Automated Test for HVB Assemblies

Software: Bronson Edralin
Hardware: James Bynes
Mentor: Gerard Visser

Date: 12/02/14

Abstract: The following is the documentation for the automated test of the High Voltage Board Assemblies, which will be part of the Belle II Detector subsystem for the KEK Particle Accelerator in Japan.

1 Introduction

The following is the documentation for the automated test of the High Voltage Board Assemblies, which will be part of the Belle II Detector subsystem for the KEK Particle Accelerator in Japan.

The first sections are about installation of packages, how to allow remote connection to PostgreSQL database, Datetime, saving files onto remote server using NFS mount and Software.

Everything should be installed on the RPi so please skip all the way to the Software section to read the high level description of the software and then read the Simple Instructions section.

2 Installation of packages

2.1 PostgreSQL

You will need to install PostgreSQL on the client and on the server in order for you to use the Application Programming Interface (API) called Psycopg2.

Download and install from this link:

- http://www.enterprisedb.com/products-services-training/pgdownload#osx

2.2 Psycopg2

Install from a package

- Linux
  - sudo apt-get install python-psycopg2
- Mac OS X
  - fink install psycopg2-py27
  - sudo port install py27-psycopg2
- Microsoft Windows
  - Jason Erickson maintains a packaged Windows port of Psycopg2 with installation executable. Download. Double click and you're done.
  - http://www.stickpeople.com/projects/python/win-psycopg/

Using a Python package manager

- pip install psycopg2

If you get error due to Anaconda

- Try searching for the package with Binstar:
  - C:\Anaconda>binstar search –t conda psycopg2
- May need to install Binstar command line client with
  - conda install binstar
- Command line search doesn't show "Platforms", so you will likely want to use web search:
  - https://binstar.org/search?q=psycopg2

- Follow link for package name that matches your platform. Click on link and next page will show the conda install command for that Binstar package. For example:
    - conda install -c https://conda.binstar.org/alefnula psycopg2

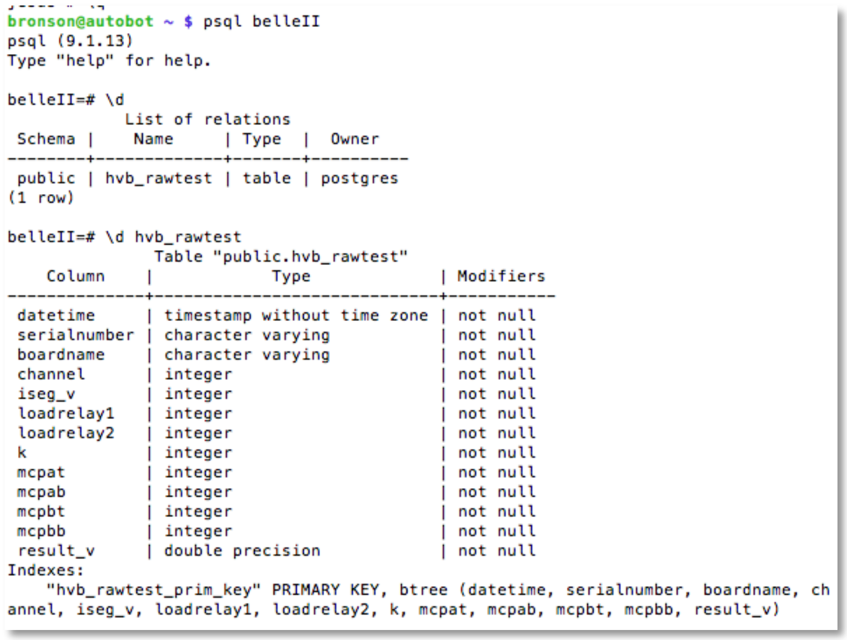
3 PostgreSQL: Popular Command-Line Cmds

Note: < > means you have to insert something there. Do not include brackets < >.

