],[]],

'RX\_NOISEFLOOR':[['\_CH1','\_CH6','\_CH11'],[]],

'TXCAP\_TMX2G\_CCT\_LOAD':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG1':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG2':[[''],[]],

'TX\_POWER\_BACKOFF':[[''],[]],#add for new version

'TX\_PWRCTRL\_ATTEN':[[''],[]],

'TX\_PWCTRL\_CHAN\_OFFSET':[[''],[]],

'TXIQ':[['\_gain','\_phase'],[]],

'BT\_TXIQ':[['\_gain','\_phase'],[]],

'TXDC':[['\_i','\_q'],[]],

'BT\_TXDC':[['\_i','\_q'],[]],

'RXIQ':[['\_gain','\_phase'],[]],

'RXDC':[['\_c\_i','\_c\_q','\_f\_i','\_f\_q'],[]],

'RXDC\_RFRX\_BT':[[],[]],

'RXDC\_RFRX\_WIFI':[[],[]],

'RXDC\_RXBB\_BT':[[],[]],

'freq\_offset\_cal':[[],[]],

'RX\_PATH\_GAIN':[[''],[]],

'RXIQ\_tot\_power':[[''],[]],

'FREQ\_OFFSET':[[''],[]],

'RX\_PATH\_SNR':[[''],[]],

'adc\_dac\_snr\_2tone':[[],[]],

'ADC\_DAC\_SNR':[[''],[]],

'rx\_switch\_gain\_check':[['\_bbrx1','\_bbrx2','\_total\_pwr\_db','\_sig\_pwr\_db','\_sw\_g'],[]],#####################################

'RX\_SWITCH\_GAIN':[[''],[]],#######################################

'dco\_sweep\_test\_ADC\_STEP':[[],[]],

'dco\_sweep\_test\_DCO':[[],[]],

'wi\_pad 0 and ri\_pad 4':[[],[]],

'wi\_pad 3 and ri\_pad 5':[[],[]],

'RTC\_freq\_170khz':[[''],[]],

'RTC\_freq\_70khz':[[''],[]],

'DVDD\_testV1':[[],[]],

'VDD\_RTC\_testV1':[[],[]],

'DVDD\_testV2' :[[],[]],

'VDD\_RTC\_testV2':[[],[]],

'LightSleep\_IDD\_VBAT' :[[],[]],

'LightSleep\_IDD\_DVDD\_IO':[[],[]],

'DeepSleep\_IDD\_VBAT':[[],[]],

'DeepSleep\_IDD\_DVDD\_IO' :[[],[]],

'Chip\_PD\_IDD\_VBAT':[[],[]],

'Chip\_PD\_IDD\_DVDD\_IO':[[],[]],

'AnaWorkIDD\_VBAT':[[],[]],

'AnaWorkIDD\_DVDD\_IO':[[],[]],

'rssi':[[''],[]],

'rx\_suc\_num':[[''],[]] ,

'RXIQ\_TEST\_-5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M\_diff':[['\_gain','\_phase'],[]],

'rxiq\_cover\_fail\_num':[[],[]],

'rxiq\_compute\_num':[[''],[]],

'rombist\_rslt':[[],[]],

'timeout\_fail':[[],[]],

'site\_num':[[],[]],

'RXIQ\_REMAIN':[[''],[]],

'txp\_pwctrl\_atten':[[''],[]],

'fb\_rxrssi':[[''],[]],

'dut\_rxrssi':[[''],[]],

'fb\_rx\_num':[[''],[]],

'fb\_rx\_num\_max':[[''],[]],

'dut\_rx\_num':[[''],[]],

'fb\_rx\_num\_sum':[[],[]],

'txp\_state':[[''],[]],

'rxsdut\_cnt':[[''],[]],

'rxsdut\_max\_rssi':[[''],[]],

'txp\_result':[[''],[]],

'txreq\_start\_time':[[],[]],

'check\_result\_t':[[],[]],

'io\_test\_result':[[],[]],

'wifi\_init\_time':[[''],[]],

'WIFI\_INIT\_ITEM':[[''],[]],

'SVN\_Version':[[''],[]],

'rx\_para\_cal\_tone':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_1':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_2':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_3':[[''],[]],

'rx\_para\_cal\_tone\_sig\_pwr\_db\_4':[[''],[]],

'filepath':'',

'timer expire':''

}

if mode=='module':

end\_flg='TEST\_NUM'

elif mode=='ate\_log':

end\_flg='---------------CHECK BOARD PASS'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

elif mode=='ate\_0530\_2noisefloor':

end\_flg='AnaWorkIDD\_DVDD\_IO'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].pop()

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].pop()

elif mode=='ate\_0530\_4noisefloor':

end\_flg='AnaWorkIDD\_DVDD\_IO'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].append('\_CH14')

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].append('\_CH14')

elif mode=='ate\_new':

end\_flg='---------------CHECK'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].append('\_CH14')

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].append('\_CH14')

elif mode=='module2515':

#end\_flg='MODULE\_TEST END'

end\_flg='TEST\_NUM'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].append('\_CH14')

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].append('\_CH14')

elif mode=='ESP32':

end\_flg='MODULE\_TEST EDN!!!'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

elif mode=='ate' :

end\_flg='---------------CHECK'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

elif mode=='130608\_fpga' :

#end\_flg='user code done'

end\_flg='rx\_suc\_num'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].pop()

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].pop()

elif mode=='130624\_fpga' :

#end\_flg='user code done'

end\_flg='rxsdut\_max\_rssi'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].pop()

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].pop()

elif mode=='130626\_fpga' :

end\_flg='txp\_result'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].pop()

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].pop()

elif mode =='ate130716':

end\_flg='AnaWorkIDD\_DVDD\_IO'

values['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

values['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

values['RX\_NOISEFLOOR'][0].pop()

v\_tmp['dco\_sweep\_test\_ADC\_STEP'][0]=['\_min\_i','\_max\_i','\_min\_q','\_max\_q']

v\_tmp['dco\_sweep\_test\_DCO'][0]=['\_low\_i','\_hgh\_i','\_low\_q','\_hgh\_q']

v\_tmp['RX\_NOISEFLOOR'][0].pop()

f=open(file\_path,'r')

lines=f.readlines()

for line in lines:

if 'site\_num,' in line:

line=line.replace(',','=').replace(';',',')

elif 'dco\_sweep\_test\_ADC\_STEP' in line or 'dco\_sweep\_test\_DCO' in line:

line=line.replace(';',',')

elif 'RTC\_freq\_170khz' in line or 'RTC\_freq\_70khz' in line:

line=line.replace('=',',')

elif 'rssi' in line or 'rx\_suc\_num' in line:

line=line.replace(':',',')

elif 'TOUT' in line:

line = line.replace('=',',')

line=line.replace(':',',').replace('PPM','').replace('us','').replace('dB','').strip('\n').strip('\n\r').strip(' ').strip(';').strip(',').split(',')

if '=' in line[0]:

line[0]=line[0].split('=')[0].strip(' ')+','+line[0]

line=','.join(line)

line=line.split(',')

item=line[0]

if item == "dut\_rx\_num" or item == "fb\_rx\_num" or item=="dut\_rssi" or item=="fb\_rssi":

continue

line=','.join(line[1:]).split(';')

for i in range(len(line)):

line[i]=line[i].split(',')

if item in values.keys():

if item == "TX\_VDD33":

if not values['vdd33'][1]==[]:

values['TX\_VDD33\_DIFF'][1].append([(values['vdd33'][1][0][0]-int(line[0][0].split('=')[1]))])

else:

values['TX\_VDD33\_DIFF'][1].append([(3300-int(line[0][0].split('=')[1]))])

if item == "TOUT":

values['TOUT'][1].append(int(line[0][0]))

elif '=' in line[0][0]:

line=line[0]

v=[]

for i in range(len(line)):

if '=' in line[i]:

line[i]=line[i].split('=')

values[item][0].append(line[i][0])

try:

v.append(int(line[i][1]))

except:

v.append(line[i][1])

values[item][1].append(v)

elif 'IQ' in item or 'DC' in item or 'rx\_switch\_gain\_check'==item or 'RX\_GAIN\_CHECK\_POWER\_hdb'==item: #any line include ';'

for j in range(len(line[0])):

v=[]

for k in range(len(line)):

v.append(line[k][j])

values[item][1].append(v)

elif item=='timer expire':

if 'pass' in line[0][0]:

value\_list[-1][item]='pass'

elif 'fail' in line[0][0]:

value\_list[-1][item]='fail'

elif item == 'TX\_VDD33':

print "item:",item

print "line:",line,'\r\n'

else:

values[item][1].append(line[0])

if end\_flg in item:

if not values['CHIP\_ID'][1]==[]:

