APPNOTE 003

A^{2}B CONTROL FROM PYTHON VIA SIGMA STUDIO

An example using ADI's COM interface to Sigma Studio, and how to work around some bugs

Rev 1c

25-Sep-20

Signal Processing

http://clk.works/

1. TABLE OF CONTENTS

1 INTRODUCTION

During development of an A^{2}B based audio system it can be convenient to be able to control the A^{2}B network from Python. This capability can be very helpful for test setups, as now various aspects of a complex test scenario can include control of A^{2}B without having to write custom code for direct A^{2}B access using the ADI A^{2}B library.

While ADI provides documentation for the API in the A^{2}B add-on, no examples are provided.

It also turns out that with the combination of the ADI's A^{2}B software release 19.3.1, Python 3.8, and Sigma Studio 4.5, the API does not work as expected. This of course might change in future releases of any of those components, but for now one needs to be aware of how to make the API work.

The basis for the app note is the question posted on the ADI Sigma Studio Engineer Zone forum:

https://ez.analog.com/dsp/sigmadsp/f/q-a/534341/sigmastudio-python-com-argument-list-pass-reference-to-return-a-byte-array

and credit for knowing the work around belongs totally with ADI engineer "JoshuaB".

There is a short video posted on the same page as this app note that shows the system in operation.

1.1 REFERENCES

There are a number of things you should read before tackling Python and A^{2}B.

1.1.1 DOCUMENTATION

- *A2B Scripting Guide*, Revision 2.0. AE_09_A2B_Scripting_Guide.pdf, section 3.4. The COM interface callas are all uppercase. This is part of the documentation in the 19.3.x software release.
- *SigmaStudio Scripting*, wiki page:
  - https://wiki-stage.analog.com/resources/tools-software/sigmastudio/usingsigmastudio/scripting

1.1.2 HARDWARE USED

The example uses a specific hardware setup.

- Analog Devices EVAL-AD2428WD1BZ A^{2}B root node eval board (also referred to as the WDZ board)
- Analog Devices EVAL-ADUSB2EBZ USBi interface (or equivalent)
- Analog Devices EVAL-ADT7420-PMDZ I^{2}C temperature sensor eval board
- Clockworks A^{2}B module and EVM board.

The supplied Sigma Studio example uses the above hardware, but other boards could be used with appropriate adjustment to the SigmaStudio schematic and/or example code.

