PROGRAMMER’S GUIDE

ThunderBERT Programmer’s Guide

Comprehensive API Documentation for MultiLane BERT Automation

Supported API Release:
Release 1.6.0

Document Version:
Version 1.5.0

Innovation for the next generation

CONTENTS

Introduction

This document serves as a programmer's guide for MultiLane Bit Error Rate Tester users focused on automation environments. It introduces and fully documents the ThunderBERT API library, a brand-new BERT architecture which significantly improves measurement analysis capabilities and optimizes test time.

The ThunderBERT API is a single library which unifies a wide family of MultiLane BERT platforms which possess different features. While most API functions will be applicable to the entire flagship MultiLane BERT family, certain functions are associated with unique BERT platforms and are not applicable to BERT's without those feature sets. This distinction between supported functions for each BERT are covered in a separate document titled "BERT Family User Guide" which can be read at multilaneinc.com. The separate document covers the bring up, installation, and navigation of the ThunderBERT Graphical User Interface.

In order to deliver a complete walkthrough to a developer audience who is leveraging ThunderBERT API towards their automation needs, this document contains high-level test flows of typical measurement sequences for various BERTs, associated Python sample code, and detailed definitions of all supported functions. Please email fae@multilaneinc.com with any questions that may arise.

Document Version Control

Version	Publication Date	Description
1.0.0	November 24th, 2020	Initial API documentation release for ML4039B, ML4039D, ML4079D, ML4039E, ML4039EN and ML4079E
1.1.0	February 1st, 2021	Added support for ML4054B and QDD Adapter
1.1.1	March 10th, 2021	Use applyConfig parameter to optimize the configuration of the BERT
1.2.0	May 19th, 2021	Added FEC test Flow for ML4054B
1.3.0	September 15th, 2021	Added clock file generator library
1.4.0	November 11th , 2021	Added support for ML4039E-ATE
1.4.1	June 7th, 2022	Changed Line Rate Range for ML4039B
1.5.0	September 4th, 2023	Added Noise and Jitter Flow. Added support for ML4079EN. Added additional error handling codes.

Supported BERTs

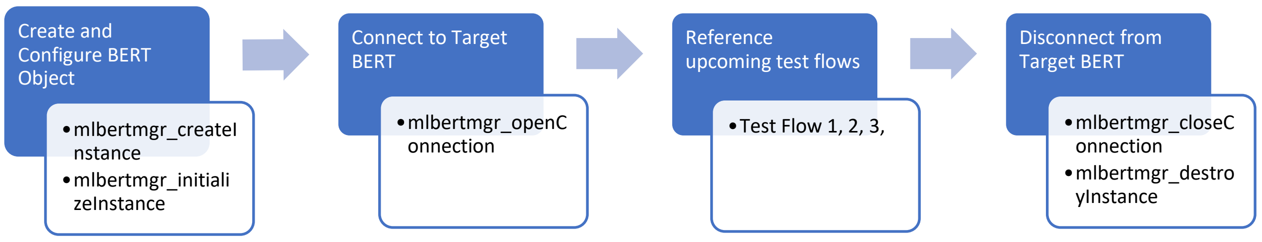
BERT PN	FW Versions Supported
ML4039B	1.0 and 1.1
ML4054B	1.1 and 1.2
QDD Adapter	1.3
ML4039D	1.1
ML4079D	1.1
ML4039E	1.6
ML4039EN	1.4
ML4079E	1.2
ML4039E-ATE	1.0
ML4079EN	1.1

General Flows

This section of the document introduces various typical sequences that a user is likely to implement when automating various MultiLane BERT platforms. Each test flow covers a unique BERT and series of tasks. The flows are characterized by specific actions, the associated API functions, and detailed Python sample code.

Main Flow: Connection and Configuration Basics

Description: This flow creates and configures a BERT instance object. Then, it requires the user to input the IP address of the target BERT device in order to establish a connection. This is the main flow and other test flows come after. Finally, after the test flows are done, the user disconnects from the target BERT Device.

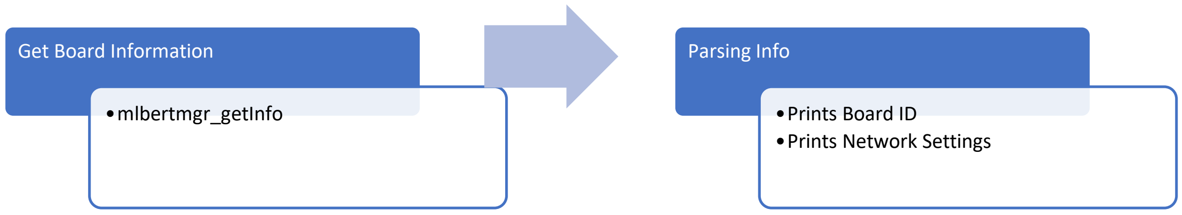

Create and Configure BERT Object
• mlbertmgr_createInstance
• mlbertmgr_initializeInstance
Connect to Target BERT
• mlbertmgr_openConnection
Reference upcoming test flows
• Test Flow 1, 2, 3,
Disconnect from Target BERT
• mlbertmgr_closeConnection
• mlbertmgr_destroyInstance

Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
import sys      # sys.maxsize
import os       # os.path
import ctypes   # DLL types marshalling
import time     # time.sleep
from pymlbertapi import pymlbertmgr

#Main Flow: Connection and Configuration Basics
def main():

    """Main function."""
    #creates Instance
    mlbert = pymlbertmgr.mlbertmgr()
    try:
        NB_CHANNELS = 4

        # Connects to device before initializing the instance
        # Edit IPADDRESS of your Instance
        IPADDRESS = "172.16.201.31"
        SUCCESS =  mlbert.mlbertmgr_openConnection(IPADDRESS)
        if SUCCESS!= pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
            raise Exception("Failed to connect to %s!" % IPADDRESS, " : ",
SUCCESS)
        print ("Connected")

        #Initialises instance
        SAVE_CONFIG = ''
        SAVE_BATHTUB = ''
        SAVE_EYE = ''
        SAVE_BATHTUB_ENABLE = 0
        SAVE_EYE_ENABLE = 0
        T_PARAMS = pymlbertmgr.InstanceParams(SAVE_CONFIG, SAVE_BATHTUB, 
SAVE_EYE, SAVE_BATHTUB_ENABLE, SAVE_EYE_ENABLE)
        SUCCESS =  mlbert.mlbertmgr_initializeInstance(T_PARAMS)
        if SUCCESS!= pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
            raise Exception("Failed to initialize Instance ! : ", SUCCESS)
        print ("Instance initialized")
        mlbert.mlbertmgr_enableMonitor(0) #Turn off all monitor flags

    finally:
        #Disconnect
        print("mlbertmgr_closeConnection: ", 
mlbert.mlbertmgr_closeConnection())
        #Destroy Instance
        mlbert.mlbertmgr_destroyInstance()
        print("mlbertmgr_destroyInstance done.")

if __name__ == "__main__":
    main()
```

Test Flow 1: Get Board Information of Any BERT

Description: After connecting to the BERT device, the user could retrieve the board information using the get_Info function. This method prints the board ID, SN, revisions and network settings (IP, MAC, Gateway).

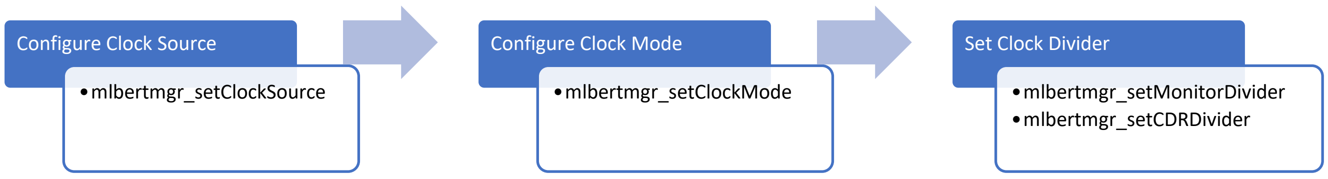


Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
#Get Board Info
INFO = ctypes.pointer(pymlbertmgr.Board_Info())
SUCCESS = mlbert.mlbertmgr_getInfo(INFO)
if SUCCESS!= pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to retrieve Board Info! : ", SUCCESS)
print ("Board Info is: ")


#Print out Board Info.
for fields in INFO[0]._fields_:
    if((fields[0] == 'ipAddress') | (fields[0] == 'Mask') | (fields[0] == 
'Gateway')):
        hexadr = getattr(INFO[0], fields[0])
        stradr = '' + str(( hexadr & 0xff))
        for i in range(3):
            hexadr = hexadr >> 8
            stradr = str((hexadr & 0xff)) + '.' + stradr
        print('\t', fields[0], ': ', stradr)

    elif(fields[0] == 'MAC'):
        hexadr = getattr(INFO[0], fields[0])
        stradr = '' + str(hex(hexadr & 0xff)[2:])
        for i in range(5):
            hexadr = hexadr>>8
            stradr = str(hex(hexadr & 0xff) [2:] + '-' + stradr)
        print('\t', fields[0], ': ', stradr)

    elif(fields[0] == 'SN'):
        SNSTR = ''
        for i in range(10):
            SNSTR = SNSTR + str(getattr(INFO[0], fields[0])[i])
        print('\t', fields[0], ': ', SNSTR)
    elif((fields[0] == 'HWRev' ) | (fields[0] == 'FWRev')):
        hexadr = getattr(INFO[0], fields[0])
        print('\t', fields[0], ': ', hexadr>>8, ".", hexadr&0xf)
    elif(fields[0] == 'adapterType'):
        if(INFO[0].isAdapterMode == True):
            print('\t', fields[0], ': ',
pymlbertmgr.ADAPTER_TYPE(getattr(INFO[0], fields[0])))
    else:
        print('\t', fields[0], ': ', getattr(INFO[0], fields[0]))
```

Test Flow 2: Configure Clock Settings

Description: The clock settings configuration is divided into three parts: clock source, mode and divider configuration. First, the user sets the clock source (Internal and External). Then, the user sets the clock output mode (Ref Clk, Monitor, External and CDR). Finally, the user sets the monitor or CDR dividers if used.



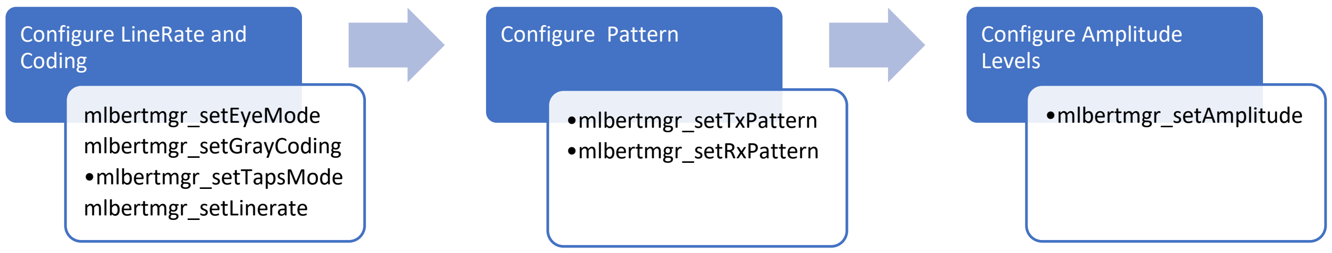
Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
APPLYCONFIG = False     # Configurations are cashed in the instrument's
memory. Enable APPLYCONFIG for the last call of the flow to trigger the
configuration of the instrument

# Edit parameters for your instance
# Clock Source
CLOCKSOURCE = pymlbertmgr.BERTMGR_CLOCKSOURCE.BERTMGR_INTERNALCLKSRC
# Output Clock Mode
CLOCKMODE = pymlbertmgr.BERTMGR_CLOCKMODE.BERTMGR_REFCLK
# Monitor Divider
```

```
DIVIDER = pymlbertmgr.BERTMGR_MONITORDIVIDER.BERTMGR_MONITOR_DIV4
# CDR Divider
CDRDIVIDER = pymlbertmgr.BERTMGR_CDRDIVIDER.BERTMGR_CDR_DIV32

#Set ClockSource
SUCCESS =  mlbert.mlbertmgr_setClockSource(CLOCKSOURCE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set ClockSource! : ", SUCCESS)
print ("ClockSource is set !")

#Set ClockMode
SUCCESS =  mlbert.mlbertmgr_setClockMode(CLOCKMODE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set ClockMode! : ", SUCCESS)
print ("ClockMode is set !")

#Clock Divider
#Set Monitor Divider
SUCCESS =  mlbert.mlbertmgr_setMonitorDivider(DIVIDER, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set Monitor Divider! : ", SUCCESS)
print ("Monitor Divider is set !")

#Set CDR Divider. Check the table of features for compatibility with the
BERT
SUCCESS =  mlbert.mlbertmgr_setCDRDivider(CDRDIVIDER, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set CDR Divider! : ", SUCCESS)
print ("CDR Divider is set !")
```

Test Flow 3: Configure Line Rate, Coding and Amplitude Levels

Description: In this flow, the user configures the linerate, coding and amplitude levels (mV) using the setter functions below.



Python Sample Code:
Environment: Python 3.8.5
Python Wrapper

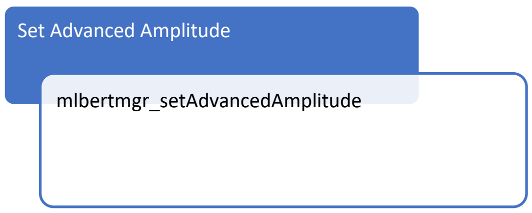
```
APPLYCONFIG = False     # Configurations are cashed in the instrument's
memory. Enable APPLYCONFIG for the last call of the flow to trigger the
configuration of the instrument

# Eye Mode
EYEMODE  = pymlbertmgr.BERTMGR_SIGMODULATION.BERTMGR_NRZ;
# Tx Taps Mode
TAPSMODE = pymlbertmgr.BERTMGR_TAPSMODE.BERTMGR_3TAPS
```

```python
# 0-Based Index of Channel (i.e. Channel 1 -> 0)
CHANNEL = 0
# FEC mode
FECMODE = pymlbertmgr.BERTMGR_FECMODE.BERTMGR_FECDISABLED
#FEC PATTERN
FECPATTERN = pymlbertmgr.BERTMGR_FECPATTERN.FECPATTERN_DISABLED
# Line Rate (Gbaud)
LINERATE = 29
# Creates PatternConfig initial struct
TXPATTERN = pymlbertmgr.PatternConfig()
# Tx Pattern
TXPATTERN.pattern = pymlbertmgr.BERTMGR_PATTERNTYPE.BERTMGR_PRBS7
# Tx Invertion
TXPATTERN.invert = False
#Tx Repetition. Reserved for user Defined Tx Pattern
TXPATTERN.repetition = 0
# Creates PatternConfig initial struct
RXPATTERN=pymlbertmgr.PatternConfig()
# Rx pattern
RXPATTERN.pattern = pymlbertmgr.BERTMGR_PATTERNTYPE.BERTMGR_PRBS7
# Rx Inversion
RXPATTERN.invert = False
# Amplitude Level mV
AMPLITUDE = 200
#Tx Pattern
#TXPATTERN.pattern = pymlbertmgr.BERTMGR_PATTERNTYPE.BERTMGR_USERDEFINED
#USER_DEFIGNED_PATTERN=  pymlbertmgr.UserDefinedPatternDefinition()
#USER_DEFIGNED_PATTERN.Pattern1.Pattern=0XAAAAFFFF55550000
#USER_DEFIGNED_PATTERN.Pattern1.Repetition=1
#USER_DEFIGNED_PATTERN.Pattern2.Pattern=0XFFFF0000FFFF0000
#USER_DEFIGNED_PATTERN.Pattern2.Repetition=0


#Set Linerate
SUCCESS =  mlbert.mlbertmgr_setLinerate(LINERATE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set Linerate! : ", SUCCESS)
print ("Linerate is set !")

# set EyeMode
SUCCESS =  mlbert.mlbertmgr_setEyeMode(EYEMODE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set EyeMode! : ", SUCCESS)
print ("EyeMode is set !")

