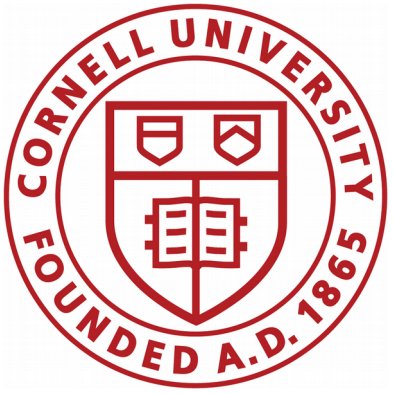


CBPM development

Antoine, Nate

CBPM meeting

January 15, 2020

Toward increasing beam current

Toward increasing the beam current

We know that:

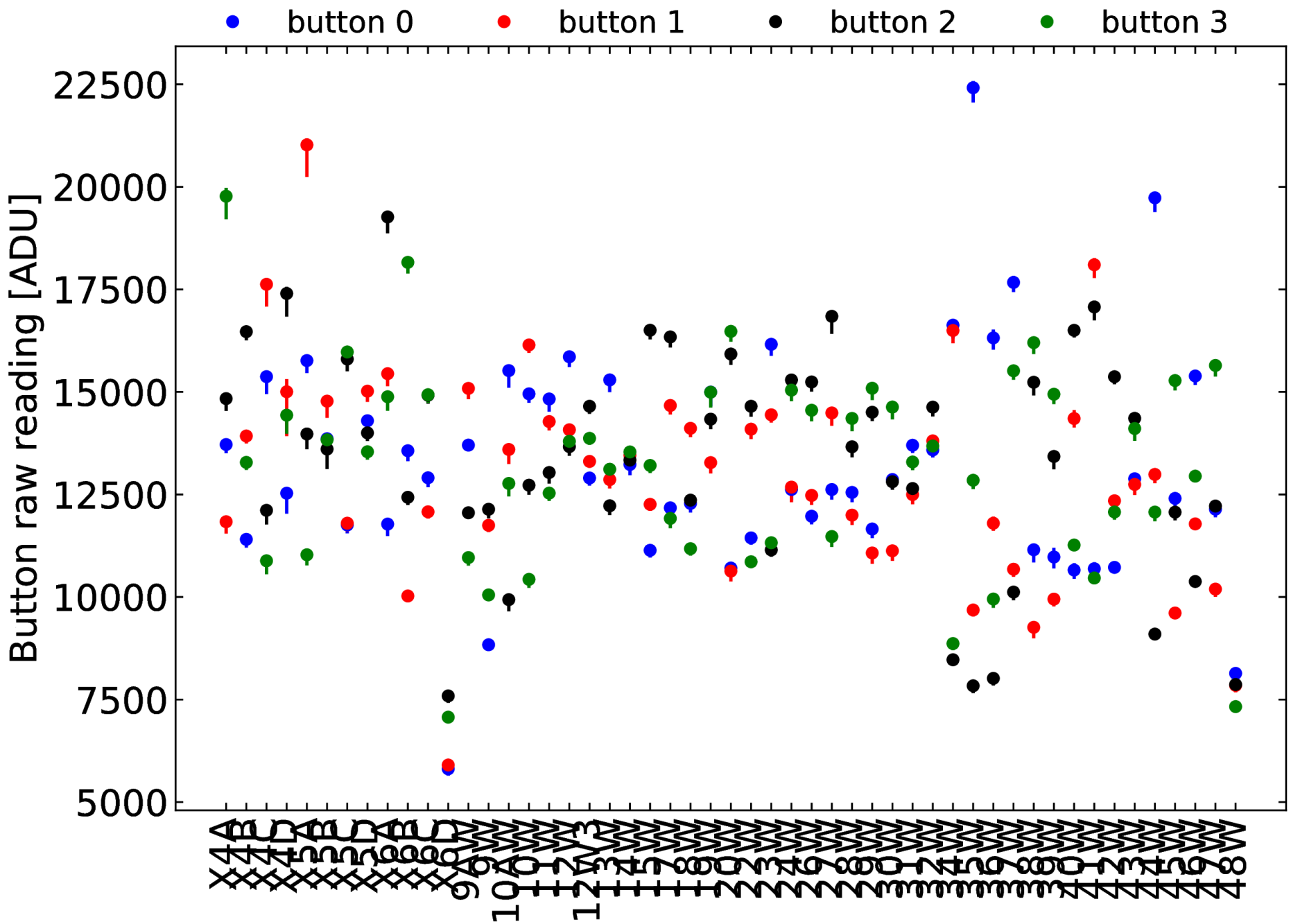
The signal-to-noise ratio of each button is an important source of resolution error. Maximizing the signal-to-noise ratio, i.e. the beam current, allows to improve the single-shot spatial resolution.

