How to interact with the firmware on WFD1 board

We interact with the Master FPGA using the IPbus protocol to issue commands within Python scripts we have written. Therefore, you need to have uHAL (the latest of many IPbus programs used by CMS) and Python packages already installed on your computer. If you already have this setup, skip the next section and continue on to learn how to use our Python scripts. If you do not any anything already installed, please read the next section.

If you DO NOT have uHAL and Python installed:

We'll start with how we installed uHAL on our Linux machine. We suggest making a new folder ipbus to contain all of the IPbus files. To check out the IPbus software from the CACTUS online repository, go to your ipbus folder and run:

```
> svn co http://svn.cern.ch/guest/cactus/tags/ipbus_sw/uhal_2_3_0 cactus
```

This will create a folder named cactus and put the IPbus software there. To build the software, go into the cactus folder and run:

```
> make Set=uhal
```

To install the required Python package, we believe that Nic originally used the conda command to find and install the needed software packages. We admit that we are not very clear on how this installation occurred, but we will describe here what we do know. Hopefully, the people from UKY will be more familiar with this part of the required installations and can help you out.

After you obtain the conda command, we believe that you create the required Python 3.4 environment by running:

```
> conda create -n ipbus python=3.4 anaconda
```

Here, "ipbus" is the name of the folder that will be created to contain the environment, and "anaconda" is "the meta-package that includes all of the actual Python packages comprising the Anaconda distribution" (cite: conda.pydata.org). We believe that this is all you will need to run, but we were not able to test it since we already have it properly installed. If this is all works, then you may continue on to the next section to learn how to interact with the Master FPGA firmware.

If you DO have uHAL and Python installed:

Before you can start to run our Python scripts, you will first have to edit your ".bashrc" file. In that file, you need to put in the following line:

```
export PATH=$PATH:<anaconda directory>/anaconda/bin
```

Here, <anaconda directory> should be replaced by the location of your installation of Python from your Anaconda distribution. If this does not make sense, you might want to look at the previous section on how to install the Python package needed. Note that you must restart your bash session in order for this change to take effect.

You will then need to add the IPbus libraries to your path in order to use the software. We will provide you with a script named "setenv.sh" which we use to do this when source by running:

```
> source setenv.sh
```

The contents of this script is copied below:

```
buildPath=<ipbus directory>/ipbus/cactus/cactuscore

export LD_LIBRARY_PATH=<anaconda directory>/anaconda/envs/ipbus/lib
export LD_LIBRARY_PATH=$buildPath/extern/boost/RPMBUILD/SOURCES/lib:
        $LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$buildPath/extern/pugixml/RPMBUILD/SOURCES/lib:
        $LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$buildPath/uhal/log/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$buildPath/uhal/grammars/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$buildPath/uhal/uhal/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$buildPath/uhal/tests/lib:$LD_LIBRARY_PATH
```

where <ipbus directory> should be replaced by the location of your uHAL installation which in the above case is under /ipbus/cactus/. Also in this script, <anaconda directory> should again be replaced by the location of your installation of Python from your Anaconda distribution. This script assumes that you installed Python under the name ipbus. If it is different, you will need to change /anaconda/envs/ipbus/lib to point to the correct directory.

Next, you need to source one more thing by running:

```
> source activate ipbus
```

This will prepend <anaconda directory>/anaconda/envs/ipbus/bin to your PATH environment variable. Note that ipbus in this command again is the assumed name under which your Python installation was created. You will need to change this to reflect any different naming convention used by the original installer. This command will also modify your PS1 environment variable so that your terminal prompt is prepended with the name of the environment. In the above case, the terminal prompt would appear as (ipbus) >.

Python scripts and related files required:

We will send you a gzipped tarball file named “SLAC_WFD1_files.tgz”. To extract the archive, you should run:

```
> tar -xzf SLAC_WFD1_files.tgz
```

In the extracted archive, you will find the following 8 files:

- connection.xml
- address_table.xml
- AMC13_AddressTable_K7.xml
- aurora_table.xml
- channelCommands.py
- reset.py
- chan_serial_status.py
- read_fake_data.py

In this section, we will briefly describe these files and point out what may need to be changed by you. If you are every confused about these instructions or what the files do, please do not hesitate to let us know. We will likely be more efficient for us to clarify any issue that arises than for you to study the code to find the problem.

connection.xml — This file is very important. It contains the IP addresses for the 2 WFD1 boards (S/N 2 and 4) and for the T1 of the AMC13. The IP addresses has been set to 192.168.1.130 for the WFD1 board S/N 4 and to 192.168.1.131 for the WFD1 board S/N 2. The IP addresses of the WFD1 boards are hardcoded in our firmware and, therefore, should not be changed. However, you will likely have to change the IP address for the T1 of the AMC13 since we are not sure of what you guys decided on for it. As of now, the T1 AMC13 IP address is set to 192.168.1.189 which was in Wes' ELOG. It should be self-explanatory on where to change the IP address if needed.

address_table.xml — This file is used in the connection.xml file, and it is a standard IPbus file which should not need to be altered.