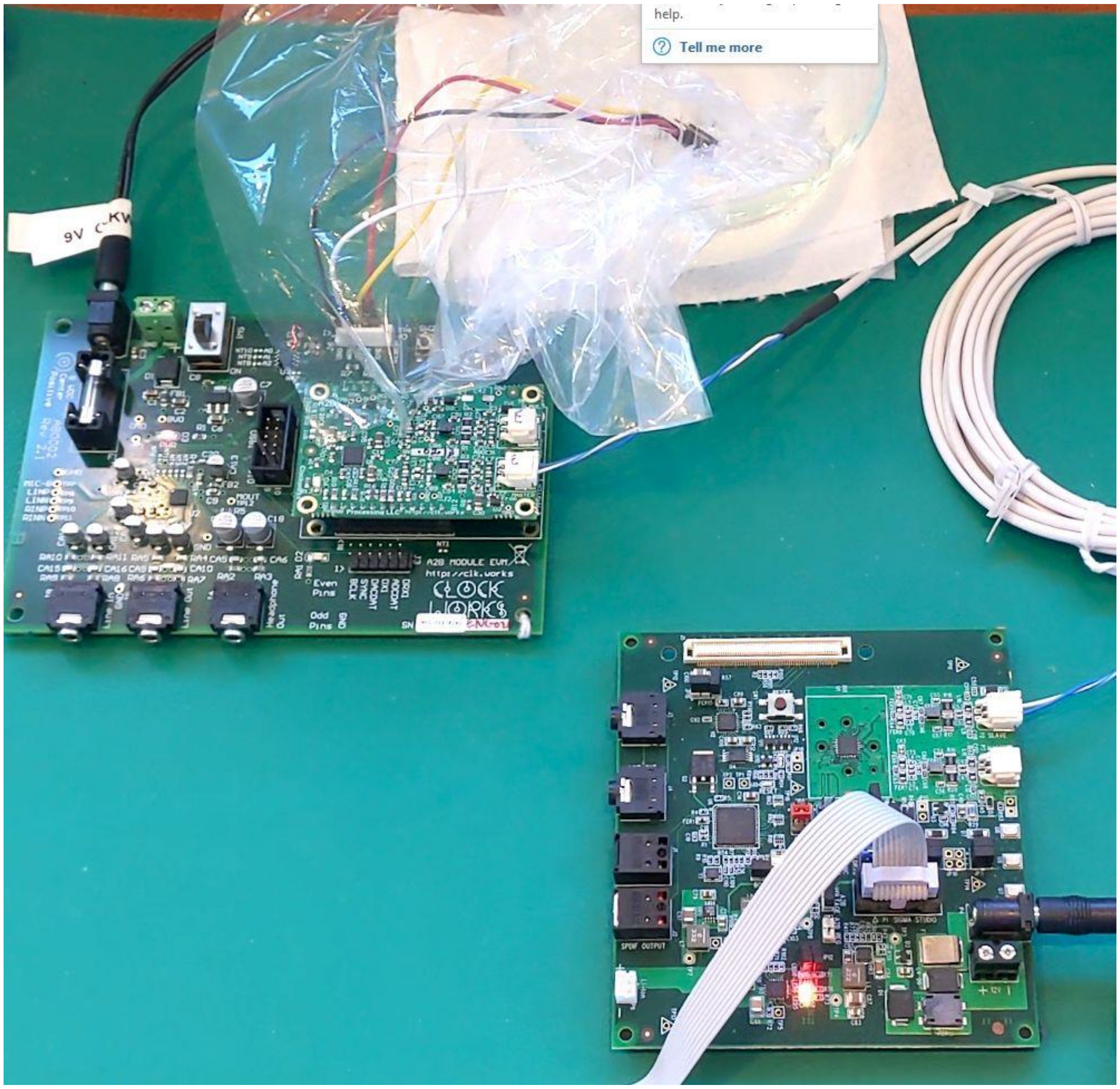

Figure 1 Hardware setup used for this example (see video for full details)

Figure 1 shows the hardware setup in more detail. The WDZ board (lower right) is controlled from a USBi (outside of the picture). The Clockworks module and EVM board (upper left) is connected via the EVM's control expansion connector (J6) with jumpers to ADI's ADT7420 temperature sensor EVM board. The pinouts for that are available from: https://wiki.analog.com/resources/eval/user-guides/eval-adicup360/hardware/adt7420 . It's default I^{2}C address is 0x48.

The ADI EVM board has 4 pins but they're wired in parallel on an 8 pin header:

1 ,2	SCL
3,4	SDA
5,6	GND
7,8	VDD (3.3V)

The pin assignments for J6 on the EVM board are as follows:

1	GND
2	SDA
3	GND
4	SCL
5	3.3V
6	RESETn

1.2 SETUP

The ADI documentation and wiki provide more details about setting up Sigma Studio for use with Windows COM. See this example: https://wiki-stage.analog.com/resources/tools-software/sigmastudio/usingsigmastudio/scripting/matlab

Most likely these three commands will be all that you need – at least until version numbers change again:

```
C:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319
regasm "C:\Program Files\Analog Devices\SigmaStudio
                          4.5\Analog.SigmaStudioServer.dll" /codebase
```

You will need to launch Sigma Studio prior to running the Python code.

2 CODE LISTING

The full code listing is provided at the end in Section 4.

The calling specifications defined the *A²B Scripting Guide* are implemented in Python by using VARIANTs. These two resources may be helpful for understanding more about what the code is doing:

http://timgolden.me.uk/pywin32-docs/html/com/win32com/HTML/variant.html https://www.oreilly.com/library/view/python-programming-on/1565926218/ch12s03s06.html

Most of the API calls that return a value return it as the last argument to the function. However from Python, due to whatever bug is at the core of this issue, the returned value ends up in the first argument.

The example is pretty simple, completing the following steps:

- Read the vendor register from the master node
- Turn on the LED on the EVM board (slave node 0, GPIO 4)
- In a loop, read the I2C temperature sensor once per second, and toggle the LED state

3 NEXT

There is one call in the API that returns two values:

A2B_GET_NETWORK_DISCOVERY_STATUS

```
<summary>Get A2B network discovery status</summary>
<param name="ICName">Name of A2B IC </param>
<param name="masterAddress"> A2B Master Transceiver I2C address (7-bit)
</param>
<param name="discStatus">Return status: True if the network if successful
else False</param>
<param name="numDiscSlaves">Return number of nodes successfully discovered in
the network</param>
public bool A2B_GET_NETWORK_DISCOVERY_STATUS(string ICName, int
masterAddress, out bool discStatus, out int numDiscSlaves)
```

Though we’ve tried the obvious things, so far the secret sauce for getting the two return values has not been determined.

Given the failure with two return values, we didn’t try this one with three:

A2B_GET_NETWORK_LINEFAULT_CODE

```
<summary>Get network line fault status returns 3 parameters: fault code,
fault node type and fault node position</summary>
<param name="ICName"> Name of A2B IC </param>
<param name="masterAddress"> A2B Master Transceiver I2C address (7-bit)
</param>
<param name="faultCode">Fault code from Interrupt Type Register</param>
<param name="faultNodeType">Fault node type : 0-Master, 1-Slave </param>
```

```
<param name="faultNodePosition">Fault Node Position : 0 for Master, 0 for 1st
Slave, 1 for 2nd Slave etc </param>
<returns></returns>
public bool A2B_GET_NETWORK_LINEFAULT_CODE(string ICName, int masterAddress,
out int faultCode, out int faultNodeType, out int faultNodePosition)
```

Other commands that we assume would also be broken are:

A2B_GET_BER_COUNT

4 THE CODE

```
# simple test of Python and A2B using the A2B Sigma Studio Extensions
#
# This version include the workaround for the return paramater passing bug
# see https://ez.analog.com/dsp/sigmadsp/f/q-a/534341/sigmastudio-python-com-argument-list-pass-reference-
to-return-a-byte-array
#
# for more about SigmaStudio scripting see:
#https://wiki.analog.com/resources/tools-software/sigmastudio/usingsigmastudio/scripting/python
#
# Watch out for this bug too:
# https://ez.analog.com/dsp/sigmadsp/f/q-a/534278/sigmastudio-com-interface-python-example-fails

import win32com.client
from win32com.client import VARIANT
import pythoncom
import time

if __name__ == "__main__":
    print('Running')
server = win32com.client.dynamic.Dispatch("Analog.SigmaStudioServer.SigmaStudioServer")
# Load a new SigmaStudio project
print('Loading Project...')
server.open_project(r'PATH_TO_YOUR_LOCATION\WDZ_EVM.dspproj')

# Compile and download the project to the DSP
server.compile_project


# example of the bug. The returned value is in the 1st parameter, not the last!
ic_name = VARIANT(pythoncom.VT_BYREF | pythoncom.VT_BSTR, "IC 1")
readdata = []
readdata_v = VARIANT( pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1, readdata)
stat = server.A2B_MASTER_REGISTER_READ(ic_name, 0x68, 0x02, 1, readdata_v)
print("A2B_MASTER_REGISTER_READ() returned {0}".format(stat))

print('Vendor ID is 0x{:02x}'.format(ic_name.value[0]))
```

```
# write to GPIO on remote node
# reset IC name (data will be returned in it due to COM bug)
ic_name = VARIANT(pythoncom.VT_BYREF | pythoncom.VT_BSTR, "IC 1")
masterAddress = 0x68
masterAddress_v = VARIANT(  pythoncom.VT_BYREF | pythoncom.VT_UI4, masterAddress)
# have to set these up even though the bug means nothing comes back in them
discStatus = 33
discStatus_v = VARIANT(  pythoncom.VT_BYREF | pythoncom.VT_BOOL, discStatus)
numDiscSlaves = 44
numDiscSlaves_v = VARIANT(  pythoncom.VT_BYREF | pythoncom.VT_UI4, numDiscSlaves)

writedata=[0x10]
writedata_v = VARIANT( pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1,writedata)
stat = server.A2B_SLAVE_REGISTER_WRITE('IC 1', 0x68, 0x69, 0, 0x4A, 1, writedata_v )
print("A2B_SLAVE_REGISTER_WRITE() returned {0}".format(stat))
time.sleep(1.)
writedata=[0x0]
writedata_v = VARIANT( pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1,writedata)
stat = server.A2B_SLAVE_REGISTER_WRITE('IC 1', 0x68, 0x69, 0, 0x4A, 1, writedata_v )

# now access an I2C peripheral, the ADT7420 temperature sensor (def address is 0x48), on slave node 0
# we'll read back register 0x0B (chip ID) to verify we can talk to it
ic_name = VARIANT(pythoncom.VT_BYREF | pythoncom.VT_BSTR, "IC 1")
masterAddress = 0x68
busAddress = 0x69
nodeID = 0
chipAddress = 0x48
readAddress = 0x0B
addrwidth = 1
readNumberBytes = 1
readdata = []
readdata_v = VARIANT( pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1, readdata)

stat =  server.A2B_SLAVE_PERIREGISTER_READ(ic_name,  masterAddress,  busAddress,  nodeID,
              chipAddress,  readAddress,  addrwidth,  readNumberBytes, readdata_v)
print('peripheral chip ID is 0x{:02x}'.format(ic_name.value[0]))


# read the temp, which is 2 bytes
readAddress = 0x00
addrwidth = 1
```

```
readNumberBytes = 2

LED_state = 0

for iLoop in range(100):
    # these get clobbered so have to set them up again
    ic_name = VARIANT(pythoncom.VT_BYREF | pythoncom.VT_BSTR, "IC 1")
    readdata = []
    readdata_v = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1, readdata)
    stat = server.A2B_SLAVE_PERIREGISTER_READ(ic_name, masterAddress, busAddress, nodeID,
                  chipAddress, readAddress,  addrwidth, readNumberBytes, readdata_v)
    # print('peripheral chip returned is 0x{0:02X}{1:02X}'.format(ic_name.value[0], ic_name.value[1]))
    temp_binary = ((int(ic_name.value[0]) << 8) | (ic_name.value[1] &0xF8)) >> 3
    temp_C = temp_binary * 0.0625
    temp_F = temp_C * (9/5) + 32
    print("binary {:04X}    deg C {:4.2f}       deg F {:4.2f}".format(temp_binary,temp_C, temp_F))
    writedata = [LED_state]
    writedata_v = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_BYREF | pythoncom.VT_UI1, writedata)
    stat = server.A2B_SLAVE_REGISTER_WRITE('IC 1', 0x68, 0x69, 0, 0x4A, 1, writedata_v)
    if LED_state == 0:
        LED_state = 0x10
    else:
        LED_state = 0
    time.sleep(1.)
```