"""# Enable Gray Coding. Applied for PAM4 Eye Mode.
ENABLE = True
SUCCESS =  mlbert.mlbertmgr_setGrayCoding(ENABLE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set Gray Coding! : ", SUCCESS)
print ("Gray Coding is set !")"""


APPLYCONFIG = True    # Trigger the configuration of all the applied
settings
# Set Taps Mode
SUCCESS =  mlbert.mlbertmgr_setTapsMode(TAPSMODE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set Taps Mode! : ", SUCCESS)
print ("Taps Mode is set !")
```

```
# Set FEC Mode. Check The table of features for compatibility
#SUCCESS =  mlbert.mlbertmgr_setFECMode(FECMODE, FECPATTERN, APPLYCONFIG)
#if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
#    raise Exception("Failed to set FEC Mode! : ", SUCCESS)
#print ("FEC Mode is set !")


for channel in range(NB_CHANNELS):
    #Set Tx Pattern
    SUCCESS =  mlbert.mlbertmgr_setTxPattern(channel, TXPATTERN,
APPLYCONFIG)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set Tx Pattern! : ", SUCCESS)
    print ("Tx Patternset is set !")

    #Set Rx Pattern
    SUCCESS =  mlbert.mlbertmgr_setRxPattern(channel, RXPATTERN,
APPLYCONFIG)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set Rx Pattern! : ", SUCCESS)
    print ("Rx Pattern is set !")

    #Set Calibrated Amplitude level. This function requires a calibrated
Instrument
    SUCCESS =  mlbert.mlbertmgr_setAmplitude(channel, AMPLITUDE,
APPLYCONFIG )
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set Amplitude Level! : ", SUCCESS)
    print ("Amplitude Level is set !")

      #set User Defined Pattern
      #SUCCESS =  mlbert.mlbertmgr_setUserDefinedPattern(channel,
      USER_DEFIGNED_PATTERN, APPLYCONFIG)
      #if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
      #      raise Exception("Failed to set User Defined Pattern! : ",
      SUCCESS)
      #print ("User Defined Pattern is set!")
```

Test Flow 4: Set Advanced Amplitude

Description: In this flow, the user sets the advanced amplitude settings.



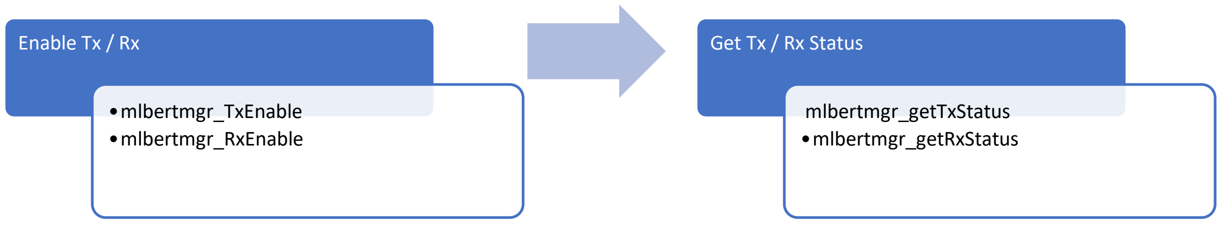
Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
# Edit parameters for your instance
# Advanced Amplitude
```

```
APROXAMPLITUDE = ctypes.pointer(ctypes.c_int(0))
ADVANCEDAMPLITUDE = pymlbertmgr.AdvancedAmplitude()
# Main Tap Value (-1000 to +1000)
ADVANCEDAMPLITUDE.mainTap = ctypes.c_int(1000)
# Post-emphasis Value (-1000 to +1000)
ADVANCEDAMPLITUDE.postEmphasis = ctypes.c_int(0)
# Pre-emphasis Value (-1000 to +1000)
ADVANCEDAMPLITUDE.preEmphasis = ctypes.c_int(0)
# Inner Eye level(500 to 1500). Applied for PAM4
ADVANCEDAMPLITUDE.innerLevel = ctypes.c_int(1000)
# Outer Eye level (1500 to 2500). Applied to PAM4
ADVANCEDAMPLITUDE.outerLevel = ctypes.c_int(2000)
# Scaling Level Percentage (70, 80, 90, 100, 110, 120)
ADVANCEDAMPLITUDE.scalingLevel = ctypes.c_int(80)
# 7-Taps Mode
for i in range(7):
    ADVANCEDAMPLITUDE.advancedTaps[i] = ctypes.c_int(0)

for channel in range(NB_CHANNELS):# set advanced Amplitude on CHANNEL
#Set Advanced Amplitude
    SUCCESS =  mlbert.mlbertmgr_setAdvancedAmplitude(channel,
                                                     ADVANCEDAMPLITUDE,
                                                     APROXAMPLITUDE,
                                                     APPLYCONFIG)

    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set Advanced Amplitude! : ", SUCCESS)
    print("APROX AMPLITUDE = ", APROXAMPLITUDE[0])
```

Test Flow 5: Enable – Disable Tx, Rx

Description: In this flow, the user sets Tx and Rx status.



Python Sample Code:

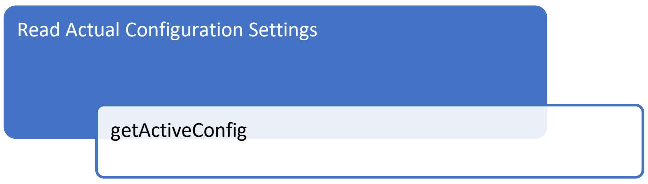
Environment: Python 3.8.5

Python wrapper

```
for channel in range(NB_CHANNELS):
    # Edit parameters for your instance
    ISENABLED = ctypes.pointer(ctypes.c_bool(False))
    STATUS = True
    # Enable  Tx
    SUCCESS =  mlbert.mlbertmgr_TxEnable(channel, STATUS)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to Enable Tx! :", SUCCESS)
    print ("Tx Enabled !")
    # Enable  Rx
    SUCCESS =  mlbert.mlbertmgr_RxEnable(channel, STATUS)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception( "Failed to Enable Rx! : ", SUCCESS)
    print ("Rx Enabled !")

    # Get Tx Status
    SUCCESS =  mlbert.mlbertmgr_getTxStatus(channel, ISENABLED)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to  Get Tx Status! : ", SUCCESS)
    print ("TX enable status : ", ISENABLED[0])

    # Get Rx Status
    SUCCESS =  mlbert.mlbertmgr_getRxStatus(channel, ISENABLED)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to  Get Rx Status! : ", SUCCESS)
    print ("RX enable status : " , ISENABLED[0])
```

Test Flow 6: Get BERT Configuration Settings

Description: In this flow, the user gets the applied configuration settings from the BERT.

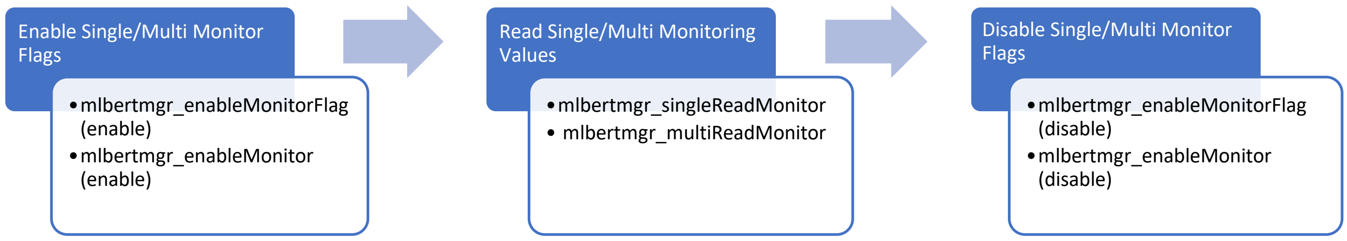


Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
CONFIG = ctypes.pointer(pymlbertmgr.ConfigurationSettings())
# Getting active BERT configuration settings
SUCCESS =  mlbert.mlbertmgr_getActiveConfig(CONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to get configuration",
pymlbertmgr.BERTMGR_STATUS(SUCCESS))
print("Get Active Configuration Done!")
```

Test Flow 7: Monitor BERT Functions

Description: In this flow, the user can either enable a single monitor flag or all monitor flags. Then, the user reads the value(s) of the enabled flag(s). The instrument has a 10s Time-Out if there is no activity on the monitoring process and therefore it must be disabled when done.



Enable Single/Multi Monitor Flags
• mlbertmgr_enableMonitorFlag (enable)
• mlbertmgr_enableMonitor (enable)
Read Single/Multi Monitoring Values
• mlbertmgr_singleReadMonitor
• mlbertmgr_multiReadMonitor
Disable Single/Multi Monitor Flags
• mlbertmgr_enableMonitorFlag (disable)
• mlbertmgr_enableMonitor (disable)

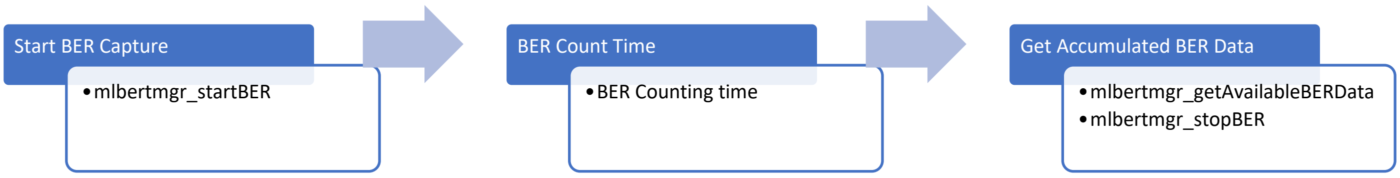
Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
# First method: Single read monitor flags.
# Reads BERT Temperature flags
SINGLEMONITORFLAG =
pymlbertmgr.BERTMGR_MONITOR_FLAGS.BERTMGR_MONITOR_TEMPERATURE
# Temperature monitor requires 4 x ushort. Refer to the documentation for
the required memory allocation per flag
SINGLE_MONITOR = (ctypes.c_ushort * NB_CHANNELS)()
# Enable Single Monitor Flag and sleep for 350 ms before starting monitor
reading.
# It is recommended to Enable the Monitor at the beginning of the main flow
to avoid any settling time.
time.sleep(0.35)
# Enable Single Monitor Flag
Enabled = True
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(SINGLEMONITORFLAG, Enabled)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Enable Single Monitor Flag: ", SUCCESS)
print("Single Monitor Flag is Enabled !")
# Single Read Monitor Flag
SUCCESS = mlbert.mlbertmgr_singleReadMonitor(SINGLEMONITORFLAG,
SINGLE_MONITOR)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Read Single Monitor Flag: ", SUCCESS)
print("single Read Monitor is Done !")
# Disable Single Monitor Flag
Enabled = False
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(SINGLEMONITORFLAG, Enabled)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Disable Single Monitor Flag: ", SUCCESS)
print("Single Monitor Flag is Disabled !")

# Second method: MultiRead monitor flags
# Refer to MONITOR_FLAGS Enum for bits order. Set to 1023 to enable all
monitor flags
MULTIMONITORFLAGS = 1023
# Monitor multiple Flags (e.g 200) following the same order of the
MONITOR_FLAGS Enum
MULTI_MONITOR = ctypes.pointer(((ctypes.c_ushort * 2) * 200)())
# Enable Multi Monitor Flags
SUCCESS = mlbert.mlbertmgr_enableMonitor(MULTIMONITORFLAGS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Enable Multi Monitor Flags ! : ", SUCCESS)
print("Multi Monitor Flags are Enabled !")
# Wait for Monitor Accumulation.
time.sleep(0.35)

# Multi-Read Monitor
SUCCESS = mlbert.mlbertmgr_multiReadMonitor(MULTIMONITORFLAGS,
MULTI_MONITOR)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Read Multi Monitor! : ", SUCCESS)
print("Multi Read Monitor is done!")

# Disable Monitor Flags
MULTIMONITORFLAGS = 0
```

```
SUCCESS = mlbert.mlbertmgr_enableMonitor(MULTIMONITORFLAGS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to  Disable Monitor! : ", SUCCESS)
print("Monitor Flags are Disabled!")
```

Test Flow 8: Execute Fundamental BER Test

Description: This flow runs a fundamental BER test.

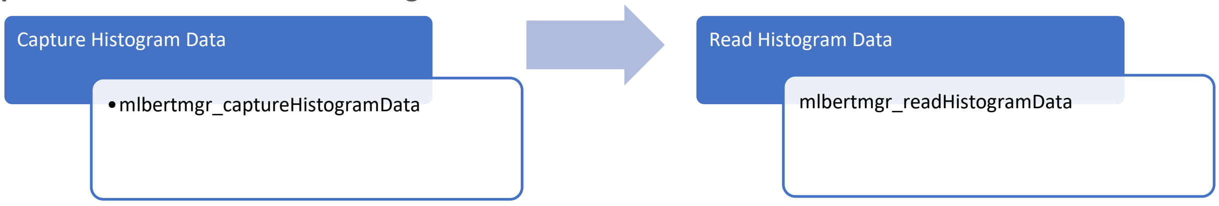


Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
# Pre-allocate MEASBERDATA Struct
MEASBERDATA = (pymlbertmgr.MeasurementsData * 1024)()
DATACOUNT = ctypes.pointer(ctypes.c_int(0))
NB_BER_CHANNELS = 4
# Enable BER Data Accumulation. Otherwise, the latest Data is captured
ACCUMULATE = False
# BER Enbaled CHANNELS. First Channel is Enabled
BERENABLEDCH = 0b00000001
VALUE = (ctypes.c_ushort * NB_BER_CHANNELS)(0)
# Before starting the BER accumulation, it is recommended to add a settling
time of 2 seconds
# ML4054B requires 5 seconds after the configuration
# ML4079EN requires 10 seconds after the configuration
# "pymlbertmgr.getConfigStatus()" will be implemented in a future library
release to check the instrument configuration status and avoid adding a
sleep time in the application script
time.sleep(3)  ## Ensure stabilization time after the BERT configuration

# Initialize Rx Lock Status and Monitor Rx Lock Status
SUCCESS = pymlbertmgr.BERTMGR_STATUS.BERTMGR_FAILED
# Call Rx lock Status in a while loop
RETRY = 20
# initialize Rx Lock Monitor Flag
SINGLEMONITORFLAG =
pymlbertmgr.BERTMGR_MONITOR_FLAGS.BERTMGR_MONITOR_RXLOCK

SUCCESS = mlbert.mlbertmgr_enableMonitor(SINGLEMONITORFLAG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to  Enable Monitor! : ", SUCCESS)
print("Monitor Flags are Enabled!")
# Wait for Monitor Data accumulation
time.sleep(0.35)
for channel in range(NB_BER_CHANNELS):
    while (VALUE[CHANNEL] == 0 and RETRY > 0):
        time.sleep(0.1)  # Sleep for 100 ms
        # Single Read Monitor of Rx Lock Status
        SUCCESS = mlbert.mlbertmgr_singleReadMonitor(SINGLEMONITORFLAG,
VALUE)
        if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
            raise Exception("Failed to Read single Monitor! : ", SUCCESS)
        RETRY -= 1
    if VALUE[channel] == 1:
        print("Rx ", channel, " is  locked!")
    else:
        print("Rx ", channel, " is not locked!")

# Disable Monitor Flags
MONITORFLAGS = 0
SUCCESS = mlbert.mlbertmgr_enableMonitor(MONITORFLAGS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to  Disable Monitor! : ", SUCCESS)
print("Monitor Flags are Disabled!")