On Unix Command-Line:

1. creatdb <database_name>
2. createuser –P –s –e <username>
    a. This will create a superuser with the assigned username
    b. Please see http://www.postgresql.org/docs/9.1/static/app-createuser.html
3. dropuser <username>
    a. This will remove an existing PostgreSQL user. Only superusers and users with CREATEROLE privilege can remove PostgreSQL users.
4. psql <database_name>
    a. You will enter into postgreSQL interactive terminal called psql for that particular database called <database_name>.

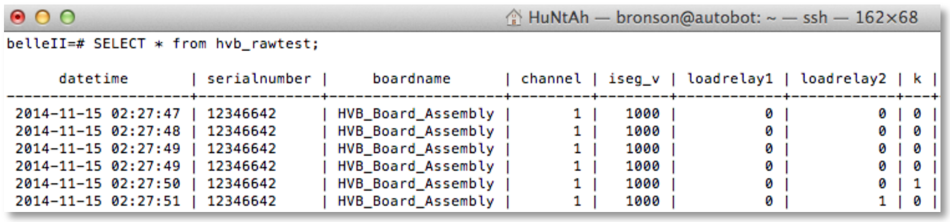
On PSQL (PostgreSQL Interactive Terminal), do not forget semicolons:

1. \l
    a. Lists all available databases, then exit. Similar to \list
2. \d
    a. Lists all relational tables in database
    b. 

```
bronson@autobot ~ $ psql belleII
psql (9.1.13)
Type "help" for help.

belleII=# \d
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | hvb_rawtest  | table | postgres
(1 row)

belleII=# \d hvb_rawtest
               Table "public.hvb_rawtest"
    Column    |            Type             | Modifiers
--------------+-----------------------------+-----------
 datetime     | timestamp without time zone | not null
 serialnumber | character varying           | not null
 boardname    | character varying           | not null
 channel      | integer                     | not null
 iseg_v       | integer                     | not null
 loadrelay1   | integer                     | not null
 loadrelay2   | integer                     | not null
 k            | integer                     | not null
 mcpat        | integer                     | not null
 mcpab        | integer                     | not null
 mcpbt        | integer                     | not null
 mcpbb        | integer                     | not null
 result_v     | double precision            | not null
Indexes:
    "hvb_rawtest_prim_key" PRIMARY KEY, btree (datetime, serialnumber, boardname, ch
annel, iseg_v, loadrelay1, loadrelay2, k, mcpat, mcpab, mcpbt, mcpbb, result_v)
```

3. ALTER USER <username> WITH PASSWORD '<pass>';
    a. Ex. ALTER USER joe WITH PASSWORD 'pass'
4. DROP USER <username>;
    a. Ex. DROP USER joe;
5. SELECT * FROM <table_name>
    a. This will perform a search query on the table.
    b. Ex. SELECT * FROM hvb_rawtest
    c. 

```
belleII=# SELECT * from hvb_rawtest;

      datetime       | serialnumber |     boardname      | channel | iseg_v | loadrelay1 | loadrelay2 | k |
---------------------+--------------+--------------------+---------+--------+------------+------------+---+
 2014-11-15 02:27:47 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          0 | 0 |
 2014-11-15 02:27:48 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          0 | 0 |
 2014-11-15 02:27:49 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          0 | 0 |
 2014-11-15 02:27:49 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          0 | 0 |
 2014-11-15 02:27:50 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          0 | 1 |
 2014-11-15 02:27:51 | 12346642     | HVB_Board_Assembly |       1 |   1000 |          0 |          1 | 0 |
```

4 Allow Remote Connection to PostgreSQL Database

In order to get this working, you will have to create a ssh tunnel by using port forwarding. You will also need to edit the postgress's config files (postgresql.conf and pg_hba.conf) to allow it to listen or allow you to connect to it remotely. The following steps will teach you this!

4.1 Port Forwarding

A solid tutorial may be seen at:

- http://www.noah.org/wiki/SSH_tunnel
- https://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/

4.2 How to allow Remote Connection to PostgreSQL Database using psql

When you install PostgreSQL, by default connection to the database using TCP/IP is not allowed.