values['filepath']=file\_path

if not values['RXIQ\_TEST\_-5M'][1]==[] and not values['RXIQ\_TEST\_5M'][1]==[]:

values['RXIQ\_TEST\_5M\_diff'][1].append([ str( int(values['RXIQ\_TEST\_-5M'][1][0][0])-int(values['RXIQ\_TEST\_5M'][1][0][0])) ])

values['RXIQ\_TEST\_5M\_diff'][1].append([ str( int(values['RXIQ\_TEST\_-5M'][1][1][0])-int(values['RXIQ\_TEST\_5M'][1][1][0])) ])

if not values['fb\_rx\_num'][1]==[]:

values['fb\_rx\_num\_max'][1].append( [ str( max( [ int(d) for d in values['fb\_rx\_num'][1][0] ]) )] )

value\_list.append(values)

values=v\_tmp

v\_tmp={ 'rc\_cal\_dout': [[''],[]],

'rx\_para\_cal':[['\_1','\_2','\_3'],[]],

'CHIP\_ID': [[''],[]],

'CHIP\_VERSION':[[''],[]],

'TEST\_NUM':[[],[]],

'vdd33':[[],[]],

'VDD33':[[],[]],

'TX\_VDD33':[[],[]],#add for new version

'TX\_VDD33\_DIFF':[[''],[]],#add for new version

'txp\_result':[[],[]],#add for new version

'TOUT':[[],[]],#ADD FOR TOUT TEST

'cal\_rf\_ana\_gain':[[],[]],

'TXBB\_TXIQ':[['\_gain','\_phase'],[ ]],

'TXBB\_TXDC':[['\_i','\_q'],[]],##############################

'RX\_GAIN\_CHECK':[['\_CH1','\_CH6','\_CH11'],[]],

'RX\_GAIN\_CHECK\_POWER\_hdb':[['\_ '],[]],

'BBRX2\_RXIQ':[['\_gain','\_phase'],[]],

'RX\_NOISEFLOOR':[['\_CH1','\_CH6','\_CH11'],[]],

'TXCAP\_TMX2G\_CCT\_LOAD':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG1':[[''],[]],

'TXCAP\_PA2G\_CCT\_STG2':[[''],[]],

'TX\_POWER\_BACKOFF':[[''],[]],#add for new version

'TX\_PWRCTRL\_ATTEN':[[''],[]],

'TX\_PWCTRL\_CHAN\_OFFSET':[[''],[]],

'TXIQ':[['\_gain','\_phase'],[]],

'TXDC':[['\_i','\_q'],[]],

'BT\_TXIQ':[['\_gain','\_phase'],[]],

'BT\_TXDC':[['\_i','\_q'],[]],

'RXIQ':[['\_gain','\_phase'],[]],

'RXDC':[['\_c\_i','\_c\_q','\_f\_i','\_f\_q'],[]],

'RXDC\_RFRX\_BT':[[],[]],

'RXDC\_RFRX\_WIFI':[[],[]],

'RXDC\_RXBB\_BT':[[],[]],

'freq\_offset\_cal':[[],[]],

'RX\_PATH\_GAIN':[[''],[]],

'FREQ\_OFFSET':[[''],[]],

'RXIQ\_tot\_power':[[''],[]],

'RX\_PATH\_SNR':[[''],[]],

'adc\_dac\_snr\_2tone':[[],[]],

'ADC\_DAC\_SNR':[[''],[]],

'rx\_switch\_gain\_check':[['\_bbrx1','\_bbrx2','\_total\_pwr\_db','\_sig\_pwr\_db','\_sw\_g'],[]],#####################################

'RX\_SWITCH\_GAIN':[[''],[]],#######################################

'dco\_sweep\_test\_ADC\_STEP':[[],[]],

'dco\_sweep\_test\_DCO':[[],[]],

'wi\_pad 0 and ri\_pad 4':[[],[]],

'wi\_pad 3 and ri\_pad 5':[[],[]],

'RTC\_freq\_170khz':[[],[]],

'RTC\_freq\_70khz':[[],[]],

'DVDD\_testV1':[[],[]],

'VDD\_RTC\_testV1':[[],[]],

'DVDD\_testV2' :[[],[]],

'VDD\_RTC\_testV2':[[],[]],

'LightSleep\_IDD\_VBAT' :[[],[]],

'LightSleep\_IDD\_DVDD\_IO':[[],[]],

'DeepSleep\_IDD\_VBAT':[[],[]],

'DeepSleep\_IDD\_DVDD\_IO' :[[],[]],

'Chip\_PD\_IDD\_VBAT':[[],[]],

'Chip\_PD\_IDD\_DVDD\_IO':[[],[]],

'AnaWorkIDD\_VBAT':[[],[]],

'AnaWorkIDD\_DVDD\_IO':[[],[]],

'rssi':[[''],[]],

'rx\_suc\_num':[[''],[]] ,

'RXIQ\_TEST\_-5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M':[['\_gain','\_phase'],[]],

'RXIQ\_TEST\_5M\_diff':[['\_gain','\_phase'],[]],

'rxiq\_cover\_fail\_num':[[],[]],

'rxiq\_compute\_num':[[''],[]],

'rombist\_rslt':[[],[]],

'timeout\_fail':[[],[]],

'site\_num':[[],[]],

'RXIQ\_REMAIN':[[''],[]],

# esp\_test.py

from PyQt4 import QtGui,QtCore

import os

import sys

import time

import serial

import esptool

import espefuse

import binascii

import subprocess

import read\_xls\_to\_h

import serial\_print\_uart\_download as spud

import threading

import timer

import read\_log\_to\_csv\_noprint as rl

import sqlite3

import serial

import get\_phy\_init

import serialCmd

import upload\_server

import datetime

import visa

import esp\_rpt

import espDownloader

import fwcheck\_ramdownload

#from param\_save\_load import \*

DEBUG = 0

sys.path.append('../')

import upload\_to\_server.upload\_to\_server as upload\_to\_server

class TestError(RuntimeError):

"""

Wrapper class for runtime errors that caused any error in test

"""

def \_\_init\_\_(self, message):

RuntimeError.\_\_init\_\_(self, message)

@staticmethod

def WithResult(message, result):

"""

Return a fatal error object that appends the hex values of

'result' as a string formatted argument.

"""

message += " (result was %s)" % hexify(result)

return TestError(message)

class esp\_testThread(QtCore.QThread):

SIGNAL\_STOP = QtCore.pyqtSignal()

SIGNAL\_RESUME = QtCore.pyqtSignal()

def \_\_init\_\_(self,\_stdout,dutconfig,testflow):

super(esp\_testThread,self).\_\_init\_\_(parent=None)

self.SIGNAL\_STOP.connect(self.ui\_stop)

self.SIGNAL\_RESUME.connect(self.thread\_resume)

self.param\_read=1

self.logpath=''

self.esp\_logstr=''

self.set\_params(\_stdout,dutconfig,testflow)

self.thread\_pause=1

self.MAC = '000000000000'

self.set\_mac\_en=0

self.tool\_ver='V0.0.1'

self.rptstr='TESTITEM'+','+'TESTVALUE'+','+'SPEC\_L'+','+'SPEC\_H'+','+'RESULT'+'\n'

if(self.chip\_type == "ESP32"):

self.THRESHOLD\_DICT=rl.get\_threshold\_dict('ATE','.//Threshold//full\_Threshold\_32.xlsx')

elif(self.chip\_type=='ESP8266'):

self.THRESHOLD\_DICT=rl.get\_threshold\_dict('ATE','../RF\_test/Threshold/full\_Threshold\_8266.xlsx')

if self.chip\_type=='ESP32':

self.memory\_download= espDownloader.ESP32FACTORY(frame=self, port=self.COMPORT, baudrate=self.BAUDRATE,

name=self.user\_fw\_download\_port,chip=self.chip\_type,

sub\_chip=self.sub\_chip\_type)

elif self.chip\_type == 'ESP8285':

self.memory\_download= espDownloader.ESP8285FACTORY(frame=self, port=self.COM\_PORT\_STR, baudrate=self.BAUDRATE,

name=self.user\_fw\_download\_port,chip=self.chip\_type)

elif (self.chip\_type == 'ESP8266') or (self.chip\_type == 'ESP8089'):

self.memory\_download= espDownloader.ESP8266FACTORY(frame=self, port=self.COMPORT, baudrate=self.BAUDRATE,

name=self.user\_fw\_download\_port,chip=self.chip\_type)

def run(self):

while True:

self.ui\_print("[state]idle clear")

try:

self.main\_test()

except:

pass

if self.autostartEn:

self.msleep(3000)

else:

break

try:

self.ser.close()

except:

pass

self.l\_print(0,'quit test')

self.ui\_print('USER QUIT TEST')

self.ui\_print('[state]finish btn up')

def main\_test(self):

self.resflag = 0

self.tester\_con\_flg=1

self.memory\_download.stopFlg=0

self.tx\_test\_res=0

self.rx\_test\_res=0

self.thread\_pause=1

self.mutex\_send\_flag = 0

self.logpath=''

self.esp\_logstr=''

self.l\_print(0,str(self.THRESHOLD\_DICT))

err = 0

def CHECK(err, err\_msg):

if err == 0:

return

self.resflag = err

self.ui\_print(err\_msg)

self.STOPTEST()

raise TestError(err\_msg)