# Start BER. This function requires Rx Lock.
mlbert.mlbertmgr_startBER(BERENABLEDCH, ACCUMULATE)
# BER Counting Time
# ML4054 BER Accumulation starts 4 seconds after enabling the BER process.
time.sleep(5)

# Get Available Data
SUCCESS = mlbert.mlbertmgr_getAvailableBERData(MEASBERDATA, DATACOUNT)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Get Available Data!", SUCCESS)

# print Out BER Data. Check MeasurementsData struct for more details
print("Datacount: ", DATACOUNT[0])
print("Measured BER Data : \r")
print("\tIs BER Enabled : ", MEASBERDATA[DATACOUNT[0] - 1].berData.enabled)
for channel in range(NB_BER_CHANNELS):
    print("\nchannel ", channel)
    print("\tEnabled Channels : ", MEASBERDATA[DATACOUNT[0] -
1].berData.enabledChannels[channel])
    print("\tLocked Channels : ", MEASBERDATA[DATACOUNT[0] -
1].berData.lockedChannels[channel])
    print("\tBER Capture Time : ", MEASBERDATA[DATACOUNT[0] -
1].berData.Time[channel])
    print("\tBit Count : ", MEASBERDATA[DATACOUNT[0] -
1].berData.BitCount[channel])
    print("\tErrorCount_MSB: ", MEASBERDATA[DATACOUNT[0] -
1].berData.ErrorCount_MSB[channel])
    print("\tErrorCount_LSB: ", MEASBERDATA[DATACOUNT[0] -
1].berData.ErrorCount_LSB[channel])
    print("\tErrorCount : ", MEASBERDATA[DATACOUNT[0] -
1].berData.ErrorCount[channel])
    print("\tAccumulatedErrorCount_MSB: ",
          MEASBERDATA[DATACOUNT[0] -
1].berData.AccumulatedErrorCount_MSB[channel])
    print("\tBER_MSB_Interval: ", MEASBERDATA[DATACOUNT[0] -
1].berData.BER_MSB_Interval[channel])
    print("\tBER_MSB_Realtime: ", MEASBERDATA[DATACOUNT[0] -
1].berData.BER_MSB_Realtime[channel])
    print("\tAccumulatedErrorCount_LSB: ",
          MEASBERDATA[DATACOUNT[0] -
1].berData.AccumulatedErrorCount_LSB[channel])
    print("\tBER_LSB_Interval: ", MEASBERDATA[DATACOUNT[0] -
1].berData.BER_LSB_Interval[channel])
    print("\tBER_LSB_Realtime: ", MEASBERDATA[DATACOUNT[0] -
1].berData.AccumulatedErrorCount_LSB[channel])
    print("\tAccumulatedErrorCount  : ", MEASBERDATA[DATACOUNT[0] -
1].berData.AccumulatedErrorCount[channel])
    print("\tBER_Interval: ", MEASBERDATA[DATACOUNT[0] -
1].berData.BER_Interval[channel])
    print("\tBER Realtime : ", MEASBERDATA[DATACOUNT[0] -
1].berData.BER_Realtime[channel])
# Stop BER
SUCCESS = mlbert.mlbertmgr_stopBER()
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Stop BER Test!", SUCCESS)
print("BER Test stopped!")
```

Test Flow 9: Read Histogram Data

Description: This flow reads the histogram data

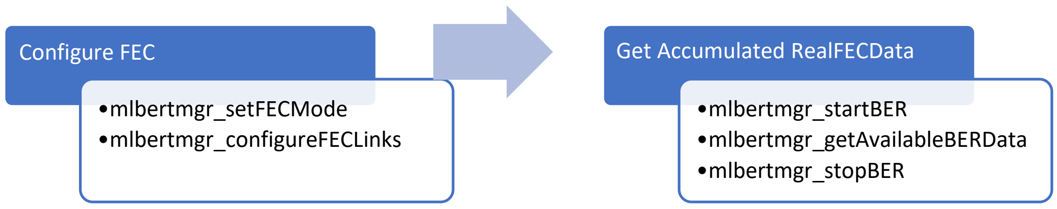


Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
# Edit parameters for your instance
# Enabled channel flags (1 bit/channel)
HISTENABLEDCHANNEL = 0b00000001
HIST = (pymlbertmgr.HistogramData * NB_CHANNELS)()
# Get Enabled Channels
ACTUAL_ENABLED = ctypes.pointer(ctypes.c_ushort())

# Non blocking API call
SUCCESS = mlbert.mlbertmgr_captureHistogramData(HISTENABLEDCHANNEL,
ACTUAL_ENABLED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Capture Histogram Data! :", SUCCESS)
print("Histogram Data is Captured!")
print("Actual Enabled Channels: ", bin(ACTUAL_ENABLED[CHANNEL]))

# Read back the captured histogram data from the BERT
SUCCESS = mlbert.mlbertmgr_readHistogramData(CHANNEL, HIST)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Read Histogram Data! :", SUCCESS)
print("Histogram Data Read is done!")
```

Test Flow 10: FEC mode

Description: In this flow, user will configure FEC mode and links, then after having starting the ber for a few seconds.
The FEC data will be stored in: `MEASBERDATA -> RealFECData_4044` struct.
Remarque: when FEC mode is active, you cannot modify the tx and rx pattern.


Configure FEC
•mlbertmgr_setFECMode
•mlbertmgr_configureFECLinks
Get Accumulated RealFECData
•mlbertmgr_startBER
•mlbertmgr_getAvailableBERData
•mlbertmgr_stopBER

Python Sample Code:

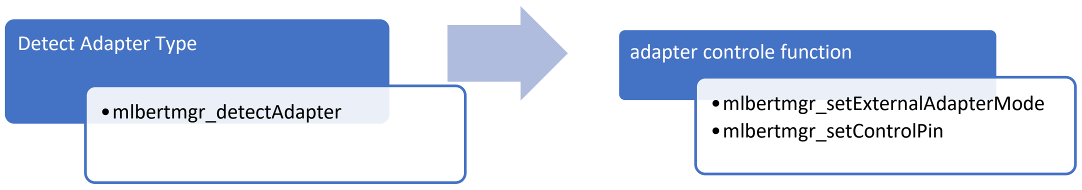
Environment: Python 3.8.5

Python wrapper

```
# FEC mode
FECMODE = pymlbertmgr.BERTMGR_FECMODE.BERTMGR_50G_KR4
#FEC PATTERN
FECPATTERN = pymlbertmgr.BERTMGR_FECPATTERN.FECPATTERN_IDLE
#channels fec link
CHANNELS = 0b11111111
SKIPRESET = False

# Set FEC Mode. Check The table of features for compatibility
SUCCESS =  mlbert.mlbertmgr_setFECMode(FECMODE, FECPATTERN, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FEC Mode! : ", SUCCESS)
print ("FEC Mode is set !")

# Set FEC Links. Check The table of features for compatibility
SUCCESS =  mlbert.mlbertmgr_configureFECLinks(CHANNELS,SKIPRESET ,
APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FEC Mode! : ", SUCCESS)
print ("FEC Mode is set !")
time.sleep(4)
# Start BER. This function requires Rx Lock.
mlbert.mlbertmgr_startBER(BERENABLEDCH, ACCUMULATE)

# BER Counting Time
# ML4054 BER Accumulation starts 4 seconds after enabling the BER process.
time.sleep(4)

#Get Available Data
SUCCESS =  mlbert.mlbertmgr_getAvailableBERData(MEASBERDATA, DATACOUNT)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Get Available Data!", SUCCESS)

for channel in range(NB_CHANNELS):
    print("channel: " , channel)
    print("\tenabled : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.enabled)
    print("\tenabledLinks : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.enabledLinks[channel])
    print("\tlockedLinks : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.lockedLinks[channel])
    print("\tTime: ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.Time[channel])
    print("\tBitCount : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.BitCount[channel])
    print("\tFEC_CorrectedBitCount_Interval : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.FEC_CorrectedBitCount_Interval[channel])
    print("\tFEC_CW_UnCorrectedCount_Interval : ", MEASBERDATA[DATACOUNT[0]
- 1].RealFECData_4044.FEC_CW_UnCorrectedCount_Interval[channel])
    print("\tFEC_CW_CorrectedCount_Interval : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.FEC_CW_CorrectedCount_Interval[channel])
    print("\tFEC_CW_ProcessedCount_Interval : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.FEC_CW_ProcessedCount_Interval[channel])
    print("\tFEC_CW_UncorrectedErrorRate_Interval : ",
MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.FEC_CW_UncorrectedErrorRate_Interval[channel])
    print("\tAccumulatedFEC_CW_UnCorrectedCount : ",
MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.AccumulatedFEC_CW_UnCorrectedCount[channel])
    print("\tAccumulatedFEC_CW_CorrectedCount : ", MEASBERDATA[DATACOUNT[0]
- 1].RealFECData_4044.AccumulatedFEC_CW_CorrectedCount[channel])
    print("\tAccumulatedFEC_CW_ProcessedCount : ", MEASBERDATA[DATACOUNT[0]
- 1].RealFECData_4044.AccumulatedFEC_CW_ProcessedCount[channel])
    print("\tAccumulatedFEC_CW_UncorrectedErrorRate : ",
MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.AccumulatedFEC_CW_UncorrectedErrorRate[channel])
    print("\tSER nSymbols : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.SER[channel].nSymbols)
    print("\tSER InstantSER : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.SER[channel].InstantSER[0])
    print("\tSER AccumulatedSER : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.SER[channel].AccumulatedSER[0])
    print("\tTotalBitCount : ", MEASBERDATA[DATACOUNT[0] -
1].RealFECData_4044.TotalBitCount[channel] ,"\n")

#Stop BER
SUCCESS =  mlbert.mlbertmgr_stopBER()
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Stop BER Test!", SUCCESS)
print ("BER Test stopped!")
```

Test Flow 11: Detect and Control Module Adapter

Description: This flow detects the type of adapter embedded into the instrument and sets the control pins. It should be used with instruments that support an integrated module host such as the ML4054B.


Detect Adapter Type
• mlbertmgr_detectAdapter
adapter controle function
• mlbertmgr_setExternalAdapterMode
• mlbertmgr_setControlPin

Python Sample Code:

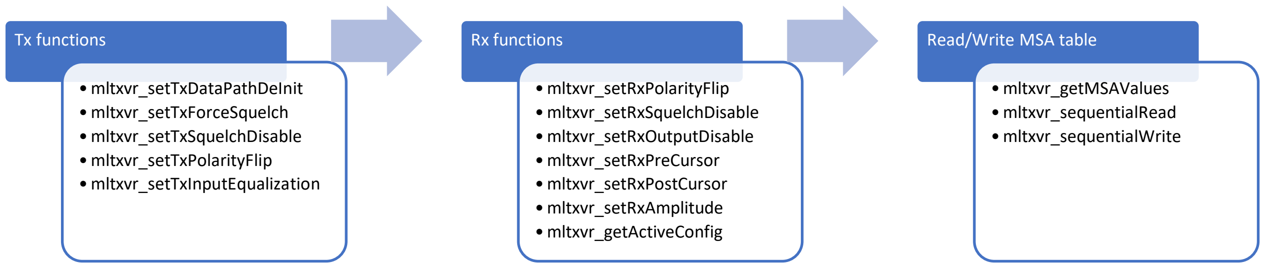
Environment: Python 3.8.5

Python wrapper

```
# Detect Module Adapter Type
ADAPTERTYPE = ctypes.pointer(ctypes.c_int())
SUCCESS = mlbert.mlbertmgr_detectAdapter(ADAPTERTYPE)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Detect Adapter! :", SUCCESS)
print("ADAPTER TYPE:", pymlbertmgr.ADAPTER_TYPE(ADAPTERTYPE[0]))

# Set Adapter I2C Control Mode to External
ISENABLED = False
SUCCESS = mlbert.mlbertmgr_setExternalAdapterMode(ISENABLED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set External Adapter Mode! :", SUCCESS)
print("Exterunal Adaper Mode is: ", ISENABLED)

# Control Adapter Pins
STATUS = False
AdapterContolePin =
pymlbertmgr.ADAPTER_HWSIGNAL_CNTRL.ADAPTER_HWSIGNAL_CNTRL_QDD_MODSEL_L
SUCCESS = mlbert.mlbertmgr_setControlPin(AdapterContolePin, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set Control Pin! :", SUCCESS)
print("ADAPTER_HWSIGNAL_CNTRL_QDD_MODSEL_L is set to ", STATUS)
```

Test Flow 12: Transceiver MSA Read/ Write functions

Description: This flow controls Tx and Rx channel parameters of the transceiver.


Tx functions
• mltxvr_setTxDataPathDeInit
• mltxvr_setTxForceSquelch
• mltxvr_setTxSquelchDisable
• mltxvr_setTxPolarityFlip
• mltxvr_setTxInputEqualization
Rx functions
• mltxvr_setRxPolarityFlip
• mltxvr_setRxSquelchDisable
• mltxvr_setRxOutputDisable
• mltxvr_setRxPreCursor
• mltxvr_setRxPostCursor
• mltxvr_setRxAmplitude
• mltxvr_getActiveConfig
Read/Write MSA table
• mltxvr_getMSAValues
• mltxvr_sequentialRead
• mltxvr_sequentialWrite

Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
# Transceiver Tx Controls
CHANNEL = 0
STATUS = False
# Transceiver TX Output Disable
SUCCESS = mlbert.mltxvr_setTxOutputDisable(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set TX Output Disable! :", SUCCESS)
print("Set TX Output Disable To: ", STATUS)

# Transceiver DataPathDeInit Configuration.
SUCCESS = mlbert.mltxvr_setTxDataPathDeInit(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set TX Data PathDeInit! :", SUCCESS)
print("Set TX Data PathDeInit To: ", STATUS)

# Transceiver TX Squelch Disable Configuration.
SUCCESS = mlbert.mltxvr_setTxSquelchDisable(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set TX Squelch Disable :", SUCCESS)
```

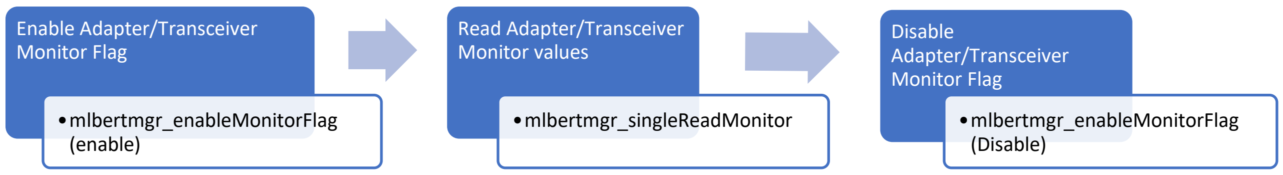
```
print("Set TX Squelch Disable To ", STATUS)

# Transceiver TX Force Squelch Configuration.
SUCCESS = mlbert.mltxvr_setTxForceSquelch(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Set TX Force Squelch! :", SUCCESS)
print("Set TX Force Squelch TO ", STATUS)

# Transceiver TX Polarity Flip Configuration.
SUCCESS = mlbert.mltxvr_setTxPolarityFlip(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set TX Polarity Flip! :", SUCCESS)
print("Set TX Polarity Flip to ", STATUS)

# Transceiver TX input equalization
# CMIS Range is from 0-12.
VALUE = 1
SUCCESS = mlbert.mltxvr_setTxInputEqualization(CHANNEL, VALUE)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set TX Input Equalization! :", SUCCESS)
print("Set TX Input Equalization To: ", VALUE)

# Transceiver RX Controls
STATUS = False
# Transceiver Rx Polarity Flip
SUCCESS = mlbert.mltxvr_setRxPolarityFlip(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set RX Polarity Flip! :", SUCCESS)
print("Set RX Polarity Flip To: ", STATUS)

# Transceiver RX Squelch Disable Configuration.
SUCCESS = mlbert.mltxvr_setRxSquelchDisable(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set RX Squelch Disable! :", SUCCESS)
print("Set RX Squelch Disable To: ", STATUS)

# Transceiver RX Output Disable Configuration.
SUCCESS = mlbert.mltxvr_setRxOutputDisable(CHANNEL, STATUS)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set RX Output Disable! :", SUCCESS)
print("Set RX Output Disable To: ", STATUS)

# Transceiver RX Output Pre-Cursor.
# CMIS Range from 0-7
VALUE = 1
SUCCESS = mlbert.mltxvr_setRxPreCursor(CHANNEL, VALUE)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set Rx Pre Cursor! :", SUCCESS)
print("Set Set RX Pre Cursor To: ", VALUE)

# Transceiver RX Output Post-Cursor.
# Range from 0-7
SUCCESS = mlbert.mltxvr_setRxPostCursor(CHANNEL, VALUE)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set Rx post Cursor! :", SUCCESS)
print("Set Set RX post Cursor To: ", VALUE)

# Transceiver RX Output Amplitude.
TRANS_RX_AMPLITUDE =
pymlbertmgr.TXVR_RX_AMPLITUDE.TXVR_RX_AMPLITUDE_100_400
```

```
SUCCESS = mlbert.mltxvr_setRxAmplitude(CHANNEL, TRANS_RX_AMPLITUDE)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Set RX Amplitude! :", SUCCESS)
print("Transceivwe Rx Amplitude is Set")

# Get Transceiver Active Configuration Settings
TRANS_ACTIVECONFIG =
ctypes.pointer(pymlbertmgr.TXVR_ConfigurationSettings())
TRANS_NB_CHANNEL = 8
SUCCESS = mlbert.mltxvr_getActiveConfig(TRANS_ACTIVECONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Get Transceiver Active Configuration! :",
SUCCESS)
print("Reading of Transceiver Active Configuration is successfull")
# Printing all value of the ActivConfg struct
for channel in range(TRANS_NB_CHANNEL):
    print("channel: ", channel)
    for fields in TRANS_ACTIVECONFIG[0]._fields_:
        print(fields[0], " ", getattr(TRANS_ACTIVECONFIG[0],
fields[0])[channel])

# Reads Transceiver MSA values
NB_PAGES = 7
MSAPAGES = (ctypes.c_int * NB_PAGES)()
MSAPAGES[0] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_LOWERMEMORY
MSAPAGES[1] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_0
MSAPAGES[2] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_1
MSAPAGES[3] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_2
MSAPAGES[4] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_3
MSAPAGES[5] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_16
MSAPAGES[6] = pymlbertmgr.TXVR_MSA_PAGE.TXVR_MSA_PAGE_17
MSAVALUES = (ctypes.c_ushort * (128 * NB_PAGES))()
SUCCESS = mlbert.mltxvr_getMSAValues(MSAPAGES, MSAVALUES, NB_PAGES)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed To Get MSA Values! :", SUCCESS)
print("Getting MSA Values is successfull!")