When you try to connect from a client to a remote PostgreSQL database using psql command, you might get "psql: could not connect to server: Connection refused" error message.

In the following example, from a client machine, we are trying to connect to a PostgreSQL database that is running on 192.168.102.1 server. As you see from the output, it clearly says that the remote PostgreSQL database is not accepting connection

```
psql -U postgres -h 192.168.102.1
  psql: could not connect to server: Connection refused
    Is the server running on host "192.168.102.1" and accepting
    TCP/IP connections on port 5432?
```

To enable TCP/IP connection for PostgreSQL database, you need to follow the two steps mentioned below.

4.2.1 Modify pg_hba.conf to add Client Authentication Record

You can find the file by using:

- find / -type f -name "*.conf"

On my RPi it was found at /etc/postgresql/9.1/main/pg_hba.conf but it may be found in /var/lib/pgsql/data/pg_hba.conf

On the PostgreSQL database server, by default, you'll notice the following records towards the end of the /var/lib/pgsql/data/pg_hba.conf. As indicated below, it accepts connections only from the localhost.

```
# IPv4 local connections: host   all      all      127.0.0.1/32      trust
# IPv6 local     connections: host    all      all      ::1/128           ident
```

Add the following line to the pg_hba.conf server. This will allow connection from "192.168.101.20" ip-address (This is the client in our example). If you want to allow connection from multiple client machines on a specific network, specify the network address here in the CIDR-address format.

```
# vi  /var/lib/pgsql/data/pg_hba.conf
host    all       all       192.168.101.20/24    trust
```

The following are various client authentication record format supported in the pg_hba.conf file. We are using the #2 format from this list.

- local database user authentication-method [authentication-option]
- host database user CIDR-address authentication-method [authentication-option]
- hostssl database user CIDR-address authentication-method [authentication-option]
- hostnossl database user CIDR-address authentication-method [authentication-option]

Instead of "CIDR-address" format, you can also specify the ip-address and the network mask in separate fields using the following record format.

- host database user IP-address IP-mask authentication-method [authentication-option]
- hostssl database user IP-address IP-mask authentication-method [authentication-option]
- hostnossl database user IP-address IP-mask authentication-method [authentication-option]

4.2.2 Change Listen Address in postgresql.conf

On the PostgreSQL database server, by default, the listen address will be localhost in the postgresql.conf file as shown below.

```
# grep listen /var/lib/pgsql/data/postgresql.conf
listen_addresses = 'localhost'
```

Modify this line and give *. If you have multiple interfaces on the server, you can also specify a specific interface to be listened.

```
# grep listen /var/lib/pgsql/data/postgresql.conf
listen_addresses = '*'
```

4.2.3 Test Remote Connection

Now, login to the client machine 192.168.101.20, and perform the psql remote connection to the PostgreSQL database server (192.168.102.1) as shown below. This time, it should work.

```
# psql -U postgres -h 192.168.102.1
Welcome to psql 8.1.11 (server 8.4.18), the    PostgreSQL interactive terminal.
 postgres=#
```

Also, if you don't want to specify the hostname in the command line parameter every time, you can setup the remote PostgreSQL database ip-address in PGHOST environment variable name as shown below.

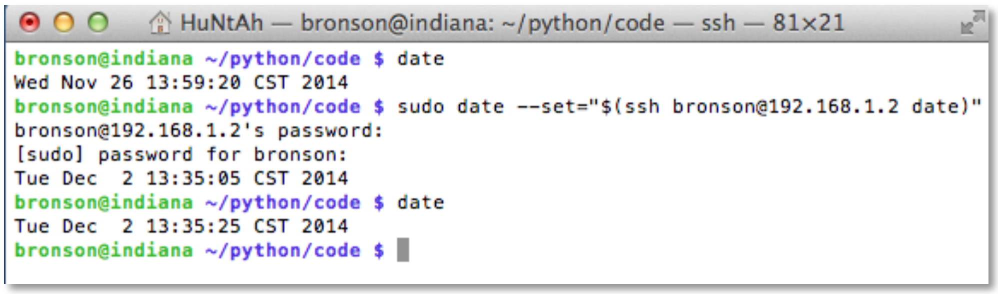
```
# export PGHOST=192.168.102.1
# psql -U postgres
Welcome to psql 8.1.11 (server 8.4.18), the PostgreSQL interactive terminal.
postgres=#
```