AMC13_AddressTable_K7 — This file is used in the connection.xml file, and it is a standard IPbus file which should not need to be altered.

aurora_table.xml — This file is used in the address_table.xml file, and it is a standard IPbus file which should not need to be altered.

channelCommands.py — This Python script defines the RD_REG and WR_REG functions which are heavily used in some of the other scripts. You should not need to alter anything.

reset.py — This Python script sends a reset signal to the AMC13 and to both of the WFD1 boards. Note that you will need to comment out part of the code if you are trying to use only one WFD1 board. It should be clear from the documentation in the code what to comment out. Also note that the DAQ link will not be established unless you run this script. If you are having trouble with the DAQ link, we suggest trying to reset both the AMC13 and the WFD1 boards again. It is important to mention that you may receive a timeout error the first time you run this script — that is okay. You should simply run the script again until the error message goes away.

If you consistently receive a timeout error, then the connection to the devices has not been made, and you should check that the IP addresses in connection.xml are set correctly.

chan_serial_status.py — This Python script is not required for your intentions. it will output the status of the aurora link between the Master and the Channel FPGA. If everything is working, you should see channel_up = 1 in the output. We have included this script since it can be useful for diagnostic purposes when things aren't working properly.

read_fake_data.py — This script is where it all happens. Since the ADC is not currently doing anything, you will have to write the proper information to the ADC header FIFO and the fake data to the ADC memory addresses using this script. We have heavily documented the code so that the function and syntax of each part is clear. Right now, this script is setup for reading fake data from just the WFD1 board S/N 2. You will need to duplicate this file for the WFD1 board S/N 4 and simply change wfd1_sn2 to be wfd1_sn4 in Line 45. Do not, however, send another trigger to the AMC13 if you are going to run this script for both WFD1 boards. The AMC13 trigger only should be sent in the first script you run, and you will have to comment out that section of code in the second script you run.

This script currently puts the following 32-bit fake data words into the ADC memory: (in order)

- ADC memory address 0x00000000: aaaaaaaa
- ADC memory address 0x00000001: bbbbbbbb
- ADC memory address 0x00000002: cccccccc
- ADC memory address 0x00000003: dddddddd
- ADC memory address 0x00000004: eeeeeeee
- ADC memory address 0x00000005: ffffffff

Note the our Master FPGA firmware assumes that you put in an EVEN number of data words. The firmware will be stuck in an idle state if you attempt to put in an odd number instead.

This script also puts the following 32-bit header words into the ADC header FIFO: (in order)

- Trigger number: 0x00000001
- Buffer size: 0x00000006
- Channel number: 0x00000000
- Post-trigger count: 0x00000006
- Memory address of first buffer word: 0x00000000

Note that the buffer size is the number of fake data words you put into the ADC memory (counting from one). Hence, you must change the value of the buffer size every time you alter the number of fake data words sent. As long as you keep the ADC memory address of your first fake data word as 0x00000000, you will not have to change the value of the memory address of the first buffer word. The trigger number, channel number, and post-trigger count do not need to be changed at all.

To run a Python script, you need to run:

```
> python script.py
```

where script.py should be replaced by the name of the script you wish to run.

To review briefly, the typical order of operations (using both WFD1 boards) should include you running the following Python scripts: (in order)

- reset.py
- read_fake_data_wfd1_sn2.py (the WFD1 board S/N 2 version of read_fake_data.py)
- read_fake_data_wfd1_sn4.py (the WFD1 board S/N 4 version of read_fake_data.py)

After you run each version of read_fake_data.py, you will see some output messages in your terminal window. You should see only 0x3 as the response codes if everything was successful. Please refer to the documentation in the read_fake_data.py code for details if you see anything that is different than 0x3. This likely means that an error occurred somewhere in the firmware while executing the script.