# Sequential MSA Read
# Register addresse range is 128->255, Except LOWERMEMORY where the
address range is 0->127
# LOWERMEMORY page index is 0
READING_PAGE_SELECT = 0
READING_REGISTER_ADDRESS = 128
READING_DATA_LENGTH = 128
READING_DATA_BUFFER = (ctypes.c_ushort * READING_DATA_LENGTH)()
READING_BANK_SELECT = 0
SUCCESS = mlbert.mltxvr_sequentialRead(READING_PAGE_SELECT,
                                       READING_REGISTER_ADDRESS,
                                       READING_DATA_LENGTH,
                                       READING_DATA_BUFFER,
                                       READING_BANK_SELECT)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Sequentially Read Transceiver Data!",
SUCCESS)
print("Sequential Reading is Successfull!")

# Sequential MSA Write
WRITING_PAGE_SELECT = 0
WRITING_REGISTER_ADDRESS = 0
WRITING_DATA_LENGTH = 128
```

```
WRITING_DATA_BUFFER = (ctypes.c_ulong * WRITING_DATA_LENGTH)()
WRITING_BANK_SELECT = 0
SUCCESS = mlbert.mltxvr_sequentialWrite(WRITING_PAGE_SELECT,
                                        WRITING_REGISTER_ADDRESS,
                                        WRITING_DATA_LENGTH,
                                        WRITING_DATA_BUFFER,
                                        WRITING_BANK_SELECT)

if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Sequentially WRITE Transceiver Data ",
SUCCESS)
print("Sequential Writing is Successfull!")
```

Test Flow 13: Monitor Adapter and Transceiver.

Description: In this flow, the user can enable the adapter and transceiver monitor flags. Then, the user reads the values of the enabled flags.


Enable Adapter/Transceiver Monitor Flag
• mlbertmgr_enableMonitorFlag (enable)
Read Adapter/Transceiver Monitor values
• mlbertmgr_singleReadMonitor
Disable Adapter/Transceiver Monitor Flag
• mlbertmgr_enableMonitorFlag (Disable)

Python Sample Code:
Environment: Python 3.8.5
Python wrapper

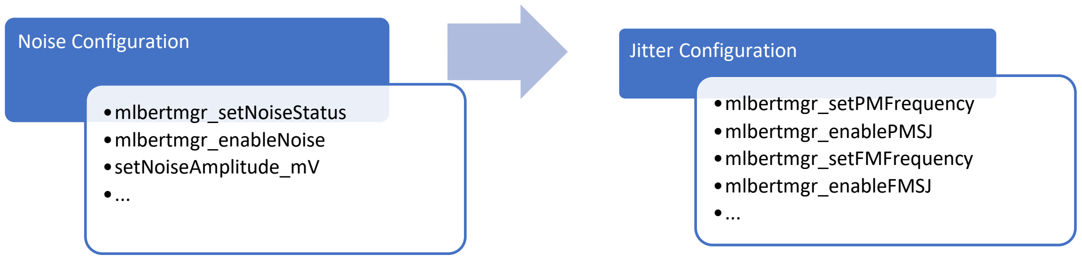
```
# Enable Adapter Monitor Flag
MONITORFLAG = pymlbertmgr.BERTMGR_MONITOR_FLAGS.BERTMGR_MONITOR_ADAPTER
# Monitor Adapter requires 26 ushort values
ADAPTER_MONITOR_VALUES = (ctypes.c_ushort * 26)()
ENABLED = True
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(MONITORFLAG, ENABLED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Enable Adapter monitor Flag: ", SUCCESS)
print("Adapter monitor Flag Is Enabled!")
# Wait for Monitor Accumulation
time.sleep(0.35)

# Single-Read Monitor
SUCCESS = mlbert.mlbertmgr_singleReadMonitor(MONITORFLAG,
ADAPTER_MONITOR_VALUES)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Read Monitor! : ", SUCCESS)
print("Adapter single Read Monitor is done!")
# Disable Monitor
ENBALED = False
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(MONITORFLAG, ENBALED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Disable Adapter monitor Flag: ", SUCCESS)
print("Adapter monitor Flag Is Disabled!")
```

```
# Print Out Adapter Monitor Values. Voltage values must be converted by
dividing by 256
print("VCC =  ", ADAPTER_MONITOR_VALUES[0] / 256, "V")
print("VCC1 =  ", ADAPTER_MONITOR_VALUES[1] / 256, "V")
print("VCC-TX =  ", ADAPTER_MONITOR_VALUES[2] / 256, "V")
print("VCC-RX =  ", ADAPTER_MONITOR_VALUES[3] / 256, "V")
print("VOLTAGE5 =  ", ADAPTER_MONITOR_VALUES[4], "V")
print("VOLTAGE6 =  ", ADAPTER_MONITOR_VALUES[5], "V")
print("VOLTAGE7 =  ", ADAPTER_MONITOR_VALUES[6], "V")
print("VOLTAGE8 =  ", ADAPTER_MONITOR_VALUES[7], "V")
print("I-VCC  =  ", ADAPTER_MONITOR_VALUES[8], "mA")
print("I-VCC1  =  ", ADAPTER_MONITOR_VALUES[9], "mA")
print("I-VCC-TX  =  ", ADAPTER_MONITOR_VALUES[10], "mA")
print("I-VCC-RX  =  ", ADAPTER_MONITOR_VALUES[11], "mA")
print("CURRENT5 =  ", ADAPTER_MONITOR_VALUES[12], "mA")
print("CURRENT6 =  ", ADAPTER_MONITOR_VALUES[13], "mA")
print("CURRENT7 =  ", ADAPTER_MONITOR_VALUES[14], "mA")
print("CURRENT8 =  ", ADAPTER_MONITOR_VALUES[15], "mA")
print("Temp1 =  ", ADAPTER_MONITOR_VALUES[16])
print("Temp2 =  ", ADAPTER_MONITOR_VALUES[17])
print("Temp3 =  ", ADAPTER_MONITOR_VALUES[18])
print("Temp4 =  ", ADAPTER_MONITOR_VALUES[19])
print("Temp5 =  ", ADAPTER_MONITOR_VALUES[20])
print("Temp6 =  ", ADAPTER_MONITOR_VALUES[21])
print("Temp7 =  ", ADAPTER_MONITOR_VALUES[22])
print("Temp8 =  ", ADAPTER_MONITOR_VALUES[23])

print("Control Signals: ")
# Read back control Pins Status
if ((ADAPTER_MONITOR_VALUES[24] & 1 << 0) == 1 << 0):
    print("\tModeSetL is enabled")
else:
    print("\tModeSetL is disabled")

if ((ADAPTER_MONITOR_VALUES[24] & (1 << 1)) == 1 << 1):
    print("\tResetL is enabled")
else:
    print("\tResetL is disabled")

if ((ADAPTER_MONITOR_VALUES[24] & 1 << 2) == 1 << 2):
    print("\tLPMode is enabled")
else:
    print("\tLPMode is disabled")

print("RO Signals: ")
# Active Low
if ((ADAPTER_MONITOR_VALUES[24] & 1 << 3) != 1 << 3):
    print("\tModePrsL is active")
else:
    print("\tModePrsL is deactive")
# Active Low
if ((ADAPTER_MONITOR_VALUES[24] & 1 << 4) != 1 << 4):
    print("\tIntL is active")
else:
    print("\tIntL is deactive")

print("Adapter IsExternalMode:  ", ADAPTER_MONITOR_VALUES[25])

# Enable Transceiver Monitor Flag
MONITORFLAG = pymlbertmgr.BERTMGR_MONITOR_FLAGS.BERTMGR_MONITOR_TRANSCEIVER
```

```
# Monitor Transceiver requires ushort values.
TRANS_MONITOR_VALUES = (ctypes.c_ushort * 80)()
ENABLED = True
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(MONITORFLAG, ENABLED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Enable Tranciver monitor Flag: ", SUCCESS)
print("Tranciver monitor Flag Is Enabled!")
# Wait for Monitor Accumulation
time.sleep(0.35)

# Single-Read Monitor
SUCCESS = mlbert.mlbertmgr_singleReadMonitor(MONITORFLAG,
TRANS_MONITOR_VALUES)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Read Monitor! : ", SUCCESS)
print("Single Read Monitor is done!")

# Disable Transceiver Monitor Flag
ENBALED = False
SUCCESS = mlbert.mlbertmgr_enableMonitorFlag(MONITORFLAG, ENBALED)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to Disable Transceiver monitor Flag: ",
SUCCESS)
print("Transceiver monitor Flag Is Disabled!")

# Conversion is Performed According to CMIS Standard
print("tempSupplyFlags:  ", TRANS_MONITOR_VALUES[0])
print("aux1Aux2Flags:  ", TRANS_MONITOR_VALUES[1])
print("aux3VendorFlags:  ", TRANS_MONITOR_VALUES[2])
print("Temp1:  ", TRANS_MONITOR_VALUES[3] / 256)
print("Temp2:  ", TRANS_MONITOR_VALUES[4] / 256)
print("Temp3:  ", TRANS_MONITOR_VALUES[5] / 256)
print("Temp4:  ", TRANS_MONITOR_VALUES[6] / 256)
print("VCC:  ", TRANS_MONITOR_VALUES[7] / 10000, "V")
print("VCC2:  ", TRANS_MONITOR_VALUES[8] / 10000, "V")
print("VCC3:  ", TRANS_MONITOR_VALUES[9] / 10000, "V")
print("VCC4:  ", TRANS_MONITOR_VALUES[10] / 10000, "V")
print("aux1:  ", TRANS_MONITOR_VALUES[11])
print("aux2:  ", TRANS_MONITOR_VALUES[12])
print("aux3:  ", TRANS_MONITOR_VALUES[13])
print("STATE_CHANGE:  ", TRANS_MONITOR_VALUES[14])
print("TX_FAULT:  ", TRANS_MONITOR_VALUES[15])
print("TX_LOS:  ", TRANS_MONITOR_VALUES[16])
print("TX_LOL:  ", TRANS_MONITOR_VALUES[17])
print("TXPOWER_HA:  ", TRANS_MONITOR_VALUES[18])
print("TXPOWER_LA:  ", TRANS_MONITOR_VALUES[19])
print("TXPOWER_HW:  ", TRANS_MONITOR_VALUES[20])
print("TXPOWER_LW:  ", TRANS_MONITOR_VALUES[21])
print("TXBIAS_HA:  ", TRANS_MONITOR_VALUES[22])
print("TXBIAS_LA:  ", TRANS_MONITOR_VALUES[23])
print("TXBIAS_HW:  ", TRANS_MONITOR_VALUES[24])
print("TXBIAS_LW:  ", TRANS_MONITOR_VALUES[25])
print("RX_LOS:  ", TRANS_MONITOR_VALUES[26])
print("RX_LOL:  ", TRANS_MONITOR_VALUES[27])
print("RXPOWER_HA:  ", TRANS_MONITOR_VALUES[28])
print("RXPOWER_LA:  ", TRANS_MONITOR_VALUES[29])
print("RXPOWER_LW:  ", TRANS_MONITOR_VALUES[30])
print("RXPOWER_LW:  ", TRANS_MONITOR_VALUES[31])
print("TX0:  ", TRANS_MONITOR_VALUES[32] / 10000, "mW")
print("TX1:  ", TRANS_MONITOR_VALUES[33] / 10000, "mW")
print("TX2:  ", TRANS_MONITOR_VALUES[34] / 10000, "mW")
print("TX3:  ", TRANS_MONITOR_VALUES[35] / 10000, "mW")
print("TX4:  ", TRANS_MONITOR_VALUES[36] / 10000, "mW")
print("TX5:  ", TRANS_MONITOR_VALUES[37] / 10000, "mW")
print("TX6:  ", TRANS_MONITOR_VALUES[38] / 10000, "mW")
print("TX7:  ", TRANS_MONITOR_VALUES[39] / 10000, "mW")
print("TX8:  ", TRANS_MONITOR_VALUES[40] / 10000, "mW")
print("TX9:  ", TRANS_MONITOR_VALUES[41] / 10000, "mW")
print("TX10:  ", TRANS_MONITOR_VALUES[42] / 10000, "mW")
print("TX11:  ", TRANS_MONITOR_VALUES[43] / 10000, "mW")
print("TX12:  ", TRANS_MONITOR_VALUES[44] / 10000, "mW")
print("TX13:  ", TRANS_MONITOR_VALUES[45] / 10000, "mW")
print("TX14:  ", TRANS_MONITOR_VALUES[46] / 10000, "mW")
print("TX15:  ", TRANS_MONITOR_VALUES[47] / 10000, "mW")
print("TX-Bias0:  ", TRANS_MONITOR_VALUES[48] * 0.002, "mA")
print("TX-Bias1:  ", TRANS_MONITOR_VALUES[49] * 0.002, "mA")
print("TX-Bias2:  ", TRANS_MONITOR_VALUES[50] * 0.002, "mA")
print("TX-Bias3:  ", TRANS_MONITOR_VALUES[51] * 0.002, "mA")
print("TX-Bias4:  ", TRANS_MONITOR_VALUES[52] * 0.002, "mA")
print("TX-Bias5:  ", TRANS_MONITOR_VALUES[53] * 0.002, "mA")
print("TX-Bias6:  ", TRANS_MONITOR_VALUES[54] * 0.002, "mA")
print("TX-Bias7:  ", TRANS_MONITOR_VALUES[55] * 0.002, "mA")
print("TX-Bias8:  ", TRANS_MONITOR_VALUES[56] * 0.002, "mA")
print("TX-Bias9:  ", TRANS_MONITOR_VALUES[57] * 0.002, "mA")
print("TX-Bias10:  ", TRANS_MONITOR_VALUES[58] * 0.002, "mA")
print("TX-Bias11:  ", TRANS_MONITOR_VALUES[59] * 0.002, "mA")
print("TX-Bias12:  ", TRANS_MONITOR_VALUES[60] * 0.002, " mA")
print("TX-Bias13:  ", TRANS_MONITOR_VALUES[61] * 0.002, " mA")
print("TX-Bias14:  ", TRANS_MONITOR_VALUES[62] * 0.002, " mA")
print("TX-Bias15:  ", TRANS_MONITOR_VALUES[63] * 0.002, " mA")
print("RX0:  ", TRANS_MONITOR_VALUES[64] / 10000, "mW")
print("RX1:  ", TRANS_MONITOR_VALUES[65] / 10000, "mW")
print("RX2:  ", TRANS_MONITOR_VALUES[66] / 10000, "mW")
print("RX3:  ", TRANS_MONITOR_VALUES[67] / 10000, "mW")
print("RX4:  ", TRANS_MONITOR_VALUES[68] / 10000, "mW")
print("RX5:  ", TRANS_MONITOR_VALUES[69] / 10000, "mW")
print("RX6:  ", TRANS_MONITOR_VALUES[70] / 10000, "mW")
print("RX7:  ", TRANS_MONITOR_VALUES[71] / 10000, "mW")
print("RX8:  ", TRANS_MONITOR_VALUES[72] / 10000, "mW")
print("RX9:  ", TRANS_MONITOR_VALUES[73] / 10000, "mW")
print("RX10:  ", TRANS_MONITOR_VALUES[74] / 10000, "mW")
print("RX11:  ", TRANS_MONITOR_VALUES[75] / 10000, "mW")
print("RX12:  ", TRANS_MONITOR_VALUES[76] / 10000, "mW")
print("RX13:  ", TRANS_MONITOR_VALUES[77] / 10000, "mW")
print("RX14:  ", TRANS_MONITOR_VALUES[78] / 10000, "mW")
print("RX15:  ", TRANS_MONITOR_VALUES[79] / 10000, "mW")
```

Test Flow 14: Noise and Jitter Configuration

Description: This flow enables noise and jitter injection. It should be used with instruments that support these options as the ML4079EN.



Python Sample Code:
Environment: Python 3.8.5
Python wrapper

```
ENBALED = False
# enable disable shallowLoopBack
SUCCESS = mlbert.mlbertmgr_setShallowLoopback(ENBALED, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable Shallow Loop back: ", SUCCESS)
print("Shallow Loop back enabled!")