5 Datetime

5.1 Keep Datetime Current

The Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. NTP is intended to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC). You can view the settings in the /etc/ntp.conf config file. As long as the Raspberry Pi has Internet access, the Datetime should remain current.

In Indiana University's network, the Raspberry Pi (RPi) did not have Internet access so this became a problem for the timestamp. In efforts to keep the scripts scalable, we did not want to do something funky for the timestamp within the script. Instead, we will synchronize the RPi's Datetime with Gerard's server at daqtest1.iucf.indiana.edu (Private IP: 192.168.1.2). On command line, it would be a simple one-liner:

- sudo date --set="$(ssh username@hostIP date)"
- See SS for example:
    - 

```
bronson@indiana ~/python/code $ date
Wed Nov 26 13:59:20 CST 2014
bronson@indiana ~/python/code $ sudo date --set="$(ssh bronson@192.168.1.2 date)"
bronson@192.168.1.2's password:
[sudo] password for bronson:
Tue Dec  2 13:35:05 CST 2014
bronson@indiana ~/python/code $ date
Tue Dec  2 13:35:25 CST 2014
bronson@indiana ~/python/code $ 
```

To make this automated, I wrote a script called sync_datetime.py, which you will see in the Software section. It's a simple PYTHON script that will basically do what you would do in command line.

5.2 Coordinated Universal Time (UTC)

Each result from the automated tests will be time stamped. For consistency reasons, UTC will be used as the standard timestamp.

In order to change the localtime of the RPi to UTC time zone, you must do this in the command line, which basically creates a softlink to the appropriate time zone:

- ln –sf /usr/share/zoneinfo/UTC /etc/localtime

6 Saving Files onto Remote Server

6.1 Description

Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network much like local storage is accessed. NFS mounts work by sharing a directory between several virtual servers. This has the advantage of saving disk space, as the home directory is only kept on one virtual private server, and others can connect to it over the network. When setting up mounts, NFS is most effective for permanent fixtures that should always be successful. The Raspberry Pi (RPi) has very little space so the need for NFS mount was clear.

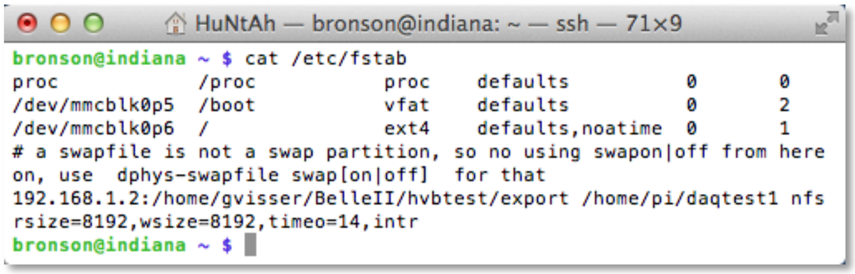
6.2 Setup

An NFS mount is set up between at least two virtual servers. The machine hosting the shared network is called the server while the ones connecting to it are called clients. In our case, the server is daqtest1.iucf.indiana.edu and the client is the Raspberry Pi that is on the same network.