CHECK(self.check\_param(), 'PARAM READ ERROR')

CHECK(self.try\_sync(), 'SYNC FAIL')

self.ui\_print('CHIP SYCN OK')

CHECK(self.check\_chip(), 'CHIP CHECK FAIL')

if self.loadmode == 1: # need load bin on ram mode

CHECK(self.load\_to\_ram(self.IMGPATH), 'LOAD RAM FAIL')

self.ui\_print('[state]RUN')

err, log=self.rf\_test\_catch\_log()

CHECK(err, 'GET TEST LOG FAIL')

if(self.en\_analog\_test):

CHECK(self.rf\_test\_analogtest(log),' ANALOG TEST FAIL\nEND TEST SEQUENCE')

if(self.en\_tx\_test):

CHECK(self.rf\_test\_txtest(log), 'TX TEST FAIL\nEND TEST SEQUENCE')

if(self.en\_rx\_test):

CHECK(self.rf\_test\_rxtest(log), 'RX TEST FAIL\nEND TEST SEQUENCE')

CHECK(self.general\_test\_gpio(), 'GPIO TEST FAIL\nEND TEST SEQUENCE')

if(self.loadmode==2):

CHECK(self.esp\_write\_flash(), 'WRITE PASS INFO FAIL\nEND TEST SEQUENCE')

CHECK(self.reboot(), 'REBOOT FAIL\nEND TEST SEQUENCE')

self.l\_print(0,'read pass flag ok')

self.ui\_print('REBOOT OK')

if self.en\_user\_fw\_check:

CHECK(self.general\_test\_fwcheck(), 'FIRMWARE CHECK FAIL\nEND TEST SEQUENCE')

self.ui\_print('ALL ITEM PASSED')

self.STOPTEST()

### RF TEST -------------------------------- ###

def rf\_test\_txtest(self, lg):

"""

Run tx packet test

Returns: err

0: success

1: fail

"""

self.l\_print(0,'start tx test')

self.l\_print(0,'"TEST ITEM:')

if self.tx\_test\_res == 0:

self.tx\_test\_res = 1

print(0,self.THRESHOLD\_DICT.keys())

if 'fb\_rx\_num' in self.THRESHOLD\_DICT.keys() and self.en\_tx\_test:

self.l\_print(0,'TX TEST BEGIN')

tx\_log=''

log\_tx\_rx = lg[0].split('\n')

thres\_tmp = self.THRESHOLD\_DICT['fb\_rx\_num']

self.l\_print(3,("fb\_rx\_num \t%r\n" %(thres\_tmp)))

for log\_tmp in log\_tx\_rx[-20:]:

if "tx packet test" in log\_tmp:

self.tx\_test\_res = 1

break

#self.tx\_test\_res=0

tx\_data=[]

for log\_tmp in log\_tx\_rx[-20:]:

if 'FREQ\_OFFSET' in log\_tmp:

freq\_data = log\_tmp.split(',')[1:-1]

freq\_data = [int(x) for x in freq\_data]

thres\_freq = self.THRESHOLD\_DICT['FREQ\_OFFSET']

if tx\_data[0]>4:

#if True: #debug

for idx in range(len(freq\_data)):

self.rpt\_append('FREQ\_OFFSET', freq\_data[idx],thres\_freq[0][idx],thres\_freq[1][idx])

self.l\_print(3,'FREQ\_OFFSET \t%d~%d\n' %(thres\_freq[0][idx], thres\_freq[1][idx]))

if int(freq\_data[idx])<thres\_freq[0][idx] or int(freq\_data[idx])>thres\_freq[1][idx]:

self.tx\_test\_res=0

self.l\_print(3,"Part failure in FREQ\_OFFSET : #%d ,%d !< %d !< %d \n\r"%(idx,thres\_freq[0][idx],int(freq\_data[idx]),thres\_freq[1][idx]))

#print "freq test failed "

#tx\_log+="Part failure in FREQ\_OFFSET : #%d ,%d !< %d !< %d \n\r"%(idx,thres\_freq[0][idx],int(freq\_data[idx]),thres\_freq[1][idx])

self.fail\_list.append("freq\_test")

#self.append\_err\_log(tx\_log)

#self.print\_dbg( tx\_log)

#tx\_log = ''

else:

self.rpt\_append('FREQ\_OFFSET','UaS-NA',thres\_freq[0][0],thres\_freq[1][0])

self.tx\_test\_res=0

self.l\_print(3,'unavailable signal')

tx\_log = ''

elif 'txp\_result' in log\_tmp:

dlist = log\_tmp.split(':')[1].split(',')[:-1]

txp\_res = [int(x) for x in dlist]

thres\_txp\_res = self.THRESHOLD\_DICT['TXP\_RES']

idx = 0 #idx=0 means txp\_res for tx index

val = txp\_res[idx]

#for idx, val in enumerate(txp\_res):

self.l\_print(3,"TXP\_RES \t%d~%d\n" %(thres\_txp\_res[0][0], thres\_txp\_res[1][0]))

self.rpt\_append('TXP\_RES\_0', val,thres\_txp\_res[0][0],thres\_txp\_res[1][0])

if val < thres\_txp\_res[0][idx] or val > thres\_txp\_res[1][idx]:

self.tx\_test\_res=0

self.l\_print(3,"Part failure in TXP\_RES[tx] : #%d ,%d !< %d !< %d \n\r"%(idx, thres\_txp\_res[0][idx], val, thres\_txp\_res[1][idx]))

self.fail\_list.append("txp\_res[tx]")

tx\_log = ''

elif 'fb\_rx\_num' in log\_tmp:

dlist = log\_tmp.split(':')[1].split(',')[:-1]

if len(dlist) > 6:

dlist = dlist[2:]

tx\_data = [ int(x) for x in dlist]

if(self.tx\_test\_res==1):

self.l\_print(0,'TX TEST OK')

self.ui\_print('TX TEST OK')

return 0

elif self.tx\_test\_res==0:

self.l\_print(0,'TX TEST FAIL')

self.ui\_print('TX TEST FAIL')

return 1

else:

self.ui\_print('TX TEST EXCEPTION')

return 1

def rf\_test\_rxtest(self, lg):

"""

Run RX packets test

Returns: err

0: success

1: fail

"""

#=============================rx test=============================================

self.l\_print(0,'start rx test')

if self.rx\_test\_res == 0:

self.rx\_test\_res = 1

if 'dut\_rx\_num' in self.THRESHOLD\_DICT.keys() and self.en\_rx\_test:

self.l\_print(0,"RX TEST BEGIN")

rx\_log=''

rssi\_log=''

log\_tx\_rx = lg[0].split('\n')

thres\_tmp = self.THRESHOLD\_DICT['dut\_rx\_num']

#self.append\_log("dut\_rx\_num \t%d~%d\n" %(thres\_tmp[0][0], thres\_tmp[0][1]))

self.l\_print(3,("dut\_rx\_num \t%d~%d\n" %(thres\_tmp[0][0], thres\_tmp[1][0])))

for log\_tmp in log\_tx\_rx[-20:]:

if "tx packet test" in log\_tmp:

self.rx\_test\_res = 1

break

#self.rx\_test\_res=1

dut\_rssi = -1

fb\_rssi = -1

for log\_tmp in log\_tx\_rx[-20:]:

#self.print\_dbg(( "log\_tmp :",log\_tmp))

#self.print\_dbg("test : in rssi test")

if 'fb\_rxrssi' in log\_tmp:

#print "test log tmp:",log\_tmp

dlist = log\_tmp.split(':')[1].split(',')[:-1]

#print "&&&&&&&&&&&&&&&&&&&&&&&&&"