# Noise configuration
ENBALED = True
APPLYCONFIG = False
SUCCESS = mlbert.mlbertmgr_setNoiseStatus(ENBALED, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable noise: ", SUCCESS)
print("noise enabled!")

NOISELINERATE = ctypes.pointer(ctypes.c_double(25.78125))
SUCCESS = mlbert.mlbertmgr_setNoiseLinerate(NOISELINERATE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set noise line rate: ", SUCCESS)
print("noise line rate set!")

STATUS = True
AMPLITUDE = 10
BURSTRATE = 25.78125
ACTUALRATE = ctypes.pointer(ctypes.c_double(0))
# Creates PatternConfig initial struct
NOISETXPATTERN = pymlbertmgr.PatternConfig()
# Tx Pattern
NOISETXPATTERN.pattern =
pymlbertmgr.BERTMGR_PATTERNTYPE.BERTMGR_USERDEFINED
# Tx Invertion
NOISETXPATTERN.invert = False
isCalibrated = False

for channel in range(NB_CHANNELS):

    SUCCESS = mlbert.mlbertmgr_enableNoise(channel, STATUS, APPLYCONFIG)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set noise eye mode: ", SUCCESS)
    print("noise eye mode set!")

    if (isCalibrated):
        NOISEAMPLITUDEMV = 16
        SUCCESS = mlbert.mlbertmgr_setNoiseAmplitude_mV(channel,
NOISEAMPLITUDEMV, APPLYCONFIG)
        if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
            raise Exception("Failed to set amplitude (mV): ", SUCCESS)
        print("noise amplitude set!")
    else:
        if (NOISETXPATTERN.pattern !=
pymlbertmgr.BERTMGR_PATTERNTYPE.BERTMGR_USERDEFINED):
            SUCCESS = mlbert.mlbertmgr_setNoiseBurstRate(channel,
BURSTRATE, ACTUALRATE, APPLYCONFIG)
            if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
                raise Exception("Failed to set noise pattern: ", SUCCESS)
            print("noise pattern set!")
            print("actual noise bert rate = ", ACTUALRATE[0])

        else:
            SUCCESS = mlbert.mlbertmgr_setNoiseLevel(channel, AMPLITUDE,
APPLYCONFIG)
            if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
                raise Exception("Failed to set noise Level: ", SUCCESS)
            print("noise Level set!")

    SUCCESS = mlbert.mlbertmgr_setNoiseTxPattern(channel, NOISETXPATTERN,
APPLYCONFIG)
    if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
        raise Exception("Failed to set noise pattern: ", SUCCESS)
    print("noise pattern set!")

# PM SJ configuration
PMPHASESHIFTAMPLITUDE = 200
SUCCESS = mlbert.mlbertmgr_setPMPhaseShift(PMPHASESHIFTAMPLITUDE,
APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set PM phase shift amplitude: ", SUCCESS)
print("PM phase shift amplitude set!")

PMFREQUENCY = 100  # PMFREQUENCY in KHz
SUCCESS = mlbert.mlbertmgr_setPMFrequency(PMFREQUENCY, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set PM frequency: ", SUCCESS)
print("PM frequency set!")

STATUS = True
APPLYCONFIG = True
SUCCESS = mlbert.mlbertmgr_enablePMSJ(STATUS, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable PM SJ: ", SUCCESS)
print("PM SJ enabled!")

PMAMPLITUDE = 100  # V
SUCCESS = mlbert.mlbertmgr_setPMSJAmplitude_ps(PMAMPLITUDE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set PM SJ Amplitude ps: ", SUCCESS)
print("PM SJ Amplitude set!")
```

```
# PM RJ configuration
STATUS = True
SUCCESS = mlbert.mlbertmgr_enablePMRJ(STATUS, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable PMRJ: ", SUCCESS)
print("PM RJ enabled!")

PMRJAMPLITUDE = 2050
SUCCESS = mlbert.mlbertmgr_setPMRJAmplitude(PMRJAMPLITUDE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set PM RJ amplitude: ", SUCCESS)
print("PM RJ amplitude set!")

PMPRBBSAMPLITUDE = 200
SUCCESS = mlbert.mlbertmgr_setPMBUJAmplitude(PMPRBBSAMPLITUDE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set PM BUJ amplitude: ", SUCCESS)
print("PM PRBS amplitude set!")

# FM SJ control

FMPHASESHIFT = 665
SUCCESS = mlbert.mlbertmgr_setFMPhaseShift(FMPHASESHIFT, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FM RJ phase shift: ", SUCCESS)
print("FM RJ phase shift is set!")

FMFREQUENCY = 100
SUCCESS = mlbert.mlbertmgr_setFMFrequency(FMFREQUENCY, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FM frequency: ", SUCCESS)
print("FM frequency set!")

STATUS = True
APPLYCONFIG = True
SUCCESS = mlbert.mlbertmgr_enableFMSJ(STATUS, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable FMSJ: ", SUCCESS)
print("FM enabled!")

VALUE = 22
SUCCESS = mlbert.mlbertmgr_setFMSJAmplitude_ps(VALUE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FM Amplitude in ps: ", SUCCESS)
print("FM Amplitude in ps is set!")

# FM RJ controle
FMRJSTATUS = 22
SUCCESS = mlbert.mlbertmgr_enableFMRJ(FMRJSTATUS, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to enable FM RJ: ", SUCCESS)
print("FM RJ enabled!")

FMRJAMPLITUDE = 22
SUCCESS = mlbert.mlbertmgr_setFMRJAmplitude(FMRJAMPLITUDE, APPLYCONFIG)
if SUCCESS != pymlbertmgr.BERTMGR_STATUS.BERTMGR_SUCCESS:
    raise Exception("Failed to set FM RJ amplitude: ", SUCCESS)
print("FM RJ amplitude is set!")
```

```
typedef struct mlbertmgr mlbertmgr;     // API wrapper structure

enum BERTMGR_AFETRIM_OPT
{
    BERTMGR_AFETRIM_NEG4DB = 0,
    ERTMGR_AFETRIM_NEG10DB
}

enum BERTMGR_CALIBRATIONMODE
{
    BERTMGR_CALMODE_ADV = -1,             // Advanced mode
    BERTMGR_CALMODE_LRLV = 0,             // Low-rate/low-voltage
    BERTMGR_CALMODE_LRHV,                 // Low-rate/high-voltage
    BERTMGR_CALMODE_HRLV,                 // High-rate/low-voltage
    BERTMGR_CALMODE_HRHV                  // High-rate/high-voltage
}

enum BERTMGR_CDRDIVIDER
{
    BERTMGR_CDR_DIV32 = 1<<5,
    BERTMGR_CDR_DIV64 = 1<<6,
    BERTMGR_CDR_DIV128 = 1<<7,
    BERTMGR_CDR_DIV256 = 1<<8,
    BERTMGR_CDR_DIV512 = 1<<9,
    BERTMGR_CDR_DIV1024 = 1<<10,
    BERTMGR_CDR_DIV2048 = 1<<11,
    BERTMGR_CDR_DIV4096 = 1<<12
}

enum BERTMGR_CLOCKMODE
{
    BERTMGR_MONITORCLOCK_CH0toCH3 = 0,
    BERTMGR_EXTERNAL,
    BERTMGR_REFCLk,
    BERTMGR_MONITORCLOCK_CH4toCH7,
    BERTMGR_CDR_CH0toCH3,
    BERTMGR_CDR_CH4toCH7,
    BERTMGR_REFCLK2
}

enum BERTMGR_CLOCKSOURCE
{
    BERTMGR_EXTERNALCLKSRC = 0,
    BERTMGR_INTERNALCLKSRC
}

enum BERTMGR_DSPMODE
{
    BERTMGR_DSP_MODE_SLC1 = 0,            // PAM4 Slicer
    BERTMGR_DSP_MODE_SLC1_LDEQ,           // PAM4 Slicer + Level-
                                          dependent equalizer (LDEQ)
    BERTMGR_DSP_MODE_SLC1_RC_SLC2,        // PAM4 Slicer + Reflection
                                          canceller (RC)
    BERTMGR_DSP_MODE_SLC1_RC_LDEQ,        // PAM4 Slicer + LDEQ + RC
    BERTMGR_DSP_MODE_DFE1,                // Decision Feedback
                                          Equalizer (DFE)
    BERTMGR_DSP_MODE_DFE1_RC_DFE2,       // DFE + RC
    BERTMGR_DSP_MODE_SLC1_MPICAN_SLC2,   // PAM4 Slicer + Multipath
                                         interference canceller
                                         (MPICAN)
    BERTMGR_DSP_MODE_SLC1_MPICAN_LDEQ,   // PAM4 Slicer + LDEQ +
                                         MPICAN
    BERTMGR_DSP_MODE_SLC1_RC_MPICAN_SLC2, // PAM4 Slicer + RC +
                                           MPICAN
    BERTMGR_DSP_MODE_SLC1_RC_MPICAN_LDEQ, // PAM4 Slicer + LDEQ +
                                           RC + MPICAN
    BERTMGR_DSP_MODE_DFE1_MPICAN_DFE2,   // DFE + MPICAN
    BERTMGR_DSP_MODE_DFE1_RC_MPICAN_DFE2 // DFE + RC + MPICAN
}

enum BERTMGR_ERRORINSERTIONMODES
{
    BERTMGR_ERRINJ_PAT_BIT0 = 0,         // bit 0 one MSB
    BERTMGR_ERRINJ_PAT_BIT1,             // bit 1 one LSB
    BERTMGR_ERRINJ_PAT_BIT01,            // bit 0 and 1 one PAM4
                                         (MSB and LSB)
    BERTMGR_ERRINJ_PAT_MSBS,             // all MSBs
    BERTMGR_ERRINJ_PAT_LSBS,             // all LSBs
    BERTMGR_ERRINJ_PAT_ALL               // all bits
}

enum BERTMGR_FECMODE
{
    BERTMGR_FECDISABLED = -1,
    BERTMGR_400G_KP8_TO_KP4 = 0,
    BERTMGR_200G_KP4_TO_KP2,
    BERTMGR_200G_KP4_TO_KP4,
    BERTMGR_100G_KP2_TO_KP1,
    BERTMGR_100G_KP4_TO_KP4,
    BERTMGR_100G_KP4_TO_KP2,
    BERTMGR_100G_PCS4_TO_KR1,
    BERTMGR_50G_KP1_TO_KP1,
    BERTMGR_50G_KP2_TO_KP2,
    BERTMGR_50G_KR2_TO_KR1,
    BERTMGR_25G_KR1_TO_KR1,
    BERTMGR_25G_KP1_TO_KP1,
    BERTMGR_50G_KS,
    BERTMGR_50G_KR,
    BERTMGR_50G_KP,
    BERTMGR_100G_KR,
    BERTMGR_100G_KP,
    BERTMGR_200G_KP,
    BERTMGR_400G_KP


    //  ML4054B FEC Modes
    BERTMGR_25G_FC = 40,
    BERTMGR_25G_KR4 = 41,
    BERTMGR_25G_KP4 = 42,
    BERTMGR_50G_FC = 43,
    BERTMGR_50G_KR4 = 44,
    BERTMGR_50G_KP4 = 45,
    BERTMGR_100G_FC = 46,
    BERTMGR_100G_KR4 = 47,
    BERTMGR_100G_KP4 = 48,
    BERTMGR_200G_FC = 49,
    BERTMGR_200G_KR4 = 50,
    BERTMGR_200G_KP4 = 51
}

enum BERTMGR_FECPATTERN
{
    BERTMGR_FECPATTERN_DISABLED = -1,
    BERTMGR_FECPATTERN_IDLE = 0,
    BERTMGR_FECPATTERN_LOCALFAULT,
    BERTMGR_FECPATTERN_REMOTEFAULT
}

enum BERTMGR_MONITOR_FLAGS
{
    BERTMGR_MONITOR_LOS = 0x1 << 0,   // LOS Enable Flag (bit 0)
    BERTMGR_MONITOR_DSP = 0x1 << 1,   // DSP Enable Flag (bit 1)
    BERTMGR_MONITOR_SIGNALDETECT = 0x1 << 2,   // Signal Detect
                                               Flag (bit 2)
    BERTMGR_MONITOR_TXLOCK = 0x1 << 3,   // Tx Lock Flag (bit 3)
    BERTMGR_MONITOR_RXLOCK= 0x1 << 4,   // RX Lock Flag (bit 4)
    BERTMGR_MONITOR_TEMPERATURE= 0x1 << 5,   // Temperature Flag
                                             (bit 5)
    BERTMGR_MONITOR_SNR= 0x1 << 6,   // SNR Flag (bit 6)
    BERTMGR_MONITOR_VOLTAGE= 0x1 << 7,   // Voltage Flag (bit 7)
    BERTMGR_MONITOR_CURRENT= 0x1 << 8,   // Current Flag (bit 8)
    BERTMGR_MONITOR_FFETAPS= 0x1 << 9,   // FFE Taps Flag (bit 9)
    BERTMGR_MONITOR_XT_TXLOCK= 0x1 << 10,   // XT Flag (bit 10)
    BERTMGR_MONITOR_ADAPTER= 0x1 << 11,   // Adaptar Flag (bit 11)
    BERTMGR_MONITOR_TRANSCEIVER= 0x1 << 12   // Transceiver Flag
                                             (bit 11)
}

enum BERTMGR_MONITORDIVIDER
{
    BERTMGR_MONITOR_DIV1 = 1<<0,
    BERTMGR_MONITOR_DIV4 = 1<<2,
    BERTMGR_MONITOR_DIV8 = 1<<3,
    BERTMGR_MONITOR_DIV16 = 1<<4,
    BERTMGR_MONITOR_DIV32 = 1<<5,
    BERTMGR_MONITOR_DIV64 = 1<<6,
    BERTMGR_MONITOR_DIV128 = 1<<7
}

enum BERTMGR_PATTERNTYPE
{
    BERTMGR_PRBS7 = 0,
    BERTMGR_PRBS9_4,
    BERTMGR_PRBS9_5,
    BERTMGR_PRBS11,
    BERTMGR_PRBS13,
    BERTMGR_PRBS15,
    BERTMGR_PRBS16,
    BERTMGR_PRBS23,
    BERTMGR_PRBS31,
    BERTMGR_PRBS58,
    BERTMGR_USERDEFINED,
    BERTMGR_JP03B,
    BERTMGR_LIN,
    BERTMGR_CJT,
    BERTMGR_SSPRQ,
    BERTMGR_SQ16,
    BERTMGR_SQ32,
    BERTMGR_IEEE8023BS_2,
    BERTMGR_IEEE8023BS_4,
    BERTMGR_OIFCEI311
}

