How to run tests on the JIVE UniBoard Hardware

1. Setting Things Up

Programming the FPGAs

Open the quartus programmer and add the jtag server at 10.88.5.2, pswd xxxxx. Then it should be possible to connect to the USB cable.

Control Connections

The UniBoard control port is connected directly to the eth1 port in the control computer (uni-ctl) with IP address 10.99.0.254. The default settings should be used when compiling unb_os so that the FPGAs have the following control IP addresses:

fn0 10.99.0.1
fn1 10.99.0.2
fn2 10.99.0.3
fn3 10.99.0.4
bn0 10.99.0.5
bn1 10.99.0.6
bn2 10.99.0.7
bn3 10.99.0.8

Test the connection by logging on to uni-ctl and pinging one of the nodes
ssh

```
ssh hargreaves@10.88.2.5 (or ssh hargreaves@uni-ctl)
ping 10.99.0.1
```

If there is no reply to ping either there is a network connection or unb_os is not loaded. This can be checked by opening a jtag terminal on the machine used to program the FPGAs.

For example `nios2-terminal -d 1` should result in something like:

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster on 10.88.0.10 [2-1]", device 1, instance 1
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

unb_osx/build 2
unb_osx/aTest is-at 0x17fc0
NET_Setup/backplane 0 node 0 [FN]
NET_Setup/setup_tse: ip=10.99.0.1
```

If not try reloading the unb_os with:

```
unb_elf fn_top app=unb_osx 1
```

Erlang setup

First start an Erlang node on uni-ctl. This only has to be done once as it can be shared by Erlang nodes running on many users' machines

```
ssh hargreaves@uni-ctl /home/hargreaves/Software/bin/unb_shell -name jh@10.88.2.5
-setcookie 1234
```

then start a local Erlang shell in a different terminal window on your own machine. Give it a unique name but the same cookie

```
../bin/unb_shell -name jh -setcookie 1234
```

and start a 'Board'

```
Board = uniboard:start([{ipbase, "10.99.0.0"}, {all, {node, 'jh@10.88.2.5'}}, {all,
{personality, evn_pers}}, {fn, {system, "/export/jive/hargreaves/fn_top_system.h"}},
{bn, {system, "/export/jive/hargreaves/bn_top_system.h"}}]).
```

Now it should be possible to see all the nodes sharing the cookie 1234 by typing `nodes().`

Next initialize the 10gbe ports. They do not have a MAC or IP address until they are initialized by the control system. The eight calls to `evn_tengbe:init` listed below set the correct MAC and IP addresses for each of the eight FPGAs. The parameters of `evn_tengbe:init` are

node (FPGA)
port (0 1 2 or 3)
MAC address
IP addresses
Gateway (last octet only, assumes same subnet as local IP)
Subnet mask (last octet only thus FF.FF.FF.mask)

```
uniboard:map(Board, fun evn_tengbe:init/6, [{fn0, 0, 16#00228608AA01, 16#C02A786a, 16#69, 16#F8}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{fn1, 0, 16#00228608AA02, 16#C02A786b, 16#69, 16#F8}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{fn2, 0, 16#00228608AA03, 16#C02A786c, 16#69, 16#F8}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{fn3, 0, 16#00228608AA04, 16#C02A786d, 16#69, 16#F8}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{bn0, 0, 16#00228608AA05, 16#0A58020A, 16#FA, 16#00}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{bn1, 0, 16#00228608AA06, 16#0A58020B, 16#FA, 16#00}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{bn2, 0, 16#00228608AA07, 16#0A58020C, 16#FA, 16#00}]).
uniboard:map(Board, fun evn_tengbe:init/6, [{bn3, 0, 16#00228608AA08, 16#0A58020D, 16#FA, 16#00}]).
```

Before the FN ports can be used, the PHY chip EPCS mode must be switched off:

```
uniboard:map(Board, fun evn_tengbe:epcsoff/2, [{fn, 0}]).
```

This is not necessary for the Bns as they do not have a PHY chip.

Next some ARP requests can be sent. This simply cycles through all the IP addresses in the subnet and stores the MAC address from any which reply. The ARP table persists until the chip is reconfigured.

```
uniboard:map(Board, fun evn_tengbe:arpon/2, [{fn, 0}]).
```

stop ARPing after a few seconds

```
uniboard:map(Board, fun evn_tengbe:arpoff/2, [{fn, 0}]).
```

Do this for the back nodes as well if the 10GbE output is to be used.
Other useful commands for reading the status and data from the correlator design are listed in file ~/erlangcode/corrtest1.erl

Testing the FN 10GbE ports

The IP address allocation for the front nodes is
192.42.120.104 Network address (do not use)
192.42.120.105 gateway (this is the JIVE switch/router)
192.42.120.106 fn0
192.42.120.107 fn1
192.42.120.108 fn2
192.42.120.109 fn3
192.42.120.110 e-uni-ctl (new 1G interface)
192.42.120.111 broadcast address (do not use)

192.42.120.110 is eth2 on the uni-ctl machine. It can be used to monitor activity on the FN ports eg

```
ssh hargreaves@uni-ctl
sudo tcpdump -i eth2 -X
```

This should see the ARP requests and replies on the UniBoard subnet. Ping also works from anywhere in the building.