SERVER: daqtest1.iucf.indiana.edu
- Private IP: 192.168.1.2

CLIENT: Indiana Raspberry Pi
- Private IP: 192.168.1.103

6.3 Setting it Up

You will need to have these settings to make NFS mount work
1. SSH into RPi (192.168.1.103)
2. In RPi's command line type:
    a. cat /etc/fstab newline > newfstab
    b. sudo mv /home/pi/newfstab /etc/fstab
    c. cat /etc/fstab
        i. 

```
bronson@indiana ~ $ cat /etc/fstab
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p5  /boot           vfat    defaults          0       2
/dev/mmcblk0p6  /               ext4    defaults,noatime  0       1
# a swapfile is not a swap partition, so no using swapon|off from here
on, use  dphys-swapfile swap[on|off]  for that
192.168.1.2:/home/gvisser/BelleII/hvbtest/export /home/pi/daqtest1 nfs
rsize=8192,wsize=8192,timeo=14,intr
bronson@indiana ~ $
```

    d. sudo mount /home/pi/daqtest1
        i. (RESULT:
        mount.nfs: rpc.statd is not running but is required for remote locking
        mount.nfs: Either use '-o nolock' to keep locks local, or start statd
        mount.nfs: an incorrect mount option was specified
    e. cd /home/pi/daqtest1
    f. ls –altr

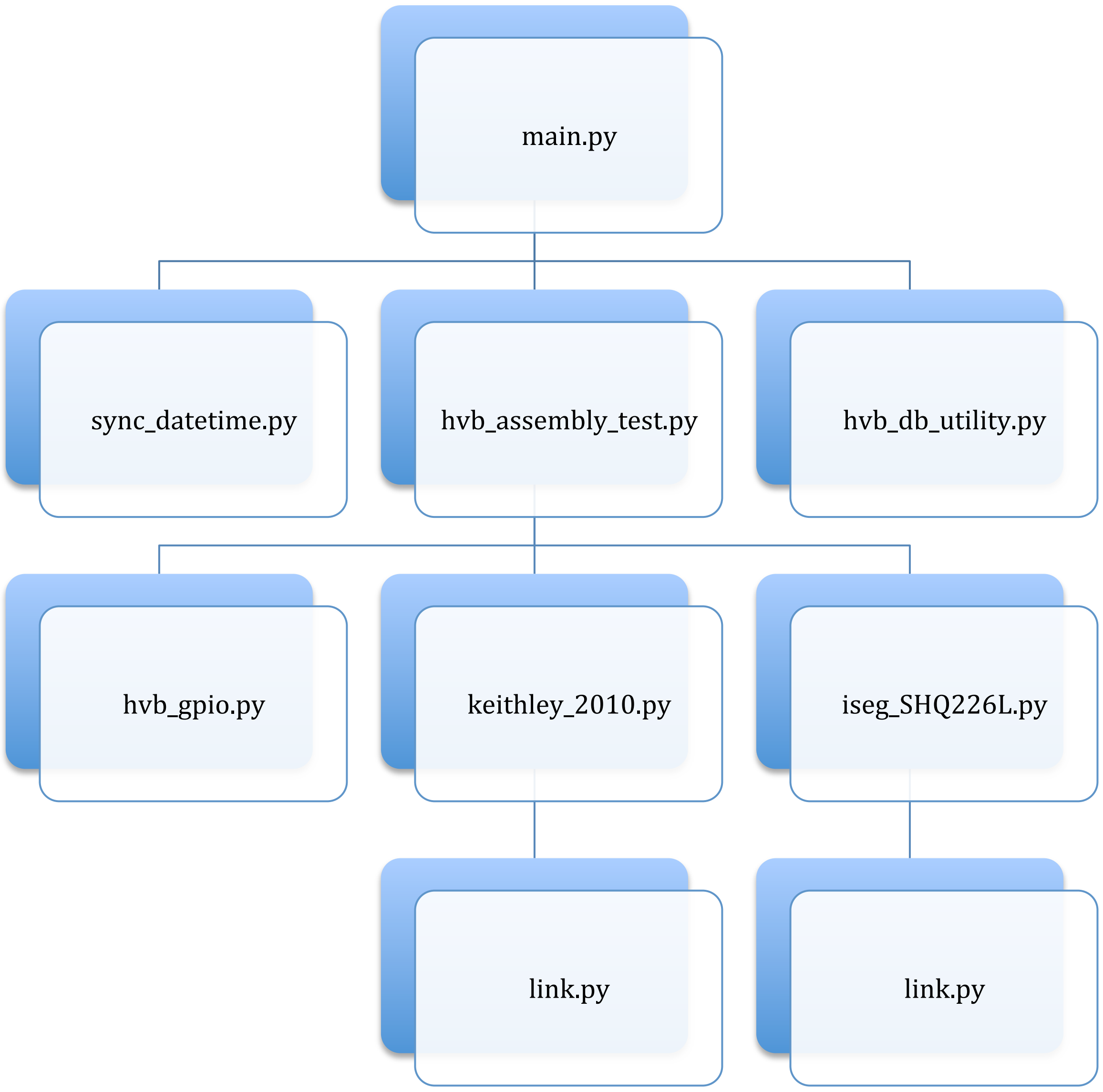
On /home/pi/daqtest1, the directory for is at *gvisser@daqtest1.iucf.indiana.edu :/home/gvisser/BelleII/hvb/export* and permissions are rwx for everyone. Export is defined in /etc/exports as /home/gvisser/BelleII/hvbtest/export 192.168.1.103(rw,sync).