Machine study:

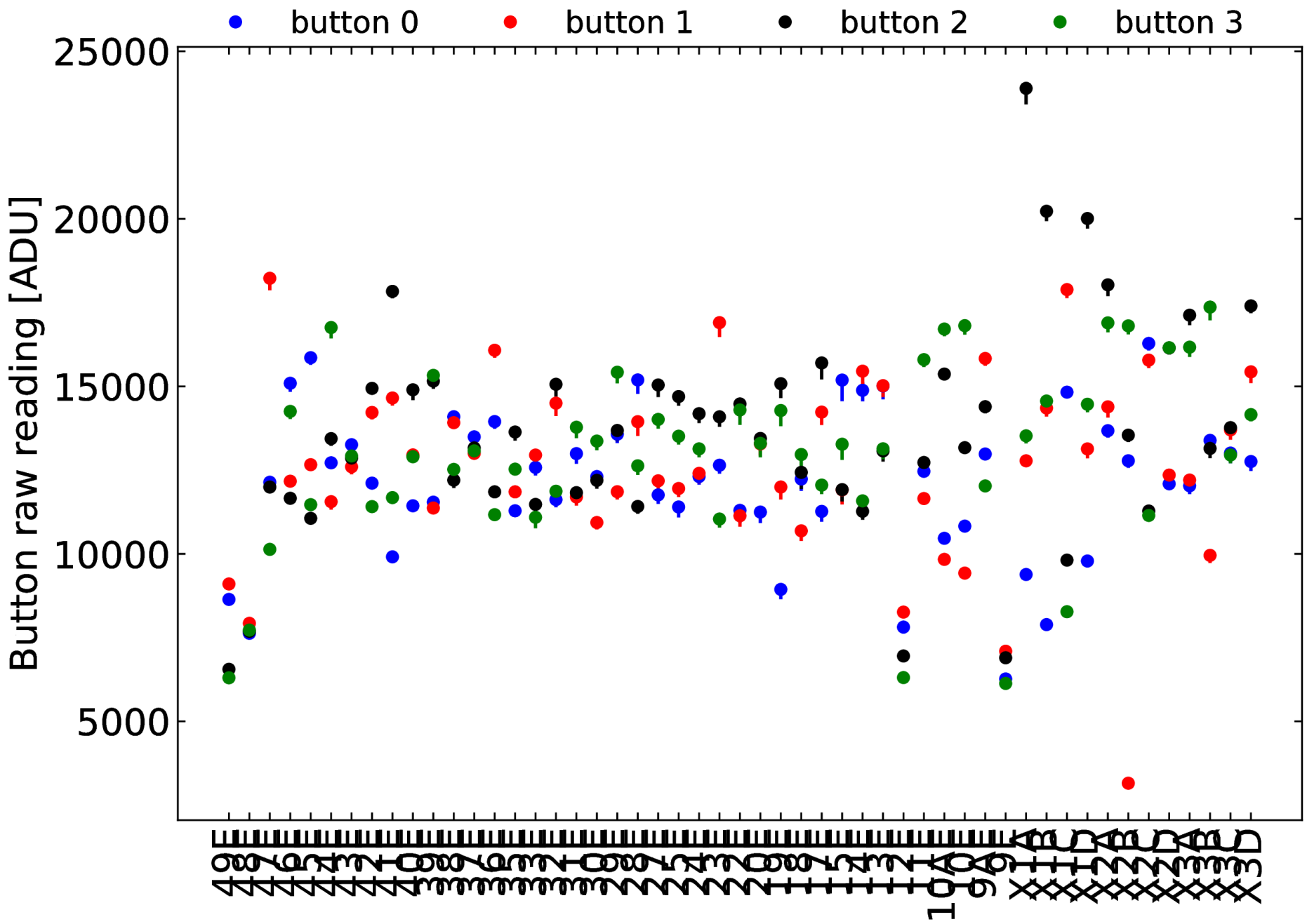
The default beam current of train 1, bunch 1 is 0.5 mA. During Machine Study time, we collected data for all the CBPMs used in orbit measurement for nominal beam conditions (see information in the elog 1840).

We are interested in maximizing the button readings close to 30,000 ADU all around the ring to minimize this source of error.

West server


button 0
button 1
button 2
button 3
Button raw reading [ADU]
22500
20000
17500
15000
12500
10000
7500
5000

East server


button 0
button 1
button 2
button 3
Button raw reading [ADU]
25000
20000
15000
10000
5000

Notes

- ✗ upper envelop is about 25,000 ADU
- ✗ lower envelop is about 10,000 ADU
- ✗ without attenuation, the maximum beam current increase is about 20% :
  - ➢ it would bring 25,000 ADU to 30,000 ADU
  - ➢ but only 10,000 ADU up to 12,000 ADU
  - ➢ it is all about relativity...
- ✗ the best improvement → move the lower envelop as high as possible
- ✗ how much room do we need for beam excursion (like tune tracker)?