#print "test dlist:",dlist

#self.print\_dbg(("test dlist fb rssi : ",dlist))

fb\_rssi\_v = [int(x) for x in dlist]

fb\_rssi\_flg = 0

if self.chip\_type == "ESP32":

print "fb\_rssi for ESP32"

fb\_rssi\_v = [ max(fb\_rssi\_v), ]

for k in range(len(fb\_rssi\_v)):

self.l\_print(0,("fb\_rxrssi \t%d~%d\n" %(self.THRESHOLD\_DICT['fb\_rxrssi'][0][k], self.THRESHOLD\_DICT['fb\_rxrssi'][1][k])))

self.rpt\_append('FB\_RXRSSI', fb\_rssi\_v[k],fb\_rssi\_v[k]<self.THRESHOLD\_DICT['fb\_rxrssi'][0][k],fb\_rssi\_v[k]<self.THRESHOLD\_DICT['fb\_rxrssi'][1][k])

if fb\_rssi\_v[k]>self.THRESHOLD\_DICT['fb\_rxrssi'][1][k] or fb\_rssi\_v[k]<self.THRESHOLD\_DICT['fb\_rxrssi'][0][k] :

self.rx\_test\_res = 0

fb\_rssi\_flg = 1

self.l\_print(3,"Part failure in FB\_RXRSSI[%d] res: %d !< %d !< %d \r\n"%(k,self.THRESHOLD\_DICT['fb\_rxrssi'][0][k],fb\_rssi\_v[k],self.THRESHOLD\_DICT['fb\_rxrssi'][1][k]))

if fb\_rssi\_flg == 1:

self.fail\_list.append("fb\_rxrssi")

fb\_rssi = int(dlist[0])

if 'dut\_rxrssi' in log\_tmp:

dlist = log\_tmp.split(':')[1].split(',')[:-1]

dut\_rssi = int(dlist[0])

dut\_rssi\_v = [int(x) for x in dlist]

dut\_rssi\_flg = 0

if self.chip\_type == "ESP32":

if DEBUG: print "dut\_rssi for ESP32"

dut\_rssi\_v = [ max(dut\_rssi\_v), ]

for k in range(len(fb\_rssi\_v)):

self.l\_print(3,("dut\_rxrssi \t%d~%d\n" %(self.THRESHOLD\_DICT['dut\_rxrssi'][0][k], self.THRESHOLD\_DICT['dut\_rxrssi'][1][k])))

self.rpt\_append('DUT\_RXRSSI',dut\_rssi\_v[k],self.THRESHOLD\_DICT['dut\_rxrssi'][0][k],self.THRESHOLD\_DICT['dut\_rxrssi'][1][k])

if dut\_rssi\_v[k]>self.THRESHOLD\_DICT['dut\_rxrssi'][1][k] or dut\_rssi\_v[k]<self.THRESHOLD\_DICT['dut\_rxrssi'][0][k] :

self.rx\_test\_res = 0

dut\_rssi\_flg = 1

self.l\_print(3,"Part failure in DUT\_RXRSSI[%d] res: %d !< %d !< %d \r\n"%(k,self.THRESHOLD\_DICT['dut\_rxrssi'][0][k],dut\_rssi\_v[k],self.THRESHOLD\_DICT['dut\_rxrssi'][1][k]))

if dut\_rssi\_flg == 1:

self.fail\_list.append("dut\_rxrssi")

self.l\_print(3,("rssi\_diff \t%d~%d\n" %(self.THRESHOLD\_DICT['rssi\_diff'][0][0], self.THRESHOLD\_DICT['rssi\_diff'][1][0])))

if (dut\_rssi-fb\_rssi)>self.THRESHOLD\_DICT['rssi\_diff'][1][0] or (dut\_rssi-fb\_rssi)<self.THRESHOLD\_DICT['rssi\_diff'][0][0] :

self.rx\_test\_res = 0

self.l\_print(3,"Part failure in RSSI res: FB: %d ; DUT: %d, %d !< %d !< %d \r\n"%(fb\_rssi,dut\_rssi,self.THRESHOLD\_DICT['rssi\_diff'][0][0],(dut\_rssi-fb\_rssi),self.THRESHOLD\_DICT['rssi\_diff'][1][0]))

#self.print\_dbg( rssi\_log)

self.fail\_list.append("rssi")

if 'txp\_result' in log\_tmp:

dlist = log\_tmp.split(':')[1].split(',')[:-1]

txp\_res = [int(x) for x in dlist]

thres\_txp\_res = self.THRESHOLD\_DICT['TXP\_RES']

idx = 1 #idx=1 means txp\_res in rx

val = txp\_res[idx]

#for idx, val in enumerate(txp\_res):

self.l\_print(3,"TXP\_RES\_1 \t%d~%d\n" %(thres\_txp\_res[0][1], thres\_txp\_res[1][1]))

self.rpt\_append('TXP\_RES\_1', val,thres\_txp\_res[0][0],thres\_txp\_res[1][0])

if val < thres\_txp\_res[0][idx] or val > thres\_txp\_res[1][idx]:

self.rx\_test\_res=0

self.l\_print(3,"Part failure in TXP\_RES[rx] : #%d ,%d !< %d !< %d \n\r"%(idx, thres\_txp\_res[0][idx], val, thres\_txp\_res[1][idx]))

self.fail\_list.append("txp\_res[rx]")

rx\_log = ''

if u'dut\_rx\_num' in log\_tmp:

dlist = log\_tmp.split(':')[1].split(',')[:-1]

if len(dlist) > 6:

dlist = dlist[2:]

rx\_data = [ int(x) for x in dlist]

if(self.rx\_test\_res):

self.l\_print(0,'RX TEST OK')

self.ui\_print('RX TEST OK')

return 0

elif(self.rx\_test\_res==0):

self.l\_print(0,'RX TEST NOK')

self.l\_print('RX TEST NOK')

return 1

else:

self.l\_print(3,'RX TEST EXCEPTION')

self.ui\_print('RX TEST EXCEPTION')

return 1

def rf\_test\_analogtest(self,lg):

if not lg[0]=='':

self.l\_print(0,lg[0])

self.data\_process(0)

if(self.ana\_test\_result):

self.l\_print(0,'analog test ok')

self.ui\_print('ANALOG TEST OK')

return 0

else:

self.l\_print(0,'analog test nok')

self.ui\_print('ANALOG TEST NOK')

return 1

def data\_process(self,block\_num):

self.l\_print(0,"Data Processing...\n")

start=time.clock()

if True:

if self.chip\_type == "ESP32":

print "ESP32:"

values\_dictlist=rl.read\_log\_data(self.logpath,'ESP32',block\_num)

else:

values\_dictlist=rl.read\_log\_data(self.logpath,'module2515',block\_num)

print "test len : ",len(values\_dictlist)

else:#debug

##debug for print

print("=============================\r\n\r\n")

print("this is only for log process debug\r\n")

print("should never be here in a formal version\r\n")

print("===============================\r\n\r\n")

value\_tmp = []

value\_tmp.append( values\_dictlist[block\_num])

\_res=rl.data\_process\_dictList\_2(value\_tmp,self.THRESHOLD\_DICT,0)

if 1:

if 'single\_chip passed' in \_res[1] or not "Part failure" in \_res[1]:

# self.print\_dbg("single chip passed...")

self.ana\_test\_result=1

self.fail\_list=[]

else:

self.ana\_test\_result=False

self.fail\_list=\_res[2]

else:

self.ana\_test\_result=\_res[0]

self.fail\_list=\_res[2]

self.t\_dataprocess=time.clock()-start

def rf\_test\_catch\_log(self):

'''

get the module self calibration and test result via uart

Return: err, logs

0: get log success

1: get log fail

'''

print self.slot\_num, "start wait"

while self.thread\_pause:

pass

start=time.clock()

self.l\_print(0,'record serial print ')

retry = False

if self.chip\_type == "ESP8089":

log=spud.get\_serial\_line\_id(self.ser,'MODULE\_TEST START!!!','req\_suc',retry = retry,chip\_type = self.chip\_type,mode=self.loadmode,wd=self) #'user code done')

elif self.chip\_type == "ESP32":

log=spud.get\_serial\_line\_id(self.ser,'MODULE\_TEST START!!!','MODULE\_TEST EDN!!!',retry = retry,chip\_type = self.chip\_type,mode=self.loadmode,wd=self) #'user code done')

else:

log=spud.get\_serial\_line\_id(self.ser,'MODULE\_TEST START!!!','MODULE\_TEST END!!!',retry = retry,chip\_type = self.chip\_type,mode=self.loadmode,wd=self) #'user code done')

print self.slot\_num, "finish RF"

self.ui\_print('[state]RFMutex')

self.mutex\_send\_flag = 1

if '' in log:

self.l\_print(3,'seria print is null')

return 1, log

return 0, log

def load\_to\_ram(self, image\_path="image/init\_data.bin"):

self.l\_print(0,'target bin is %s'%image\_path)

self.ui\_print('UART DOWNLOADing...')

self.l\_print(3,("Start UartDownload...,%s"%image\_path))

last\_time = None

dr=self.memory\_download.memory\_download(image\_path)

if dr:

return 0

else:

self.memory\_download.stopFlg=1

self.memory\_download.disconnect()

self.l\_print(0,"download disconnect...")