enum BERTMGR_SIGMODULATION
{
    BERTMGR_PAM4 = 0,
    BERTMGR_NRZ
}

enum BERTMGR_STATUS
{
    BERTMGR_SUCCESS = 0,           // Operation successful
    BERTMGR_FAILED = 1,            // Operation failed
    BERTMGR_TIMEOUT = 2,           // Operation timed out
    BERTMGR_NOT_CONNECTED = 3,     // Device not connected
    BERTMGR_INVALID_INPUT = 4,     // Invalid input provided
    BERTMGR_INVALID_INSTANCE = 5,  // Invalid instance
    BERTMGR_INVALID_CALIBRATION = 6,  // Invalid calibration
    BERTMGR_INCOMPATIBLE_CONFIG = 7,  // Operation not compatible
with current configuration
    BERTMGR_UNSUPPORTED_OPTION = 8,   // Option not supported for
the current board
    BERTMGR_CLOCK_FAILED = 9,      // Clock failed
    BERTMGR_BER_DISABLED = 10,     // BER disabled
    BERTMGR_BER_ENABLED = 11       // BER enabled }

enum BERTMGR_TAPSMODE
{
    BERTMGR_3TAPS = 0,
    BERTMGR_7TAPS
}

struct AdvancedAmplitude
{
    int mainTap;
    int postEmphasis;
    int preEmphasis;
    int innerLevel;
    int outterLevel;
    int scalingLevel;
    int advancedTaps[7];
}

struct AmpRange
{
    int min;          // Minimum optimal amplitude value
    int max;          // Maximum optimal amplitude value
    BERTMGR_CALIBRATIONMODE calMode;    // Calibration mode
}

struct BERData
{
    bool enabled;
    bool enabledChannels[MAXCHANNELS];// Channels enabled indicator
    bool lockedChannels[MAXCHANNELS]; // Channels lock indicator
    double Time[MAXCHANNELS];         // Constructed time data
    ulong BitCount[MAXCHANNELS];      // Bit Count data MSB/LSB
    uint ErrorCount_MSB[MAXCHANNELS];
    uint ErrorCount_LSB[MAXCHANNELS];
    ulong ErrorCount[MAXCHANNELS];
    //  Constructed data
    ulong AccumulatedErrorCount_MSB[MAXCHANNELS];
    double BER_MSB_Interval[MAXCHANNELS];
    double BER_MSB_Realtime[MAXCHANNELS];
    ulong AccumulatedErrorCount_LSB[MAXCHANNELS];
    double BER_LSB_Interval[MAXCHANNELS];
    double BER_LSB_Realtime[MAXCHANNELS];
    ulong AccumulatedErrorCount[MAXCHANNELS];
    double BER_Interval[MAXCHANNELS];
    double BER_Realtime[MAXCHANNELS];
    ulong TotalBitCount[MAXCHANNELS];// Total Bit Count data
                                        MSB+LSB
 }

struct Board_Info
{
    ushort boardID;
    ushort HWRev;
    ushort FWRev;
    ushort SilabRev;
    uint ipAddress;
    uint Mask;
    uint Gateway;
    ulong MAC;
    byte SN[10];
    bool Bootloader_Flag;
    bool isAdapterMode;
    ADAPTER_TYPE adapterType;
}

struct ConfigurationSettings
{
    double linerate;
    BERTMGR_SIGMODULATION eyeMode;
    bool grayMaping;
    bool preCoding;
    bool chipMode;
    BERTMGR_CLOCKSOURCE clockSource;
    BERTMGR_CLOCKMODE clockType;
    int divider;
    bool FEC;
    BERTMGR_FECMODE FECMode;
    BERTMGR_FECPATTERN FECPattern;
    TAPSMODE Tapsmode ;
    bool IEEEMode;
    bool allTaps[7];

    //Parameters for PRBS pattern configuration
    BERTMGR_PATTERNTYPE txPattern[MAXCHANNELS];
    BERTMGR_PATTERNTYPE rxPattern[MAXCHANNELS];
    bool txInvert[MAXCHANNELS];
    bool rxInvert[MAXCHANNELS];
    bool txEnable[MAXCHANNELS];
    bool rxEnable[MAXCHANNELS];

    //  Parameters for channel's TX amplitude
    int amplitude[MAXCHANNELS];
    AdvancedAmplitude advancedAmplitude[MAXCHANNELS];
    AmpRange amplitudeRange[MAXCHANNELS];
    // Parameters for error insertion

    BERTMGR_ERRORINSERTIONMODES Errormodes[MAXCHANNELS];
    byte duration[MAXCHANNELS];
    byte gap[MAXCHANNELS];
    bool errorState[MAXCHANNELS];

    //  Parameters for DFE mode
    BERTMGR_DSPMODE DSPmode[MAXCHANNELS];

    //  Calibration validation status
    bool calIsValid;

    //  Noise settings
    NoiseSettings noiseSettings;

    //  Shallow loopback
    bool ShallowLoopback;

    //  Enabled FEC links
    ushort FECLinks;

    //  User Defined patterns definitions
    UserDefinedPatternDefinition UserDefinedPattern[MAXCHANNELS];

    //  AFE Trim option
    BERTMGR_AFETRIM_OPT   AFE_Trim;
    bool FECAvailability;
    int MonitorDivider;
    int CDRDivider;
    int CDRSource;
    int CTLE[MAXCHANNELS];
    bool PMenable;
    bool PMRJenable;
    ushort PMamplitude;
    ulong PMfrequency;
    ushort PMRJamplitude;
    ushort PhaseShift;
    ushort PMPRBSamplitude;
    ushort PMdataswing;
    ushort PMpattern;

    bool FMenable;
    bool FMRJenable;
    ushort FMamplitude;
    ulong FMfrequency;
    ushort FMRJamplitude;
    ushort FMShift;
    bool JTOLavailibility;
}

struct EmulatorFECData
{
    bool enabled;
    bool enabledLinks[FECMAXNUMLINKS];   // Enabled link channel
                                         indicator
    bool lockedLinks[FECMAXNUMLINKS];    // Links lock indicator
    uint FEC_CorrectedBitError[FECMAXNUMLINKS];
    uint FEC_BlockCount[FECMAXNUMLINKS];
    uint FEC_SaturatedSymbolError[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CorrectedBitError[FECMAXNUMLINKS];
    ulong AccumulatedFEC_BlockCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_SaturatedSymbolError[FECMAXNUMLINKS];
    SERData SER[FECMAXNUMLINKS];
}

struct ErrorStruct
{
    BERTMGR_ERRORINSERTIONMODES pattern;
    byte gap;
    byte duration;
}

struct FixedPatternDefinition
{
     ulong Pattern;
     byte Repetition;
}

struct HistogramData
{
    uint values[160];
}

struct InstanceParams
{
    char saveConfig[MAX_ADDR_LEN];//clock files path, used for
                                  version < 1.3
    char saveBathtub[MAX_ADDR_LEN]; // Save location of BathTub
    char saveEye[MAX_ADDR_LEN]; // save location of Eye
    int saveBathtubEnable; // Enable BathTub save
    int saveEyeEnable; // Enable Eye save
}

Struct MeasurementsData
{
    BERData berData;                  // BER Channels Measurements
    RealFECData realFecData;          // Real FEC Links Measurements
    EmulatorFECData emulatorFecData; // Emulator FEC Links
                                        Measurements
}

struct NoiseSettings /// Struct for Noise Settings
{
    double NoiseLinerate;
    bool NoiseStatus;
    bool NoiseChannelEnabled[MAXCHANNELS];
    int NoiseLevel[MAXCHANNELS];
    BERTMGR_PATTERNTYPE txPatternNoise[MAXCHANNELS];
    BERTMGR_SIGMODULATION NoiseeyeMode;
     UserDefinedPatternDefinition
    NoiseUserDefinedPattern[MAXCHANNELS];
}

struct PatternConfig
{
    BERTMGR_PATTERNTYPE pattern;
    bool invert;
    ulong userDefined[2];
    int repetition;
}

struct RealFECData
{
    bool enabled;
    bool enabledLinks[FECMAXNUMLINKS];  //Links enabled indicator
    bool lockedLinks[FECMAXNUMLINKS];   // Links lock indicator
    double Time[FECMAXNUMLINKS];        // Constructed time data
    ulong BitCount[FECMAXNUMLINKS];     // Bit Count data
    uint FEC_Skew[FECMAXNUMLINKS];
    uint FEC_Corrected_Ones_Interval[FECMAXNUMLINKS];
    uint FEC_Corrected_Zeros_Interval[FECMAXNUMLINKS];
    ulong FEC_ErrorCount_Interval[FECMAXNUMLINKS];
    uint FEC_Symbol_ErrorCount_Interval[FECMAXNUMLINKS];
    uint FEC_CorrectedBitCount_Interval[FECMAXNUMLINKS];
    double FEC_Symbol_ErrorRate_Interval[FECMAXNUMLINKS];
    double FEC_CorrectedBitRate_Interval[FECMAXNUMLINKS];
    double FEC_Frame_ErrorRate_Interval[FECMAXNUMLINKS];
    uint FEC_CW_UnCorrectedCount_Interval[FECMAXNUMLINKS];
    uint FEC_CW_CorrectedCount_Interval[FECMAXNUMLINKS];
    uint FEC_CW_ProcessedCount_Interval[FECMAXNUMLINKS];
    double FEC_CW_UncorrectedErrorRate_Interval[FECMAXNUMLINKS];
    ulong AccumulatedFEC_Corrected_Ones[FECMAXNUMLINKS];
    ulong AccumulatedFEC_Corrected_Zeros[FECMAXNUMLINKS];
    ulong AccumulatedFEC_ErrorCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_Symbol_ErrorCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CorrectedBitCount[FECMAXNUMLINKS];
    double AveragedFEC_Symbol_ErrorRate[FECMAXNUMLINKS];
    double AveragedFEC_CorrectedBitRate[FECMAXNUMLINKS];
    double AveragedFEC_Frame_ErrorRate[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_UnCorrectedCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_CorrectedCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_ProcessedCount[FECMAXNUMLINKS];
    double AccumulatedFEC_CW_ncorrectedErrorRate[FECMAXNUMLINK];
    SERData SER[FECMAXNUMLINKS];
    ulong TotalBitCount[MAXCHANNELS]; // Total Bit Count data MSB +
                                         LSB

}
struct RealFECData_4044
{
    bool enabled;
    bool enabledLinks[FECMAXNUMLINKS];  // Links enabled indicator
    bool lockedLinks[FECMAXNUMLINKS];   // Links lock indicator
    double Time[FECMAXNUMLINKS];        // Constructed time data
    ulong BitCount[FECMAXNUMLINKS];     // Bit Count data
    uint FEC_CorrectedBitCount_Interval[FECMAXNUMLINKS];
    uint FEC_CW_UnCorrectedCount_Interval[FECMAXNUMLINKS];
    uint FEC_CW_CorrectedCount_Interval[FECMAXNUMLINKS];
    uint FEC_CW_ProcessedCount_Interval[FECMAXNUMLINKS];
    double FEC_CW_UncorrectedErrorRate_Interval[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_UnCorrectedCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_CorrectedCount[FECMAXNUMLINKS];
    ulong AccumulatedFEC_CW_ProcessedCount[FECMAXNUMLINKS];
    double AccumulatedFEC_CW_UncorrectedErrorRate[FECMAXNUMLINKS];
    SERData SER[FECMAXNUMLINKS];
    ulong TotalBitCount[FECMAXNUMLINKS];// Total Bit Count data
```

```
};

struct SERData
{
    int nSymbols;
    uint InstantSER[SERMAXNUMSYMBOLS];
    ulong AccumulatedSER[SERMAXNUMSYMBOLS];
}

struct UserDefinedPatternDefinition
{
    FixedPatternDefinition Pattern1;
    FixedPatternDefinition Pattern2;
}
```

Additional Struct and Enumeration Definitions for Host Module:

```
enum ADAPTER_EXTERNALMODE

{
   ADAPTER_EXTERNALMODE_DISABLED = 0,
   ADAPTER_EXTERNALMODE_HW_ENABLED,
   ADAPTER_EXTERNALMODE_SW_ENABLED
}

enum ADAPTER_HWSIGNAL_CNTRL

{
   ADAPTER_HWSIGNAL_CNTRL_QDD_MODSEL_L = 0,
   ADAPTER_HWSIGNAL_CNTRL_QDD_RESET_L,
   ADAPTER_HWSIGNAL_CNTRL_QDD_INITMODE,
   ADAPTER_HWSIGNAL_CNTRL_QSFP_MODSEL_L,
   ADAPTER_HWSIGNAL_CNTRL_QSFP_RESET_L,
   ADAPTER_HWSIGNAL_CNTRL_QSFP_LPMODE,
   ADAPTER_HWSIGNAL_CNTRL_OSFP_LPWn,
   ADAPTER_HWSIGNAL_CNTRL_OSFP_RSTn
}

enum ADAPTER_TYPE

{
   ADAPTER_TYPE_UNDETECTED = -1,
   ADAPTER_TYPE_NOADAPTER = 0,
   ADAPTER_TYPE_QDD,
   ADAPTER_TYPE_OSFP,
   ADAPTER_TYPE_QSFP,
   ADAPTER_TYPE_SFP,
   ADAPTER_TYPE_CFP2,
   ADAPTER_TYPE_SFP_DD,
}

enum TXVR_RX_AMPLITUDE

{
   TXVR_RX_AMPLITUDE_100_400 = 0,
   TXVR_RX_AMPLITUDE_300_600,
   TXVR_RX_AMPLITUDE_400_800,
   TXVR_RX_AMPLITUDE_600_1200,
   TXVR_RX_AMPLITUDE_RESERVED,
   TXVR_RX_AMPLITUDE_CUSTOM
}
```

enum TXVR_MSA_PAGE

```
{
    TXVR_MSA_PAGE_LOWERMEMORY = 0,
    TXVR_MSA_PAGE_0,
    TXVR_MSA_PAGE_1,
    TXVR_MSA_PAGE_2,
    TXVR_MSA_PAGE_3,
    TXVR_MSA_PAGE_16,
    TXVR_MSA_PAGE_17
}
```

struct TXVR_ConfigurationSettings