6.4 Set auto-Mount for NFS on Reboot

On this particular Raspberry Pi (RPi), the Operating System (OS) used is called Raspbian. This flavor of linux is slightly different from what people are used to. In order to have the RPi automatically perform the mount on boot, you must edit the /etc/rc.local file and add this line to the end of the file:
- mount –o nolock /home/pi/daqtest1

7 Software

The following PYTHON scripts were written to automate the testing of the HVB Assemblies: main.py, sync_datetime.py, hvb_assembly_autotest.py, hvb_gpio.py, keithley_2010.py, iseg_SHQ226L.py, link.py and hvb_db_utility.py.



7.1 main.py

This script is responsible for putting everything together. In this file you can edit the important setting parameters for the automated test of the HVB Assemblies.

Default setting parameters include:

1) MULTIMETER_ADDR = “192.168.1.102”
2) HV_SUPPLY_ADDR = “RS232”
3) ISEG_VOLTAGE = “1000”
4) ISEG_RAMP_SPEED = “10“

5) TIME_PERIOD = “2”
6) PURPOSE = “HVB_RawTest”
7) LOC_DIR_FOR_STORING_CSV = “/home/pi/daqtest1/”

If you want to start the automated test, you must type this in your command line:

- sudo python main.py

A command prompt will pop up with description of the test and setting parameters. It will prompt you for three things:

1. “Is the current DateTime **DATETIME** correct? (yes/no): “
2. “Please enter serial number of board: ”
3. “Would you like to upload csv file **CSV_FILENAME** to database? (yes/no): “

7.2 sync_datetime.py

There is only one function we use in this script and it’s update_datetime(USERNAME,HOST). This function is used to synchronize the time on the RPi with another server’s time.

7.3 hvb_assembly_test.py

There is only one function we use in this script and it’s:

- hvb_assembly_test(multimeter_addr,hv_supply_addr,SERIAL_NUMBER,ISEG_VOLTAGE,ISEG_RAMP_SPEED,TIME_PERIOD,PURPOSE,LOC)

This script will basically take 20 measurements per channel. These 20 measurements consist of all the combinations for 2 load relays (00, 01, 10, 11) for each K, MCPAT, MCPAB, MCPBT, and MCPBB (00001, 00010, 00100, 01000, 10000). There are a total of 8 channels, which will give you 160 measurements ($(8\ chan) * (20\ meas) = 160\ meas$).

7.4 hvb_db_utility.py

This script is written to handle the uploading of data onto a postgreSQL database. A class is built inside this and it’s called DatabaseUtility. To create an object and use this class, you must do:

- db = hvb_db_utility.DatabaseUtility(host,port,dbname,user,password)

Two functions in the DatabaseUtility class are:

- create_table(tableName)
- insert_data_into_database(insertFilename, tableName)

One really good feature in this script is that it will create an error log file and I named it HVB_ASSEMBLY_DB_log. This feature is really important because if an error occurs, you can view this error and see exactly where the error occurred by looking at the log file. Then you can debug and fix it.