return 1

### GENERAL TEST -------------------------------- ###

def general\_test\_gpio(self):

if(self.chip\_type=='ESP8266'):

if self.en\_gpio\_8266\_test:

return self.general\_test\_gpio\_8266()

elif(self.chip\_type=='ESP32'):

if self.en\_gpio\_32\_test:

return self.general\_test\_gpio\_32()

return 0

def general\_test\_gpio\_8266(self):

i=0

self.gpio\_02\_test\_pin=self.testflow['GPIO\_8266\_TEST\_PIN']

self.gpio\_02\_test\_value=self.testflow['GPIO\_8266\_TEST\_VAL']

self.gpio\_02\_read\_en=int(self.testflow['GPIO\_8266\_TEST\_READ\_EN'])

self.gpio\_02\_target\_testvalue=self.testflow['GPIO\_8266\_TEST\_VAL\_TARGET']

self.l\_print(0,'start 02 gpio test')

self.msleep(200)

serTestRes = self.ser

if not serTestRes.isOpen():

serTestRes.open()

serTestRes.flush()

self.l\_print(2,'gpio test pin is:%s'%self.gpio\_02\_test\_pin)

self.l\_print(2,'gpio test value is:%s'%self.gpio\_02\_test\_value)

self.l\_print(2,'gpio test read en?%d'%self.gpio\_02\_read\_en)

self.l\_print(2,'gpio test target value is:%s'%self.gpio\_02\_target\_testvalue)

gpio\_cmd = "gpio\_test %s %s %s\n\r"%(self.gpio\_02\_test\_pin,self.gpio\_02\_test\_value,self.gpio\_02\_test\_value)

serTestRes.write(gpio\_cmd)

gpio\_log = ''

while True:

res\_line = serTestRes.readline()

if "PASS" in res\_line:

self.l\_print(3,'read res:%s'%res\_line)

self.l\_print(3,'GPIO TEST 1 PASS')

self.gpio\_test\_res1 = 1

break;

elif "FAIL" in res\_line:

self.l\_print(3,'read res:%s'%res\_line)

self.l\_print(3,'GPIO TEST 1 FAIL')

self.gpio\_test\_res = 0

break;

else:

self.l\_print(0,'step1,read gpio test return value exception,count is %d'%i)

self.gpio\_test\_res=0

i+=1

if self.gpio\_02\_read\_en == 1:

i=0

serTestRes.flushInput()

serTestRes.flushOutput()

serTestRes.write("gpio\_read\n\r")

res\_line = serTestRes.readline()

if "GPIO\_READ" in res\_line:

val\_rd = int(res\_line.strip("\r\n").split(',')[1],16)

val\_tgt = int(self.gpio\_02\_target\_testvalue,16)

self.l\_print(3,("val\_rd: {}".format(hex(val\_rd))))

self.l\_print(3,("val\_tgt: {}".format(hex(val\_tgt))))

if val\_rd&val\_tgt == val\_tgt:

self.gpio\_test\_res2=1

self.l\_print(3,("{}".format(res\_line)))

self.l\_print(0,'pass')

else:

self.l\_print(0,'fail')

self.l\_print(3,("\r\ngpio\_read1 fail:"+res\_line+";target:"+self.gpio\_02\_target\_testvalue+"\r\n"))

self.gpio\_test\_res = 0

else:

self.l\_print(3,'step2,read gpio value exception')

self.gpio\_test\_res=0

gpio\_v = int(self.gpio\_02\_test\_value,16)

gpio\_v = hex(0xffff^gpio\_v)

gpio\_cmd = "gpio\_test %s %s %s\n\r"%(self.gpio\_02\_test\_pin,gpio\_v,self.gpio\_02\_test\_value)

serTestRes.write(gpio\_cmd)

while True:

res\_line = serTestRes.readline()

if "PASS" in res\_line:

self.l\_print(3,("log:%s"%res\_line))

self.l\_print(3,'GPIO TEST 2 PASS')

self.gpio\_test\_res3 = 1

break;

elif "FAIL" in res\_line:

self.l\_print(3,(("log:%s"%res\_line)))

self.l\_print(3,'GPIO TEST 2 FAIL')

self.gpio\_test\_res = 0

break;

else:

self.gpio\_test\_res=0

self.l\_print(0,'step3,read gpio return str exception,count is %d'%i)

i+=1

if self.gpio\_02\_read\_en == 1:

serTestRes.flushInput()

serTestRes.flushOutput()

serTestRes.write("gpio\_read\n\r")

res\_line = serTestRes.readline()

if "GPIO\_READ" in res\_line:

val\_rd = int(res\_line.strip("\r\n").split(',')[1],16)

val\_tgt = int(self.window.gpio\_8266\_target\_val,16)

self.l\_print(3,("val\_rd:%x" %hex(val\_rd)))

self.l\_print(3,("val\_tgt:%x"%hex(val\_tgt)))

if val\_rd&val\_tgt == 0:

self.gpio\_test\_res4=1

self.l\_print(3,'step4 pass')

else:

self.l\_print(3,'step4 fail')

self.l\_print(3,("\r\ngpio\_read2 fail:"+res\_line+";target:"+self.gpio\_02\_target\_testvalue+"\r\n"))

self.gpio\_test\_res = 0

else:

self.l\_print(3,'step4,read gpio return value exception')

self.gpio\_test\_res=0

#serTestRes.close()

if(self.gpio\_test\_res==0):

self.l\_print(3,'gpio test fail,please check step log for err info')

else:

try:

if self.gpio\_02\_read\_en:

if(self.gpio\_test\_res1) and (self.gpio\_test\_res2) and (self.gpio\_test\_res3) and (self.gpio\_test\_res4):

self.l\_print(0,'general\_test\_gpio\_8266 test pass')

self.ui\_print('GPIO\_02 TEST PASS')

self.gpio\_test\_res=1

else:

self.l\_print(3,'gpio02 read en=0')

if(self.gpio\_test\_res1) and (self.gpio\_test\_res3):

self.l\_print(0,'general\_test\_gpio\_8266 test pass')

self.ui\_print('GPIO\_02 TEST PASS')

self.gpio\_test\_res=1

return 0

except:

self.l\_print(3,'gpio test fail,please check the which step err')

self.gpio\_test\_res=0

return 1

def general\_test\_gpio\_32(self):

self.l\_print(0,'start 32 gpio test')

self.gpio\_32\_test\_val\_0=self.testflow['GPIO\_32\_TEST\_VAL\_0']

self.gpio\_32\_test\_val\_1=self.testflow['GPIO\_32\_TEST\_VAL\_1']

self.gpio\_32\_test\_val\_2=self.testflow['GPIO\_32\_TEST\_VAL\_2']

gpio\_cmd = "ESP\_TEST\_GPIO %s %s %s\r" %(self.gpio\_32\_test\_val\_0, self.gpio\_32\_test\_val\_1, self.gpio\_32\_test\_val\_2)

self.l\_print(2,(("gpio cmd:%s"%gpio\_cmd)))

test\_log = self.\_test\_item('GPIO', gpio\_cmd)

if test\_log == []:

self.gpio\_test\_res = 0

self.l\_print(0,'general\_test\_gpio\_32 test fail,read value is null')

for i in test\_log:

if "Input result" in i:

gpio\_index = i.find('0x')

gpio\_result= i[gpio\_index:].split(',')

self.l\_print(3,("gpio\_result is:%s "%gpio\_result))

if (int(gpio\_result[0], 16) == int(self.gpio\_32\_test\_target\_0, 16)

and int(gpio\_result[1], 16) == int(self.gpio\_32\_test\_target\_1, 16)

and int(gpio\_result[2], 16) == int(self.gpio\_32\_test\_target\_2, 16)):

self.l\_print(3,'general\_test\_gpio\_32 test ok')

self.gpio\_test\_res=1

self.ui\_print('GPIO\_32 TEST OK')

break

else:

self.l\_print(3,'the return value is not equal the target value')

self.gpio\_test\_res=0

break

else:

self.l\_print('line have no keyword for general\_test\_gpio\_32 test')

self.gpio\_test\_res=0

if(self.gpio\_test\_res==0):

self.l\_print(3,'general\_test\_gpio\_32 test nok')

self.ui\_print('GPIO\_32 TEST NOK')

return 1

return 0

def \_test\_item(self, test\_name, test\_cmd, break\_str = None, timeout = None):

"""