```
{
    bool DataPathDeInit[MAXCHANNELS];
    bool TXOuputDisable[MAXCHANNELS];
    bool TXPolarityFlip[MAXCHANNELS];
    bool TXSquelchDisable[MAXCHANNELS];
    bool TXForceSquelch[MAXCHANNELS];
    byte TXEqualization[MAXCHANNELS];
    bool RXOutputDisable[MAXCHANNELS];
    bool RXPolarityFlip[MAXCHANNELS];
    bool RXSquelchDisable[MAXCHANNELS];
    TXVR_RX_AMPLITUDE RXOutputAmplitude[MAXCHANNELS];
    byte RXOutputPreCursor[MAXCHANNELS];
    byte RXOutputPostCursor[MAXCHANNELS];
```

Function Definitions:

When implementing ThunderBERT functions, please refer to the General Flows section for proper function order and execution.

mlbertmgr* mlbertmgr_createInstance()

Description:
Creates a new BERT API instance. It is recommended that each device should have its own instance. BERT instance must be created in order to connect to and control a BERT (this function must be run before other functions such as openConnection and initializeInstance).

Inputs:
None.

Outputs:
Pointer to created BERT API instance.

BERTMGR_STATUS mlbertmgr_openConnection(mlbertmgr * inst, char * address)

Description:
Connects to the BERT board using an IP address (IP, PXI resource name). This function should be run after creating an instance for the BERT.

Inputs:
inst: pointer to instance.
address: the board IP address.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Main Flow.

`BERTMGR_STATUS mlbertmgr_initializeInstance``(mlbertmgr * inst, InstanceParams t_params)`

Description:
Initializes the instance. BERT instance should be initialized after the connection is opened using mlbertmgr_openConnection().
This API is used to configure the location of the bathtub curve and eye report.

Inputs:
inst: pointer to instance.
t_params: instance parameters `InstanceParams`.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Main Flow.

`BERTMGR_STATUS mlbertmgr_closeConnection` `(mlbertmgr * inst)`

Description:
Closes connection to the instance. Connection to instance can only be closed before destroying the instance using mlbertmgr_destroyInstance().

Inputs:
inst: pointer to instance.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Main Flow.

`void mlbertmgr_destroyInstance` `(mlbertmgr* inst)`

Description:
Destroys the BERT API Instance after closing the connection using mlbertmgr_close connection()

Inputs:
inst: pointer to instance.

Outputs:
None.

Example:
Used in Main Flow.

BERTMGR_STATUS mlbertmgr_applyConfiguration(mlbertmgr * inst)

Description:
Applies the current configuration parameters.

Inputs:
inst: pointer to instance.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_captureHistogramData (mlbertmgr * inst, ushort enabledChannels, ushort* actualEnabled)

Description:
Requests a histogram capture for enabled channels. This is a no blocking mode API call.

Inputs:
inst: pointer to instance.
enabledChanels: enabled channel flags (1 bit/channel).

Outputs:
actualEnabled: reference to enabled channel flags(1bit/channel) output.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 8.

BERTMGR_STATUS mlbertmgr_configureFECLinks(mlbertmgr * inst, ushort channels, bool applyConfig)

Description :
Configures FEC links channels. Refer to the feature support table for available FEC options.

Inputs:
inst: pointer to instance.
channels: 16-bits flags for each channel. To enable a channel set its bit to 1, 0 otherwise.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 10.

BERTMGR_STATUS mlbertmgr_enableFMRJ(mlbertmgr * inst, bool enable, bool applyConfig)

Description:
Enables FM RJ injection.

Inputs:
inst: pointer to instance.
enable: enable/disable FM RJ.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_enableFMSJ(mlbertmgr * inst, bool enable, bool applyConfig)

Description:
Enables FM SJ injection.

Inputs:
inst: pointer to instance.
enable: enable/disable FM SJ.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_enableMonitor(mlbertmgr * inst, int enabledFlagsValue)

Description:
Set enabled monitoring flags, Refer to BERTMGR_MONITOR_FLAGS enum for bits order.

Inputs:
inst: pointer to instance.
enabledFlagsValue: monitoring flags setter `BERTMGR_MONITOR_FLAGS`.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 7.

`BERTMGR_STATUS mlbertmgr_enableMonitorFlag``(mlbertmgr * inst, MONITOR_FLAGS flag, bool isEnabled)`

Description:
Sets individual monitoring flag status.

Inputs:
inst: pointer to instance.
flag: monitoring flag BERTMGR_MONITOR_FLAGS.
isEnabled: enable status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 7.

`BERTMGR_STATUS mlbertmgr_enableNoise``(mlbertmgr * inst, int channel, bool enable, bool applyConfig)`

Description:
Enable/disable noise injection.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
enable: enable/disable noise injection.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_enablePMRJ``(mlbertmgr * inst, bool enable, bool applyConfig)`

Description:
Enables PM RJ injection.

Inputs:
inst: pointer to instance.
enable: enable/disable PM RJ.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_enablePMSJ``(mlbertmgr * inst, bool enable, bool applyConfig)`

Description:
Enables PM SJ injection.

Inputs:
inst: pointer to instance.
enable: enable/disable PM SJ.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_getActiveConfig``(mlbertmgr * inst, ConfigurationSettings* initConfig)`

Description:
Gets the active configurations on the BERT.

Inputs:
inst: pointer to instance.
initConfig: board configuration parameters `ConfigurationSettings`.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 6.

`BERTMGR_STATUS mlbertmgr_getAvailableBERData``(mlbertmgr * inst, MeasurementsData data[BERMAXITEMSPOP], int &datacount)`

Description:
Gets available BER data.

Inputs:
inst pointer to instance.

Outputs:
datacount: reference to the number of captured data.
data: reference to accumulated BER Data `MeasurementsData`.
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 9.

BERTMGR_STATUS mlbertmgr_getCDRLock(mlbertmgr * inst, int channel, bool& lock)

Description:
Gets CDR lock status.

Inputs:
inst: pointer to instance.
Channel: 0-based index of channel.
Lock: returned CDR lock status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

BERTMGR_STATUS mlbertmgr_getClockOut(mlbertmgr * inst, double * clockOutRate)

Description:
Gets the clock output frequency in MHz. Not Implemented.

Inputs:
inst: pointer to instance.
clockOutRate: pointer to clock out rate.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

BERTMGR_STATUS mlbertmgr_getGrayCoding(mlbertmgr * inst, bool* isEnabled))

Description:
Reads Gray coding status.

Inputs:
inst: pointer to instance.
isEnabled: reference to gray coding status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_getHistogramData(mlbertmgr * inst, ushort enabledChannels, HistogramData output[])

Description:
Gets histogram data for enabled channels.

Inputs:
inst: pointer to instance.
enabledChannels: enabled channels flag (1 bit/channel).

Outputs:
output: reference to channel's HistogramData.
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_getInfo(mlbertmgr * inst, Board_Info* info)

Description:
Returns board information such as IP, MAC, Revision, Gateway, Mask, SN, and board ID.

Inputs:
inst: pointer to instance.

Outputs:
info: reference to the board info Board_Info.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 1.

BERTMGR_STATUS mlbertmgr_getRxStatus(mlbertmgr * inst, int channel, bool * isEnabled)

Description:
Read the state of Rx whether it is enabled or not.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.

Outputs:
isEnabled: reference to the status of the Rx line.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 5.

`BERTMGR_STATUS mlbertmgr_getTxEmulationTapsFromLossAtNyquist``(mlbertmgr * inst, int* taps, double lossDb)`

Description:
Calculates Tx Emulation Taps from Loss at Nyquist.

Inputs:
inst: Pointer to instance.
lossDb: dB loss value at Nyquist.

Outputs:
taps: reference to the calculated taps.
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_getTxEmulationTapsFromSParams``(mlbertmgr * inst, int* taps, char s2pFilePath[255])`

Description:
Calculates Tx Emulation Taps from S-parameter file.

Inputs:
inst: Pointer to instance.
s2pFilepath [255]: directory path of the s2p file.

Outputs:
taps: reference to the calculated taps.
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_getTxStatus``(mlbertmgr * inst, int channel, bool * isEnabled)`

Description:
Read the state of Tx whether it is enabled or not.

Inputs:
inst: Pointer to instance.
channel: 0-based index of channel.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.
IsEnabled: reference to the status of the Tx line.

Example:
Used in Test Flow 5.

BERTMGR_STATUS mlbertmgr_loadCalibrationValues(mlbertmgr * inst, int channel, int mode, double * Data, int* lenData, bool applyConfig)

Description:
Loads calibration values.

Inputs:
inst: Pointer to instance.
channel: 0-based index of channel.
mode: calibration mode.
applyConfig: (not implemented).

Outputs:
Data: reference to the calibration values.
lenData: length of the output calibration data.
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_loadOptimalSettings(mlbertmgr * inst, int channel, int mode, int * Data, int* lenData, bool applyConfig)

Description:
Loads Optimal Settings.

Inputs:
inst: Pointer to instance.
channel: 0-based index of channel.
mode: optimal settings mode .
applyConfig: (not implemented).

Outputs:
Data: reference to the optimal settings.
lenData: length of the output optimal settings.
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_multiReadMonitor(mlbertmgr * inst, int enabledFlagsValue, ushort values[])

Description:
Reading all enabled monitoring values in the following order.
- **-LOS** requires 3 ushort.
- **-DSP** requires 1 ushort/ channel.
- **-SIGNALDETECT** requires 1 ushort/ channel.
- **-TXLOCK** requires 1 ushort/ channel.
- **-RXLOCK** requires 1 ushort/ channel.
- **-TEMPERATURE** requires 4 ushort.
- **-SNR** requires 1 ushort/ channel.

-VOLTAGE requires 1 ushort/ channel.
-CURRENT requires 1 ushort/ channel.
-FFETAPS requires 16 ushort values/ channel.
-MONITOR_XT_TXLOCK requires 1 ushort/ channel.
-MONITOR_ADAPTER requires 26 ushort.
-TRANSCEIVER requires 80 ushort values.

Inputs:
inst: pointer to instance.
enabledFlagsValue: monitoring flags setter `BERTMGR_MONITOR_FLAGS.`

Outputs:
value: reference to multiple monitor flag values for all channels.
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 7 and Test Flow 13

`BERTMGR_STATUS mlbertmgr_readHistogramData``(mlbertmgr * inst, int channel, HistogramData* output)`

Description:
Reads channel histogram data. Must be called after a capture request: mlbertmgr_CaptureHistogramData.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.

Outputs:
output: reference to channel's `HistogramData`.
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 8.

`BERTMGR_STATUS mlbertmgr_readLOS``(mlbertmgr * inst, ushort &value)`

Description:
Gets LOS monitor flag status.

Inputs:
inst pointer to instance.

Outputs:
value: Reference to loss of signal monitor flag status
Returns an attribute of the `BERTMGR_STATUS` Enum.

BERTMGR_STATUS mlbertmgr_RxEnable(mlbertmgr * inst, int channel, bool status)

Description:
Enables/Disables the Rx line.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
status: status of the Rx line.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 5.

BERTMGR_STATUS mlbertmgr_setActiveConfig(mlbertmgr * inst, ConfigurationSettings initConfig, bool forceUpdate)

Description:
Initializes the board's settings using a single API call.

Inputs:
inst: pointer to instance.
initConfig: Configuration Settings ConfigurationSettings.
ForceUpdate: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setAdvancedAmplitude (mlbertmgr * inst, int channel, AdvancedAmplitude advAmplitude, int *output, bool applyConfig)

Description:
Sets advanced amplitude for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
advAmplitude: advanced amplitude values AdvancedAmplitude.
applyConfig: (not implemented).

Outputs:
output: reference to calculated approximate amplitude.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 4.

BERTMGR_STATUS mlbertmgr_setAFETrim(mlbertmgr * inst, BERTMGR_AFETRIM_OPT value, bool applyConfig)

Description:
Sets AFE Trim option.

Inputs:
inst: pointer to instance.
value: AFE Trim option value BERTMGR_AFETRIM_OPT.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setAmplitude (mlbertmgr * inst, int channel, int amplitude, bool applyConfig)

Description:
Sets the peak-to-peak amplitude in mV.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
amplitude: the amplitude value in mV.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 3.

BERTMGR_STATUS mlbertmgr_setCDRChannelSource(mlbertmgr * inst, int option, bool applyConfig)

Description:
Sets CDR channel source.

Inputs:
inst: pointer to instance.
optional: CDR channel source.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setCDRDivider (mlbertmgr * inst, BERTMGR_CDRDIVIDER divider, bool applyConfig)

Description:
Sets CDR clock divider.

Inputs:
inst: pointer to instance.
divider: CDR divider value from BERTMGR_CDRDIVIDER.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 2.

BERTMGR_STATUS mlbertmgr_setClockMode(mlbertmgr * inst, BERTMGR_CLOCKMODE clockMode, bool applyConfig)

Description:
Sets the output clock mode of the BERT.

Inputs:
inst: pointer to instance.
clockMode: clock mode BERTMGR_CLOCKMODE.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 2.

BERTMGR_STATUS mlbertmgr_setClockSource (mlbertmgr * inst, BERTMGR_CLOCKSOURCE clockSource, bool applyConfig)

Description:
Sets the clock source to either Internal or External.

Inputs:
inst: pointer to instance.
clockSource: clock source BERTMGR_CLOCKSOURCE.

applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 2.

BERTMGR_STATUS mlbertmgr_setCTLE(mlbertmgr * inst,int channel, int CTLE, bool applyConfig)

Description:
Sets CTLE for selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
CTLE: CTLE value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setDSPmode(mlbertmgr * inst, int channel, BERTMGR_DSPMODE DSPmode, bool applyConfig)

Description:
Sets Rx channel equalizer mode.

Inputs:
inst: Pointer to instance.
channel: 0-based index of channel.
DSPmode: Rx equalizer mode from BERTMGR_DSPMODE Enum.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 4.

BERTMGR_STATUS mlbertmgr_setErrorPattern(mlbertmgr * inst, int channel, ErrorStruct error, bool applyConfig)

Description:
Sets error insertion pattern:BERTMGR_ERRORINSERTIONMODES, gap, duration.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
ErrorStruct: error insertion parameter `ErrorStruct`.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

BERTMGR_STATUS mlbertmgr_setErrorRate`(mlbertmgr * inst, int channel, double rate, double* actualrate, bool applyConfig)`

Description:
Sets error rate in million error/s.

Inputs:
- inst: pointer to instance.
- channel: 0-based index of channel.
- rate: rate in Gbps.
- applyConfig: (not implemented).

Outputs:
- Returns an attribute of the `BERTMGR_STATUS` Enum.
- actualrate: reference to the calculated actual rate

BERTMGR_STATUS mlbertmgr_setEyeMode`(mlbertmgr * inst, BERTMGR_SIGMODULATION eyeMode, bool applyConfig)`

Description:
Sets the eye mode to either NRZ or PAM4.

Inputs:
inst: pointer to instance.
eyeMode: eye mode `BERTMGR_SIGMODULATION`.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 3.

BERTMGR_STATUS mlbertmgr_setFECMode`(mlbertmgr * inst, BERTMGR_FECMODE mode, BERTMGR_FECPATTERN pattern ,bool applyConfig)`

Description:
Set FEC mode. Refer to the feature support table for available FEC Modes.

Inputs:
inst: pointer to instance.
mode: the FEC mode BERTMGR_FECMODE.
Pattern: the FEC pattern BERTMGR_FECPATTERN.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 10.

BERTMGR_STATUS mlbertmgr_setFMFrequency (mlbertmgr * inst, ushort frequency, bool applyConfig)

Description:
Sets the FM frequency.

Inputs:
inst: pointer to instance.
frequency: FM frequency in kHz.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setFMPhaseShift (mlbertmgr * inst, ushort value, bool applyConfig)

Description:
Sets the PM phase shift.

Inputs:
inst: pointer to instance.
value: FM phase shift digital value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setFMRJAmplitude (mlbertmgr * inst, ushort amplitude, bool applyConfig)

Description:
Sets the FM RJ digital amplitude.

Inputs:
inst: pointer to instance.
amplitude: FM RJ digital amplitude.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setFMSJAmplitude_ps (mlbertmgr * inst, ushort amplitude, bool applyConfig)`

Description:
Sets the FM SJ calibrated amplitude.

Inputs:
inst: pointer to instance.
amplitude: FM SJ calibrated amplitude (ps).
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setGrayCoding(mlbertmgr * inst, bool enable, bool applyConfig)`

Description:
Set Gray Coding for PAM4 signal mode.

Inputs:
inst: pointer to instance.
enable: gray coding enabling status.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 3.

`BERTMGR_STATUS mlbertmgr_setInnerEyeLevel(mlbertmgr * inst, int channel, int innerLevel, bool applyConfig)`

Description:
Sets the inner eye level for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
innerLevel: the inner level value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setLinerate(mlbertmgr * inst, double * linerate, bool applyConfig)`

Description:
Applies the linerate to the BERT. Refer to the table of feature support for available line rates.

Inputs:
inst: pointer to instance.
linerate: linerate in Gbps.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 3.

`BERTMGR_STATUS mlbertmgr_setMainTap(mlbertmgr * inst, int channel, int mainTap, bool applyConfig)`

Description:
Sets the main tap for selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
mainTap: main tap value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setMonitorDivider (mlbertmgr * inst, int divider, bool applyConfig)`

Description:
Sets the output clock Monitor divider.

Inputs:
inst: pointer to instance.
divider: divider value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 2.

BERTMGR_STATUS mlbertmgr_setNoiseAmplitude_mV(mlbertmgr * inst, int channel, int value, bool applyConfig)

Description:
Sets calibrated noise amplitude in mV

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
value: noise amplitude (mV)
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setNoiseBurstRate(mlbertmgr * inst, int channel, double burstRate, double* actualrate, bool applyConfig)

Description:
Sets noise burst rate.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
burstRate: the burst rate.
applyConfig: (not implemented).

Outputs:
actualrate: reference to the calculated actual rate.
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setNoiseLevel(mlbertmgr * inst, int channel, int NoiseLevel, bool applyConfig)

Description:
Sets noise level.

Inputs:
inst: pointer to the instance.
channel: 0-based index of channel.
NoiseLevel: noise level.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setNoiseLinerate(mlbertmgr * inst, double * linerate, bool applyConfig)

Description:
Sets noise linerate.

Inputs:
inst: pointer to instance.
linerate: linerate in Gbps.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setNoiseStatus(mlbertmgr * inst, bool enable, bool applyConfig)

Description:
Sets noise status on all channels.

Inputs:
inst: pointer to instance.
enable: enable noise injection on all channels.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setNoiseTxPattern(mlbertmgr * inst, int channel, PatternConfig txPattern, bool applyConfig)

Description:
Sets TX pattern for noise.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
txPattern: Tx pattern type PatternConfig.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_setOuterEyeLevel``(mlbertmgr * inst, int channel, int outerLevel, bool applyConfig)`

Description:
Sets the outer eye level for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
outerLevel: the outer level value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_setPMBUJAmplitude` `(mlbertmgr * inst, ushort amplitude, bool applyConfig)`

Description:
Sets the PM BUJ amplitude.

Inputs:
inst: pointer to instance.
amplitude: PM BUJ digital amplitude.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_setPMFrequency` `(mlbertmgr * inst, ushort frequency, bool applyConfig)`

Description:
Sets the PM frequency value.

Inputs:
inst: pointer to instance.
frequency: PM frequency value in kHz.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setPMPhaseShift (mlbertmgr * inst, ushort value, bool applyConfig)

Description:
Sets the PM phase shift value.

Inputs:
inst: pointer to instance.
value: phase shift digital value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setPMRJAmplitude (mlbertmgr * inst, ushort amplitude, bool applyConfig)

Description:
Sets the PM RJ amplitude value.

Inputs:
inst: pointer to instance.
amplitude: PM RJ digital amplitude.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setPMSJAmplitude_ps (mlbertmgr * inst, int amplitude, bool applyConfig)

Description:
Sets the PM RJ amplitude values in ps.

Inputs:
inst: pointer to instance.
amplitude: PM amplitude in ps.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setPostEmphasis(mlbertmgr * inst, int channel, int postEmphasis, bool applyConfig)

Description:
Sets the post emphasis for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
postEmphasis: post emphasis value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setPreEmphasis``(mlbertmgr * inst, int channel, int preEmphasis, bool applyConfig)`

Description:
Sets the pre-emphasis for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
preEmphasis: the pre-emphasis value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

`BERTMGR_STATUS mlbertmgr_setRxPattern``(mlbertmgr * inst, int channel, PatternConfig rxPattern, bool applyConfig = false)`

Description:
Sets the RX pattern. Refer to the table of feature support for available Rx Patterns.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
rxPattern: Rx pattern type PatternConfig.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 3.

`BERTMGR_STATUS mlbertmgr_setScalingLevel``(mlbertmgr * inst, int channel, int scalingLevel, bool applyConfig)`

Description:
Sets scaling level for the selected channel.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
scalingLevel: scaling level value.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setShallowLoopback(mlbertmgr * inst, bool enable, bool applyConfig)

Description:
Sets shallow Loopback. Not implemented.

Inputs:
inst: pointer to instance.
enable: enable/disable shallow loopback.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_setTapsMode(mlbertmgr * inst, TAPSMODE mode, bool applyConfig)

Description:
Sets Tx linear FFE taps mode to either 3 taps or 7 taps.

Inputs:
inst: pointer to instance.
mode: taps mode from BERTMGR_TAPSMODE Enum.
applyConfig: trigger the configuration on the instrument, otherwise the parameters are stored in BERT memory and applied once a new trigger occurs.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 3.

BERTMGR_STATUS mlbertmgr_setTxPattern(mlbertmgr * inst, int channel, PatternConfig txPattern, bool applyConfig)

Description:
Sets the TX pattern. Refer to the table of feature support for available Tx Patterns.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
txPattern: Tx pattern type `PatternConfig`.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 3.

`BERTMGR_STATUS mlbertmgr_setUserDefinedPattern` `(mlbertmgr * inst, int channel, UserDefinedPatternDefinition userDefinedPattern, bool applyConfig)`

Description:
Allows the user to define a specific pattern.

Inputs:
inst: pointer to instance.
Channel: 0-based index of channel.
userDefinedPattern: object holding the pre-defined pattern.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

`BERTMGR_STATUS mlbertmgr_singleReadMonitor``(mlbertmgr * inst, MONITOR_FLAGS flag, ushort value[])`

Description:
Reads individual monitoring enabled using mlbertmgr_enableMonitor or mlbertmgr_enableMonitorFlag.

Inputs:
inst: pointer to instance.
flag: monitoring flag `BERTMGR_MONITOR_FLAGS`.

Outputs:
value: reference to monitor flag value for all channels.
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 7 and Test Flow 13.

void mlbertmgr_startBER(mlbertmgr * inst, ushort channels, bool accumulate)

Description:
Starts continuous BER capture. The time interval between continuous captures is arround 100 ms. A BER stabilization process is implemented in the ML4054B to ensure BER stabilization and repetitive measurements. This process takes about 4 seconds before the BER counter is ready, and therefore the BER count time should be greater than 4 seconds.

Inputs:
inst: pointer to instance.
channels: 16-bits flags. To enable a channel set its corresponding bit to 1 and 0 otherwise.
accumulate: enable accumulate BER Data.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 9.

BERTMGR_STATUS mlbertmgr_stopBER(mlbertmgr * inst)

Description:
Stops the BER acquisition.

Inputs:
inst: pointer to instance.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 9.

BERTMGR_STATUS mlbertmgr_stopErrorInsertion(mlbertmgr * inst, int channel, bool applyConfig)

Description:
Stops error insertion for the continuous injection mode.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
applyConfig: (not implemented).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

BERTMGR_STATUS mlbertmgr_TxEnable(mlbertmgr * inst, int channel, bool status)

Description:
Enables/Disables the Tx line.

Inputs:
inst: pointer to instance.
channel: 0-based index of channel.
status: status of the Tx line.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 5.

Additional functions for module Host:

BERTMGR_STATUS mlbertmgr_detectAdapter(mlbertmgr * inst, ADAPTER_TYPE * type)

Description:
Reads adapter type.

Inputs:
inst: pointer to instance.
type: pointer to an ADAPTER_TYPE.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.
Type: type of the module host adapter.

Example:
Used in Test Flow 11.

BERTMGR_STATUS mlbertmgr_setControlPin(mlbertmgr * inst, ADAPTER_HWSIGNAL_CNTRL cntrl, bool status)

Description:
Sets adapter control pin.

Inputs:
inst: pointer to instance.
cntrl: pin control selection ADAPTER_HWSIGNAL_CNTRL.
status: pin status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Used in Test Flow 11.

BERTMGR_STATUS mlbertmgr_setExternalAdapterMode(mlbertmgr * inst, bool isEnabled)

Description:
Sets external adapter mode status.

Inputs:
inst: pointer to instance.
isEnabled: external mode enabler.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Used in Test Flow 11.

BERTMGR_STATUS mltxvr_getActiveConfig(mlbertmgr * inst, TXVR_ConfigurationSettings* activeConfig)

Description:
Reads transceiver active confguration.

Inputs:
inst: pointer to instance.

Outputs:
activeConfig: Return the active configuration TXVR_ConfigurationSettings.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxAmplitude(mlbertmgr * inst, int channel, TXVR_RX_AMPLITUDE value)

Description:
Sets transceiver Rx amplitude.

Inputs:
inst: pointer to instance.
channel: channel selection.
value: Rx amplitude range TXVR_RX_AMPLITUDE.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxOutputDisable(mlbertmgr * inst, int channel, bool status)

Description:
Sets Rx Output Disable status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: Rx disable status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxPolarityFlip(mlbertmgr * inst, int channel, bool status)

Description:
Sets Rx polarity flip status.

Inputs:
inst: pointer to instance.
channel: channel selection.
status: Rx polarity flip status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxPostCursor(mlbertmgr * inst, int channel, int value)

Description:
Sets Rx Post-Cursor value. According to CMIS, the range of values is from 0 to 7.

Inputs:
inst: pointer to instance.
channel: channel Selection.
value: Rx Post-Cursor value.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxPreCursor(mlbertmgr * inst, int channel, int value)

Description:
Sets Rx Pre-Cursor value. According to CMIS, the range of values is from 0 to 7.

Inputs:
inst: pointer to instance.
channel: channel selection.
value: Rx Pre-Cursor value.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setRxSquelchDisable(mlbertmgr * inst, int channel, bool status)

Description:
Sets Rx squelch disable status.

Inputs:
inst: pointer to instance.
channel: channel selection.
status: Rx squelch disable status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 12.

`BERTMGR_STATUS mltxvr_setTxDataPathDeInit``(mlbertmgr * inst, int channel, bool Status)`

Description:
Sets Tx DataPathDeInit status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: DataPathDeInit status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 12.

`BERTMGR_STATUS mltxvr_setTxForceSquelch``(mlbertmgr * inst, int channel, bool status)`

Description:
Sets Tx force squelch status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: Tx force squelch status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 12.

`BERTMGR_STATUS mltxvr_setTxInputEqualization``(mlbertmgr * inst, int channel, int value)`

Description:
Sets Tx input equalization value. According to CMIS, the range of values is from 0 to 12.

Inputs:
inst: pointer to instance.
channel: channel selection.
value: Input Equalization value.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setTxOutputDisable(mlbertmgr * inst, int channel, bool status)

Description:
Sets Tx output disable status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: Tx disable status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setTxPolarityFlip(mlbertmgr * inst, int channel, bool Status)

Description:
Sets Tx polarity flip status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: Tx polarity flip status.

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

BERTMGR_STATUS mltxvr_setTxSquelchDisable(mlbertmgr * inst, `int channel, bool status)`

Description:
Sets Tx squelch disable status.

Inputs:
inst: pointer to instance.
channel: channel selection.
Status: Tx squelch disable status.

Outputs:
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 12.

`BERTMGR_STATUS mltxvr_getMSAValues``(mlbertmgr * inst, TXVR_MSA_PAGE pages[], ushort values[], byte numberOfPages)`

Description:
Reads Transceiver MSA values.

Inputs:
inst: pointer to instance.
pages: pages to read.

Outputs:
values: MSA values. Number of pages to read x 128.
Returns an attribute of the `BERTMGR_STATUS` Enum.

Example:
Used in Test Flow 12.

`BERTMGR_STATUS mltxvr_sequentialRead``(mlbertmgr * inst, ushort pageSelect, ushort registerAddress, ushort dataLength, ushort* dataBuffer, ushort bankSelect = 0)`

Description:
Transceiver I2C/MDIO sequential read.

Inputs:
inst: pointer to instance.
pageSelect: page selection.
registerAddress: address to start reading from.
dataLength: length of data to be read.
bankSelect: bank selection (default = 0).

Outputs:
dataBuffer: returned data.
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