One possible **first** course of action: attenuate all the buttons that reads above 20,000 ADU (about 5 buttons) to allow a 50% beam current increase. Also do an inspection of the buttons reading below 10,000 ADU (about 30 buttons).

Python3 and MPM_NET

Python3 and MPM_NET

Documentation

Python3 and CBI_NET

Documentation about using CBI_NET from Python3 is ready. I can put it anywhere useful: directories, wiki, ...

CESR BPM
Version 1.0
January 7, 2020

On using the CBI_NET client library with Python3

A. Chapelain[1]
[1]Cornell Laboratory for Accelerator-based Sciences and Education.

Contents

1 Introduction 1
2 Where to find the CBI_NET client library 1
3 How to compilte CBI_NET client 2
4 Python3 and CBI_NET example code 2

1 Introduction

CBI_NET client is a custom library written in C that allows communication between the CBPM electronics and a Linux environment. More information is available here [1]. CBI_NET is currently used by software written in C and in MATLAB. This note presents how to use CBI_NET with Python3.

2 Where to find the CBI_NET client library

The shared[1] CBI_NET library is part of the nightly build and is located in the directory containing the various libraries:

```
/nfs/cesr/online/lib/Linux_x86_64_intel/nightly/production/lib/libcbi_net.so
```

There also is a debug version of the library available here:

```
/nfs/cesr/online/lib/Linux_x86_64_intel/devel/debug/lib/libcbi_net.so
```

[1]A library that is automatically linked into a program when the program starts, and exists as a standalone file. Shared library on Linux system have a .so file extension

3 How to compilte CBI_NET client

The CBI_NET code (which includes more than just the client) can be retrieve from SVN via the command:

```
$ svn co https://accserv.lepp.cornell.edu/svn/Comm/Comm_libs/cbi_net
```

The CBI_NET client source code is located in the client directory. To compile the client from a Linux machine on the CLASSE network, one need first to enable the compilation of shared library by exporting the environment variable:

```
$ export ACC_ENABLE_SHARED=Y
```

and then run from the top level of the cbi_net directory the command:

```
$ mk
```

or alternatively for the debug version of the library:

```
$ mkd
```

The shared CBI_NET client library libcbi_net.so will be placed in a directory created during the compilation stage located with respect to the top level of the CBI_NET directory:

```
$ ../production/lib/
```

or alternatively for the debug version:

```
$ ../debug/lib/
```

4 Python3 and CBI_NET example code

The code shown in this section was tested with the following Python3 setup:

```
source /opt/rh/rh-python36/enable
export PYTHONPATH=/nfs/opt/python3.6/packages.sl7
```

There are different ways to call C libraries from Python3. This example relies on ctypes: "ctypes is a foreign function library for Python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python". Information about ctypes can be found here [2].

The following code snippet is a minimalistic example that uses ctypes to call functions of the CBI_NET client library. The code opens a socket to a CBPM module (the specific IP address corresponds to a bench-test module), read the version number of its FPGA firmware and close the socket.


```python
# import the ctypes functionalities
from ctypes import create_string_buffer, cdll

# create an instance of the cbi_net library
cbi_net_lib = cdll.LoadLibrary('/nfs/cesr/online/lib/Linux_x86_64_intel/
    nightly/production/lib/libcbi_net.so')

# open a socket to a CBPM module by using the 'cbi_net_fdopen' function
    provided by the library
socket = cbi_net_lib.cbi_net_fdopen(b'192.168.32.191')

# define variables required to access the FGPA firmware version number
address = 150994944 + 2
num_read = 8
data_type = 4

# create a string buffer that will store the FPGA version number
string_buffer = create_string_buffer(8)

# read the CBPM module memory to retrieve the FPGA version number using the '
    cbi_net_rd_mem' function provided by the library
num_words =  cbi_net_lib.cbi_net_rd_mem(socket, address, num_read, data_type,
     string_buffer)

# print out the FPGA version number
print('FPGA version: ', ord(string_buffer[0]))

# close the socket
cbi_net_lib.cbi_net_close_socket(socket)
```

The main difficulty in calling C function from Python3 is to match the variable type that C requires. ctypes provides the required functionality but it might not be straightforward to use it properly. One example is the fact that there is no pointer variable type in native Python3 but there is in C.. An other important difference has to do with variable types being mutable or immutable. More information about the differences between Python3 and C and how ctypes help can be found here [3].

References

[1] CBI_NET documentation: link
[2] ctypes documentation: link
[3] Pointers in Python: What's the Point?: link

Additional materials