Common method of sending a serial command and get response

"""

i=0

self.send\_count=1

#send command more times when test adc

self.l\_print(3,'test item is %s'%test\_name)

self.msleep(50)

timeout\_ori = self.ser.timeout

if timeout == None:

self.ser.timeout = 0.8

else:

self.ser.timeout = timeout

ser\_temp = self.ser

ser\_temp.baudrate = 115200

if not ser\_temp.isOpen():

ser\_temp.open()

ser\_temp.flush()

ser\_temp.flushInput()

self.l\_print(0,("%s test cmd: %s" %(test\_name, test\_cmd)))

while(i<self.send\_count):

ser\_temp.write(test\_cmd)

res\_line = []

read\_flag=0

while True:

temp = ser\_temp.readline()

if temp == '':

break

elif break\_str != None:

if break\_str in temp:

res\_line.append(temp)

self.l\_print(3,'%s:%s'%(break\_str,temp))

read\_flag=1

break

res\_line.append(temp)

self.l\_print(3,'breakstr non-inside,value:%s'%temp)

else:

res\_line.append(temp)

self.l\_print(3,'none breakstr,value:%s'%temp)

read\_flag=1

if(read\_flag==1):

self.l\_print(3,'test sucess')

break

i+=1

return res\_line

def general\_test\_fwcheck(self):

if(self.loadmode==1):

return self.general\_test\_fwcheck\_ram()

elif(self.loadmode==2):

return self.general\_test\_fwcheck\_flash()

return 1

def general\_test\_fwcheck\_flash(self):

self.l\_print(0,'start firmware check')

self.l\_print(2,'firmware check port is %s'%self.user\_fw\_download\_port)

self.l\_print(2,'firmware check baudrate is %d'%self.user\_fw\_download\_baud)

try:

self.ser.close()

except:

pass

try:

self.fwser=serial.Serial(port=self.user\_fw\_download\_port, baudrate=self.user\_fw\_download\_baud,

timeout=3)

except:

self.ui\_print('OPEN FIRMWARE CHECK PORT ERROR')

return 1

if(self.fw\_cmdEn==0):

self.l\_print(0,'fw check with no cmd')

res,data=self.\_read\_fw(ser=self.fwser,cmd\_str='',pattern=self.fw\_targetstr,ser\_tout=self.user\_fw\_download\_timeout,

delay=self.user\_fw\_download\_delay,baud=self.user\_fw\_download\_baud)

elif(self.fw\_cmdEn):

for param in self.cmd\_group:

temp\_list=[]

temp\_list=param.split(',')

cmd=temp\_list[0]

targetstr=temp\_list[1]

tout=temp\_list[2]

self.l\_print(0,'fw check with cmden=1')

res,data=self.\_read\_fw(ser=self.fwser, cmd\_str=cmd, pattern=targetstr,ser\_tout=tout,

delay=0.5,baud=self.user\_fw\_download\_baud)

self.l\_print(3,'re-send cmd')

res,data=self.\_read\_fw(ser=self.fwser, cmd\_str=cmd, pattern=targetstr,ser\_tout=tout,

delay=0.5,baud=self.user\_fw\_download\_baud)

if not res==True:

self.l\_print(3,'%s cmd check firmware error'%cmd)

break

try:

self.fwser.close()

except:

self.l\_print(0,'close fw check port error1')

if res==True:

self.l\_print(0,'firmware check ok')

self.ui\_print('FIRMWARE CHECK OK')

return 0

else:

self.l\_print(0,'firmware check nok')

return 1

def general\_test\_fwcheck\_ram(self):

try:

if(self.ser.isOpen()):

self.ser.close()

except:

self.l\_print(0,'close port error for firmware check\_\_loadmode=2')

return 1

if(self.fw\_cmdEn==0):

self.l\_print(0,'fw check with no cmd')

check\_res=fwcheck\_ramdownload.run(self.COMPORT,self.BAUDRATE,self.fw\_cmdEn,'',self.chip\_type,self.fw\_targetstr,

self.user\_fw\_download\_delay,self.user\_fw\_download\_timeout,self.user\_fw\_download\_port,self.user\_fw\_download\_baud)

elif(self.fw\_cmdEn):

for cmd,targetstr,tout in self.send\_cmd,self.fwstr\_withcmd,self.fwtmo\_withcmd:

self.l\_print(0,'fw check with cmden=1')

check\_res=fwcheck\_ramdownload.run(self.COMPORT, self.BAUDRATE, self.fw\_cmdEn,cmd, self.chip\_type,

targetstr,

0.5,

tout,

self.user\_fw\_download\_port,self.user\_fw\_download\_baud)

if not check\_res:

self.l\_print(3,'%s cmd check firmware error'%cmd)

break

if check\_res:

self.ui\_print('FIRMWARE CHECK OK')

return 0

else:

self.ui\_print('FIRMWARE CHECK ERROR')

return 1

def \_read\_fw(self,ser,cmd\_str,pattern,ser\_tout = 1,delay = 1, baud = None):

if not ser.isOpen():

ser.open()

if not baud == None:

print "set new baud: ", baud

ser.baudrate = baud

ser.timeout = 0.5

start\_time = time.time()

if not cmd\_str == '':

ser.write(cmd\_str+"\r\n")

if pattern == None:

return (True,None)

while True:

line = ser.read(1024)

self.l\_print(3,'firmware check read line:%s'%line)

if pattern.upper() in line.upper():

#ser.close()

return (True,line)

elif line == '':

ser.close()

return (False,None)

else:

pass

if delay>0:

self.msleep(int(1000\*delay))

if time.time() - start\_time >= ser\_tout:

self.l\_print(3,'read firmware version timeout')

return (False, None)

pass

### COMMON TEST -------------------------------- ###

def try\_sync(self):

'''

try sycn with chip

Returns:

0: sync success

1: fail

2: already tested

'''

self.ui\_print('[state]SYNC')

if(self.loadmode==1): # ram test mode

return self.try\_sync\_ram()

elif self.loadmode == 2: # flash test mode

rst = self.try\_sync\_flash()

try:

self.ser.close()

except:

pass

if rst < 0:

return 1

elif rst == 1:

return 2

elif rst==0:

return 0

def try\_sync\_flash(self):

try:

self.ser = serial.Serial(port=self.COMPORT, baudrate=self.BAUDRATE, timeout=0.1)

except:

self.ui\_print('open serial fail')

return -1 # open serial fail

cmd='esp\_read\_efuse\_128bit\r'

self.ser.write(cmd)

rst = self.ser.readline()

if rst.find('esp\_read\_efuse\_128bit:') >= 0:

return 0

timeout = 5

if self.dutconfig['chip\_conf']['freq'] == '26M':

self.ser.baudrate = 74880

t = time.time()

while time.time()-t < timeout:

try:

rl = self.ser.readline()

except:

return -4 # error cause read from serial

if rl.find('pass flag res:1') >= 0:

return 1 # into normal mode, and already test pass

elif rl.find('pass flag res:0') >= 0:

return -2 # into normal mode without test pass

elif rl.find('jump to run test bin') >= 0:

return 0 # into test mode

if len(rl)>1:

if DEBUG: print rl

return -3 # sync timeout

def try\_sync\_ram(self):

'''

try sycn the chip by slip cmd within 20 times

'''

sync\_res = 0

sync\_count=0

connect\_status = 1

self.memory\_download.disconnect()

try:

connect\_res = self.memory\_download.com\_connect(self.COMPORT, self.BAUDRATE)

self.ser=self.memory\_download.esp.\_port

self.l\_print(0,'conncet result is %d'%connect\_res)

except serial.SerialException:

return 1

if connect\_res:

self.memory\_download.ESP\_SET\_BOOT\_MODE(0) #try outside io control boot mode

while(sync\_count<=20):

sync\_count+=1

self.l\_print(3,("sync\_count is : %d "%sync\_count))

self.ui\_print("sync\_count is : %d "%sync\_count)

self.msleep(300) #delay to run uart\_connect() again , if shorter than 0.2, fail to connect again

try:

sync\_res = self.memory\_download.device\_sync()

self.l\_print(2,'sync res is %d'%sync\_res)

if sync\_res > 0: #should be status 1 or 2

connect\_status = 1

self.memory\_download.esp.\_port.setDTR(False)

try:

if not self.chip\_type == "ESP32" and not self.chip\_type == "ESP8089":

self.memory\_download.set\_higher\_baud(1152000)

except:

sync\_res = 0

else:

connect\_status = 0

except:

connect\_status = 0

if sync\_res == 1:

self.l\_print(0,'chip sync ok')

return 0

else:

self.l\_print(3,'chip sync Nok,re-sync again')

else:

sync\_res = 0

return 1

def check\_chip(self):

if self.loadmode == 2:

return self.check\_chip\_flash()

elif self.loadmode == 1:

return self.check\_chip\_ram()

def check\_chip\_flash(self):