7.5 hvb_gpio.py

This handles the General Purpose Input Output (GPIO) pins. We are using certain pins as digital outputs to control the relays on the board. Pending on the outputs we can select each channel of the hvb assembly by selecting each board of the hvb test boards that were fabricated to test the hvb assemblies.

Six functions stand out in this script:

1. mux_relays(C, B, A)
    a. This is used when we want to select K, MCPAT, MCPAB, MCPBT and MCPBB (00001, 00010, 00100, 01000, 10000).
2. board_select(C, B, A)
    a. This is used when we want to select each channel of the hvb board assembly.
3. load_relays(B,A)
    a. This is used when we want to go through all the combinations of the load relays (00, 01, 10, 11).
4. invert_binary(state)
    a. This was needed for some of the GPIO pins, because we used an inverted decoder.
5. change_states(channel_addr, load_state, mux_state)
    a. This will ensure the right GPIO pins are turned on and off to control the relays on the hvb test board. We used discontinuity tests to double-check our work.
6. reset_all_gpio()
    a. This will turn all GPIO pins low, which are the default values.

7.6 keithley_2010.py and iseg_SHQ226L.py

These are basically libraries that hold all the commands to control the instruments. You can practically call these drivers for the instruments. The classes were built with the intent to make it easier when you write the automation test script. These libraries are highly scalable. Whenever you need to use one of these instruments, you can use these built libraries, which use the object-oriented concept of Getters and Setters.

Example of using it:

-
  ```
  from keithley_2010 import *
  keith=Keithley_2010("192.168.1.102",1234)
  id=keith.identification()
  print id
  keith.configure = "VOLTAGE"
  keith.configure_voltage = "DC"
  func,acdc = keith.configure_voltage
  ```

7.7 link.py

This script handles all the drivers for the RS232 to USB and Ethernet connection. This script is highly scalable whenever you need to create a link so you can communicate with instruments.

There are 3 classes used in this script:

1. Ethernet
   a. This uses a software package that you have to install called vxi11. This works really well for only certain instruments if it is supported from what it seems.
2. Ethernet_Controller
   a. This worked really well by using PYTHON's socket module. We had to use a PROLOGIX GPIB-ETHERNET CONTROLLER so we could assign an IP to the Keithley2010 Multimeter and have it run on the network. Then we can control the Keithley2010 Multimeter by using the Ethernet_Controller class in the link.py script.
3. RS232
   a. This will handle any RS232 to USB connection.

All classes have the same following functions:

1. cmd(cmd)
   a. You use this function when you want to set something by giving the instrument a command.
2. ask(cmd)
   a. You use this function when you require some feedback. Example, maybe you want to know what voltage is being read from the Keithley 2010 Multimeter.
3. ask_print(cmd)
   a. This is similar to ask(cmd) but the result will be displayed on screen. This may be good for debugging purposes.

8 Simple Instructions

On the Indiana University's network, there is a server called daqtest1 with public IP (daqtest1.iucf.indiana.edu) and private IP (192.168.1.2). Underneath that server is the Raspberry Pi (RPi) with no access to the Internet and its private IP is 192.168.1.103. In order to upload results from csv file to PostgreSQL database on IDLab's server with public IP (idlab.phys.hawaii.edu), you have to enable port forwarding. IDLab's server is based at University of Hawaii.

Do the following on command line before you run the main.py script. This basically creates a pipe to upload to PostgreSQL database on port 5432:

- ssh –L 5432:localhost:5432 bronson@192.168.1.2
  - You will need to type the password for the username
- ssh –L 5432:localhost:5432 postgres@idlab.phys.hawaii.edu

    - You will need to type the password for the username

Now, all you have to do is type:

- sudo python main.py

It will prompt you with straightforward questions. Just follow along and the results will be on their way!