```
BERTMGR_STATUS mltxvr_sequentialWrite(mlbertmgr * inst, ushort
pageSelect, ushort registerAddress, ushort dataLength, ushort*
dataBuffer, ushort bankSelect = 0)
```

Description:
Transceiver I2C/MDIO Sequential write.

Inputs:
inst: pointer to instance.
pageSelect: page selection.
registerAddress: address to start writing to.
dataLength: length of data to be written.
dataBuffer: data to write.
bankSelect: bank selection (default = 0).

Outputs:
Returns an attribute of the BERTMGR_STATUS Enum.

Example:
Used in Test Flow 12.

Feature Support

This section indicates which functions are supported across the various ThunderBERT platforms. As the ThunderBERT API covers BERTs with different core use cases and functional capabilities, the following table will serve as a reference point to associate specific functions with the supported BERTs.

BERT Parameter	ML4039B	ML4054B	ML4039D	ML4079D	ML4039E ML4039E-ATE	ML4039EN	ML4079E	ML4079EN
Channel	4	8	4	8	4	4	8	8
Data Rate (GBaud NRZ/PAM4)	1.12 - 1.54 (NRZ) 2.24 – 6.1 (NRZ) 6.5 -6.9 (NRZ) 7 -29 (Both)	1.12 - 1.54 (NRZ) 2.24 - 6.1 (NRZ) 2.24 - 29 (Both)	9 - 14.2 (NRZ) 22 - 29.5 (Both)	9 - 14.2 (NRZ) 22 - 29.5 (Both)	23 - 29 (Both) 46 – 58 (Both)	23 - 29 (Both) 46 - 58 (Both)	23 - 29 (Both) 46 - 58 (Both)	20-29 (both) 36-61 (both)
TX Pattern	PRBS 7/9/11/13 /15/23/31 /58/9_4, JP03B, IEEE 802.3bs, OIF-CEI-3.1, User defined	PRBS 7/9/11/13 /15/23/31 /58/9_4 JP03B, IEEE 802.3bs, OIF-CEI-3.1 User defined	PRBS 7/9/11/13 /15/16/23 /31/58 /9_4, JP03B, CJT, LIN, SSPRQ, User Defined	PRBS 7/9/11/13 /15/16/23 /31/58 /9_4, JP03B, CJT, LIN, SSPRQ, User Defined	PRBS 7/9/11/13 /15/16/23/31 /58/9_4 SQ16, SQ32, LIN, CJT, SSPRQ, User Defined, JP083B	PRBS 7/9/11/13 /15/16/23/31 /58/9_4 SQ16, SQ32, LIN, CJT, SSPRQ, User Defined, JP083B	PRBS 7/9/11/13 /15/16/23/31 /58/9_4 SQ16, SQ32, LIN, CJT, SSPRQ, User Defined, JP083B	PRBS 7/9/11/13 /15/16/23/31 /58/9_4 SQ16, SQ32, LIN, CJT, SSPRQ, User Defined, JP083B
RX Pattern	PRBS 7/9/11/13 /15/23/31	PRBS 7/9/11/13 /15/23/31	PRBS 7/9/11/13 /15/16/23/31	PRBS 7/9/11/13 /15/16/23/31	PRBS 7/9/11/13 /15/16/23/31	PRBS 7/9/11/13 /15/16/23/31	PRBS 7/9/11/13 /15/16/23/31	PRBS 7/9/11/13 /15/16/23/31
SNR/ Histogram	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Error injection	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Clock OUT	Reference, Monitor, External	Reference, Monitor Tx PLL	Reference, Monitor	Reference, Monitor	Reference, Monitor	Reference, Monitor	Reference, Monitor	Reference, Monitor
External clock IN	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Monitor Clock Divider	4, 8, 16, 32, 64	4, 8, 16, 32, 64	4, 8, 16, 32, 128	4, 8, 16, 32, 128	4, 8, 16, 32, 128	4, 8, 16, 32, 128	4, 8, 16, 32, 128	8, 16, 32, 64
CDR CLOCK Divider	32, 64, 128, 256, 512, 1024, 2048, 4096	32, 64, 128, 256, 512, 1024, 2048, 4096	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
Supported RX EQ Types	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, DFE1_RC_DFE2, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ, DFE1_MPICAN_DFE2, DFE1_RC_MPICAN_DFE2	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, DFE1_RC_DFE2, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ, DFE1_MPICAN_DFE2, DFE1_RC_MPICAN_DFE2	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ	SLC1, SLC1_LDEQ, SLC1_RC_SLC2, SLC1_RC_LDEQ, DFE1, SLC1_MPICAN_SLC2, SLC1_MPICAN_LDEQ, SLC1_RC_MPICAN_SLC2, SLC1_RC_MPICAN_LDEQ
CTLE	Supported	Supported	Supported	Supported	Not Supported	Not Supported	Not Supported	Not Supported
Automatic RX FFE Taps	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
FEC	HW based Real FEC	HW based Real FEC	Emulator based FEC	Emulator based FEC	HW based Real FEC	HW based Real FEC	HW based Real FEC	HW based Real FEC
Noise Injection	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Supported	Not Supported	Supported
Embedded Module Host	Not Supported	Supported	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported

North America
48521 Warm Springs Blvd. Suite 310
Fremont, CA 94539
USA
+1 510 573 6388

Worldwide
Houmal Technology Park
Askarieh Main Road
Houmal, Lebanon
+961 81 794 455

Asia
14F-5/ Rm.5, 14F., No 295
Sec.2, Guangfu Rd. East Dist.,
Hsinchu City 300, Taiwan (R.O.C)
+886 3 5744 591