'''

try check chip's mac and efuse

'''

getmac\_res=0

self.memory\_download.disconnect()

try:

self.ser = serial.Serial(port=self.COMPORT, baudrate=self.BAUDRATE,timeout=0.5)

self.l\_print(0,'serial open ok')

self.ui\_print('SER OPEN OK')

except serial.SerialException:

self.ui\_print('SERIAL PORT EXCEPTION')

return 1

try:

getmac\_res=self.memory\_download.esp\_getmac(self.ser)

except:

self.l\_print(3,'get mac error')

return 1

if getmac\_res:

self.l\_print(1,self.memory\_download.ESP\_MAC)

self.MAC=self.memory\_download.ESP\_MAC.replace('0x','').replace('-','').replace(':','')

self.l\_print(3,"mac sta: %s"%self.memory\_download.ESP\_MAC)

self.ui\_print('[mac]{}'.format(self.MAC))

self.logpath=self.esp\_gen\_log()

if(self.logpath is not ''):

return 0

else:

self.ui\_print('GEN LOG PRINT NOK')

return 1

return 1

def check\_chip\_ram(self):

if self.sub\_chip\_type == 'ESP3D2WD':

self.memory\_download.esp\_config\_spi\_mode()

if self.en\_tx\_test>0 or self.en\_rx\_test > 0:

if self.tester\_con\_flg==1:

self.ui\_print('CONNECT TESTER OK')

elif self.tester\_con\_flg==0:

self.ui\_print('CONNECT TESTER NOK')

else:

self.ui\_print('CONNECT TESTER SKIPED')

flash\_res = self.memory\_download.get\_flash\_id(self.memory\_download.esp)

if flash\_res == False:

return 1

if(not self.chip\_type == 'ESP8089'):

if(self.memory\_download.flash\_device\_id != 0):

self.l\_print(3,'flash detected')

else:

self.l\_print(3,'error,flash not detected')

self.log\_item("ERROR! FLASH NOT DETECTED!")

return 1

if self.set\_mac\_en == 0 :

pass

try:

res = self.memory\_download.get\_mac()

except:

return 1

if res == True:

self.MAC=self.memory\_download.ESP\_MAC.replace('0x','').replace('-','').replace(':','')

self.l\_print(3,"mac sta: %s"%self.memory\_download.ESP\_MAC)

self.ui\_print('[mac]{}'.format(self.MAC))

if(not self.chip\_type == 'ESP8089'):

self.flash\_manufacturer\_id=self.memory\_download.flash\_manufacturer\_id

self.flash\_dev\_id=self.memory\_download.flash\_device\_id

if self.set\_mac\_en==0:

self.l\_print("mac ap: %s"%self.memory\_download.ESP\_MAC\_AP)

elif res==False:

self.l\_print(3,"get mac failed...")

sync\_res=0

self.memory\_download.disconnect()

connect\_res=0

self.memory\_download.stopFlg=1

self.l\_print(0,'read reg failed : reset connect\_res and sync\_res..')

if self.testflow.set\_mac\_en==1:

pass

else:

self.ui\_print('GET MAC FAILED')

elif res==-1:

self.l\_print(3,"EFUSE CHECK failed...")

sync\_res=0

self.memory\_download.disconnect()

connect\_res=0

self.memory\_download.stopFlg=1

self.l\_print(0,'EFUSE CHECK ERROR...')

if self.testflow.set\_mac\_en==1:

pass

else:

self.l\_print(3,"CHIP EFUSE CHECK ERROR!!! FAIL")

self.ui\_print("CHIP EFUSE CHECK ERROR!!! FAIL")

else:

self.l\_print(3,"read reg failed...")

self.memory\_download.disconnect()

self.memory\_download.stopFlg=1

self.l\_print(3,'read reg failed : reset connect\_res and sync\_res..')

if self.window.set\_mac\_en==1:

pass

else:

self.ui\_print('READ REGISITER ERROR')

self.logpath=self.esp\_gen\_log()

try:

if self.logpath is not '':

self.l\_print(0,'log path is %s'%self.logpath)

except:

self.ui\_print('GENERAL LOG FAIL')

return 1

return 0

### OTHER FUNCTIONS -------------------------------- ###

def rpt\_append(self,item,value,thres\_l=-1000,thres\_h=1000):

testitem\_str=item

#if(eval(value)>eval(item\_thres\_h\_dict[testitem[item]])) or (eval(value)<eval(item\_thres\_l\_dict[testitem[item]])):

if(value>thres\_l) and (value<thres\_h):

res='PASS'

else:

res='FAIL'

value\_str=str(value)

spec\_l\_str=str(thres\_l)

spelc\_h\_str=str(thres\_h)

res\_str=res

tempstr=testitem\_str+','+value\_str+','+spec\_l\_str+','+spelc\_h\_str+','+res\_str+'\n'

self.rptstr=self.rptstr+tempstr

return 0

def esp\_gen\_rpt(self):

tool\_ver=self.tool\_ver

chip\_type=self.chip\_type

fac\_=self.fac\_

po=self.po

mac=self.MAC

res=self.resflag

if(res==1):

res='PASS'

else:

res='FAIL'

rptstr=self.rptstr

\_path='C:/ESP\_REPORT/'

timestr=time.strftime('%Y%m%d%H-%M-%S',time.localtime(time.time()))

try:

if(not os.path.exists(\_path)):

os.makedirs(\_path)

filename=\_path+po+'\_\_'+res+mac+'\_'+timestr+'.csv'

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-----ESP MODULE TEST REPORT-----\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'+'\r\n'

ver\_str='TEST TOOL VERSION:'+tool\_ver+'\r\n'

chip\_str='CHIP TYPE:'+chip\_type+'\r\n'

fac\_str='FACTORY:'+fac\_+'\r\n'

title=title\_str+ver\_str+chip\_str+fac\_str

rptstr=title+rptstr

with open(filename,'a') as fn:

fn.write(rptstr)

except:

print 'esp report creat error'

def l\_print(self,print\_type=0,log\_str=''):

sys.stdout=sys.stdout

esp\_logpath=self.logpath

try:

if(print\_type==0):

temp\_str=log\_str+'\r\n'

print(log\_str)

with open(esp\_logpath,'a') as fn:

fn.write(temp\_str)

elif(print\_type==1):

temp\_str='[para\_in]:'+log\_str+'\r\n'

print(log\_str)

with open(esp\_logpath,'a') as fn:

fn.write(temp\_str)

elif(print\_type==2):

temp\_str='[para\_out]:'+log\_str+'\r\n'

print(log\_str)

with open(esp\_logpath,'a') as fn:

fn.write(temp\_str)

elif(print\_type==3):

temp\_str='[debug]:'+log\_str+'\r\n'

print(log\_str)

with open(esp\_logpath,'a') as fn:

fn.write(temp\_str)

except IOError:

self.esp\_logstr=self.esp\_logstr+temp\_str

def esp\_gen\_log(self):

logpath='..//logs//'

mac=self.MAC

tool\_ver=self.tool\_ver

chip\_type=self.chip\_type

fac\_=self.fac\_

timestr=time.strftime('%Y-%m-%d-%H-%M-%S',time.localtime(time.time()))

filename=mac+timestr+'.txt'

logpath=logpath+filename

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-----ESP MODULE TEST LOG-----\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'+'\r\n'

ver\_str='TEST TOOL VERSION:'+tool\_ver+'\r\n'

chip\_str='CHIP TYPE:'+chip\_type+'\r\n'

fac\_str='FACTORY:'+fac\_+'\r\n'

title=title\_str+ver\_str+chip\_str+fac\_str

try:

with open(logpath,'a') as fn:

fn.write(title)

fn.write(self.esp\_logstr)

except:

logpath=''

return logpath

return logpath

def ui\_print(self,log):

if log.endswith('OK'):

v\_l=log.split(' ')

temp\_str=log[:len(v\_l[-1])\*-1]

re\_filling=22

fill\_num=re\_filling-len(temp\_str)

re\_fill\_str=' '

re\_fill\_str=re\_fill\_str\*fill\_num

log=temp\_str+re\_fill\_str+v\_l[-1]

self.\_stdout.write(log)

def upload\_server(self, rst):

no\_err = "0x00"

err\_params = "0x01"

err\_conn = "0x02"

err\_upload = "0x03"

err\_other = "0xff"

err\_msg = {

"err\_code" : no\_err,

"err\_info" : ''

}

up\_data = {"server\_ip":"120.76.204.21",

"server\_port":"6666",

"device\_type":str(self.dutconfig['chip\_conf']['chip\_type']),

"fw\_ver":str(self.dutconfig['common\_conf']['fw\_ver']),

"esp\_mac":str(self.MAC),

"cus\_mac":"",

"flash\_id":"",

"test\_result":rst,

"factory\_sid":str(self.dutconfig['common\_conf']['fac\_sid']),

"batch\_sid":str(self.dutconfig['common\_conf']['batch\_sid']),

"efuse":"",

"chk\_repeat\_flg":"False",

"po\_type":"0"}

uploader = upload\_to\_server.Uploader(up\_data)

try:

t\_err, rsp = uploader.client()

if t\_err != 0:

err\_msg["err\_code"] = err\_conn

err\_msg["err\_info"] = "tcp conn error"

elif rsp["status"] == 500:

err\_msg["err\_code"] = err\_upload

err\_msg["err\_info"] = rsp["message"]

elif rsp["status"] == 200:

err\_msg["err\_code"] = no\_err

err\_msg["err\_info"] = "send to server success"

else:

err\_msg["err\_code"] = err\_other

err\_msg["err\_info"] = "unknow error"

except:

err\_msg["err\_code"] = err\_conn

err\_msg["err\_info"] = "tcp conn error"

print (err\_msg)

if err\_msg['err\_code'] == '0x00':

index= str(rsp['batch\_index'])

total= str(rsp['batch\_cnt'])

self.ui\_print('[upload]'+index+'/'+total)

self.ui\_print('UPLOAD OK')

else:

self.ui\_print('[state]upload-f')

def esp\_write\_flash(self):

self.l\_print(0,'ALL TEST PASS,WRITE PASS FLAG')

cmdstr=self.time2cmdstr()

print cmdstr

try:

self.ser.flushInput()

self.ser.flushOutput()

self.msleep(100)

self.ser.write(cmdstr)

except:

self.l\_print(3,'write pass info nok1')

self.ui\_print('WRITE PASS INFO NOK1')

return 1

temp=self.ser.readline()

print temp

if 'esp\_set\_fac\_info\_pass' in temp:

self.l\_print(3,'write pass info ok2')

self.ui\_print('WRITE PASS INFO OK2')

return 0

elif 'start ok' in temp:

self.l\_print(3,'write pass info ok4')

self.ui\_print('WRITE PASS INFO OK4')

return 0

else:

self.l\_print(3,'write pass info nok3')

self.ui\_print('WRITE PASS INFO NOK3')

return 1

def reboot(self):

try:

if self.ser.isOpen():

self.ser.close()

self.msleep(100)

except:

self.l\_print(0,'close port exception')

self.ser=serial.Serial(port=self.COMPORT, baudrate=self.BAUDRATE, timeout=3)

cmd='esp\_en\_reboot\r'

self.l\_print(0,'write reboot cmd')

self.ser.write(cmd)

self.ser.write(cmd)

#if self.dutconfig['chip\_conf']['freq'] == '26M':

# self.ser.baudrate = 74880

rst = self.ser.readline()

print '1'+rst

if rst.find('pass flag res:1') >= 0:

return 0

timeout = 10

t = time.time()

while time.time()-t < timeout:

rl = self.ser.readline()

print '2'+rl

if rl.find('pass flag res:1') >= 0:

return 0

elif rl.find('pass flag res:0') >= 0:

return -1

elif rl.find('jump to run test bin') >= 0:

return -2

print rl

return -3

def time2cmdstr(self):

timestr=time.strftime('%Y%m%d%H%M%S',time.localtime(time.time()))

facstr='11'

cmdstr=''

temp=facstr+'-'+timestr

for i in temp:

if i=='-':

i='0x2d'

else:

i='0x3'+i

cmdstr+=i+' '

cmdstr=cmdstr+'\r'

cmdstr='esp\_set\_flash\_test\_pass\_info'+' '+cmdstr

return cmdstr

def STOPTEST(self):

if self.thread\_pause==0:

self.thread\_pause = 1 # for rf test mutex

if self.resflag == 1:

print self.slot\_num, "finish RF"

self.ui\_print('[state]RFMutex') # for rf test mutex

self.ui\_print('[state]fail\_record')

if self.position == 'cloud':

self.upload\_server('fail')

else:

self.ui\_print('[state]pass\_record')

if self.position == 'cloud':

self.upload\_server('success')

try:

self.ser.close()

except:

self.l\_print(0,'close port error1')

if self.resflag == 1:

self.esp\_gen\_rpt()

self.ui\_print('[state]fail')

elif self.resflag==0:

self.l\_print(0,'all item test passed')

self.ui\_print('[state]pass')

self.esp\_gen\_rpt()

try:

self.ser.close()

except:

self.l\_print(0,'close port error2')

elif self.resflag==2:

self.l\_print(0,'already passed module')

self.ui\_print('[state]passed')

def stopthread(self):

self.ui\_print('[state]finish btn up')

self.ui\_print('[state]idle')

if self.thread\_pause==0:

self.thread\_pause = 1 # for rf test mutex

if self.mutex\_send\_flag == 0:

print self.slot\_num, "finish RF"

self.ui\_print('[state]RFMutex') # for rf test mutex

self.SIGNAL\_RESUME.emit()

self.msleep(200)

try:

self.ser.close()

except:

pass

self.terminate()

def thread\_resume(self):

self.thread\_pause=0

def ui\_stop(self):

self.stopthread()

def set\_params(self,stdout\_='',dutconfig='',testflow=''):

try:

self.send\_cmd=[]

self.fwstr\_withcmd=[]

self.fwtmo\_withcmd=[]

self.cmd\_group=[]

self.testflow=testflow

self.dutconfig=dutconfig

self.\_stdout=stdout\_

self.slot\_num =self.dutconfig['common\_conf']['dut\_num']

self.COMPORT=self.dutconfig['DUT'+self.slot\_num]['port1']

self.BAUDRATE=int(self.dutconfig['DUT'+self.slot\_num]['rate1'])

self.chip\_type=self.dutconfig['chip\_conf']['chip\_type']

self.sub\_chip\_type=''

self.autostartEn=int(self.dutconfig['common\_conf']['auto\_start'])

self.loadmode=self.dutconfig['common\_conf']['test\_from']

self.efusemode=self.dutconfig['chip\_conf']['efuse\_mode']

self.user\_fw\_download\_port=self.dutconfig['DUT'+self.slot\_num]['port2']

self.user\_fw\_download\_baud=int(self.dutconfig['DUT'+self.slot\_num]['rate2'])

self.user\_fw\_download\_delay=int(self.testflow['USER\_FW\_VER\_DELAY(s)'])

self.user\_fw\_download\_timeout=int(self.testflow['USER\_FW\_VER\_TIMEOUT(s)'])

self.IMGPATH=self.dutconfig['common\_conf']['bin\_path']

self.fac\_=self.dutconfig['common\_conf']['fac\_sid']

self.po=self.dutconfig['common\_conf']['po\_no']

self.position=self.dutconfig['common\_conf']['position']

self.en\_user\_fw\_check=int(self.testflow['USER\_FW\_CHECK'])

self.fw\_targetstr=self.testflow['USER\_FW\_VER\_STR']

self.fw\_cmd\_combin=self.testflow['USER\_TEST\_CMD<cmd,rsp,tmo>']

except:

self.l\_print(1,'read param error')

self.param\_read=0 #check it before start test

try:

cmd\_list=self.fw\_cmd\_combin.replace('<',';').replace('>',';').strip(';').split(';;;')

for cmd in cmd\_list:

t\_l=cmd.replace('"',',').split(',')

t\_str=t\_l[0]+','+t\_l[2]+','+t\_l[4]

self.cmd\_group.append(t\_str)

except:

self.l\_print(3,'read fw check cmd and fw target str error,if fw no need cmd write,ignore it')

if self.fw\_targetstr.upper()=='ESPCMD\_EN':

self.fw\_cmdEn=1

else:

self.fw\_cmdEn=0

if(self.loadmode=='RAM'):

self.loadmode=1

elif(self.loadmode=='FLASH'):

self.loadmode=2

try:

if(self.chip\_type=='ESP-WROOM-02'):

self.chip\_type='ESP8266'

elif self.chip\_type in ('ESP-WROOM-32',"ESP32-WROVER"):

self.chip\_type='ESP32'

except:

self.l\_print(0,'get chip type error')

self.en\_tx\_test=1 if int(self.testflow['TX'])>0 else 0

self.en\_rx\_test=1 if int(self.testflow['RX'])>0 else 0

self.en\_analog\_test=1 if int(self.testflow['ANALOG'])>0 else 0

self.en\_gpio\_8266\_test=1 if int(self.testflow['GPIO\_8266\_TEST'])>0 else 0

self.en\_gpio\_32\_test=1 if int(self.testflow['GPIO\_32\_TEST'])>0 else 0

def check\_param(self):

#check image avaiable

if(self.loadmode==1):

if not os.path.exists(self.IMGPATH):

self.l\_print(1,'IAMGE un-avaiable')

return 1

return 0

if \_\_name\_\_=='\_\_main':

pass