```
ping 192.42.120.106
```

Testing the BN 10GbE ports

Note that the BN design currently uses Port 2 so the CX4 cables should be connected to the relevant connector at the back of the UniBoard. As there is only one port instantiated in the design it is still addressed as Port 0 in the Erlang commands.

The BN IP addresses are

uni-bn0 10.88.2.10
uni-bn1 10.88.2.11
uni-bn2 10.88.2.12
uni-bn3 10.88.2.13

Again, ping should work from uni-ctl and jop72 (or any other machine). It is necessary to do some ARPs before sending data out of these ports:

```
uniboard:map(Board, fun evn_tengbe:arpon/2, [{bn, 0}]).
uniboard:map(Board, fun evn_tengbe:arpoff/2, [{bn, 0}]).
```

Setting up the Capturer

The capturer is not part of the UniBoard svn To install it type the following in a Linux shell:

```
mkdir ccs
cd ccs
svn checkout
svn+ssh://hargreaves@jop72.nfra.nl/export/home/jive_cvs/UNIBOARD/data_capture
cd data_capture/
make
```

Start a new Erlang node (unique name, same cookie) and create a Board

```
~/UniBoard_FP7/UniBoard/trunk/Software/bin/unb_shell -pa unb_capture/ebin -name
jhcapturer -setcookie 1234

Board = uniboard:start([{ipbase, "10.99.0.0"}, {all, {node, 'jh@10.88.2.5'}}, {all,
{personality, evn_pers}}, {fn, {system, "/export/jive/hargreaves/fn_top_system.h"}},
{bn, {system, "/export/jive/hargreaves/bn_top_10gbe_system.h"}}]).
```

Then capturing can be started and stopped using the following Erlang command sequence

```
{ok,Pid} = unb_capture:start(Board, bn0, "10.88.2.5", 'jh@10.88.2.5').
File  = unb_capture:capture(Pid,"/tmp/output.10").
unb_capture:uncapture(Pid, File).
f(File).
unb_capture:shutdown(Pid).
f(Pid).
```

When the capturer is started it sets the destination UDP port to 30000 and the destination IP address to the third parameter in the `unb_capture:start` command.

2. Processing Data

Change the integration period

By default `evn_fn:start` sends 16 FFT periods of data to the back nodes, and `evn_bn:runint` processes 16 FFT periods of data stored in the corner turner. The run length, or integration period can be changed (for example to 32 FFT periods) by setting

```
uniboard:map(Board, fun evn_fn:set_tint/2, [{fn0, 32}]).
```

in the front node, and

```
uniboard:map(Board, fun evn_bn:setnumffts/2, [{bn, 32}]).
```

in the back node.

Set the Normalization

Selects a 9 bit slice of the 18 bit data output from the filter bank. For sine wave tests select bits 13:6 using:

```
uniboard:map(Board, fun evn_fn:set_norm/2, [{fn0, 6}]).
```

Set the Station Code

Set the two character ASCII code for each station in use. Eg to set station 1 on fn0 to accept data from Jb (0x4a62) use

```
uniboard:map(Board, fun evn_fn:set_stationcode/3, [{fn0, 1, 16#4a62}]).
```

Transmit data to the FN

First send data to the Fns. It is best to do this from uni-ctl otherwise jumbo frames may get truncated.

```
cd ~/cprogs
./vdif2udp 192.42.120.106 1000 sine_2cycle.vdif
```

See if it has arrived at fn0 with

```
uniboard:map(Board, fun evn_fn:stat/1, [fn0]).
```

Then send it on to the back nodes with

```
uniboard:map(Board, fun evn_fn:start/1, [fn0]).
```

See if it has arrived at all the back nodes with

```
uniboard:map(Board, fun evn_bn:stat/1, [bn]).
```

There are two versions of the BN firmware, one which allows the products to be read back one at a time via the Nios and 1GbE control port, and one which streams the products automatically to 1 10GbE port. The 10GbE version will become the production version.

Reading back products via the Nios (Test)

When the BN is set up to output through the Nios do

```
uniboard:map(Board, fun evn_bn:runint/3, [{bn0, 124, 1}]).
```

The parameters of `evn_bn:runint` are

node (FPGA)
the MAC cell number (0 to 131)
the product within that MAC cell (1 to 16)

The data for all 1024 frequency bins will be listed to the screen and also saved in a file called products. The columns are frequency bin, validity count, real product, imaginary product. Only one

product can be read out at a time but the process can be repeated as often as necessary. Note that the processing alternates between two sets of data held in the two corner turner memories. So the command must be performed twice to get the original data back.

Reading back products via the 10GbE (Production)

First make sure a capturer is running. Then select the products to output, for example the following command selects 4 products 0x07bf to 0x07c2.

```
uniboard:map(Board, fun evn_bn:enable_product/2,[{bn0, [16#07c0, 16#0600, 16#07d0, 16#0100]}]).
```

Note that the network connected to the back nodes is currently limited to non-jumbo frames. Thus if more than 74 products are requested the packets will disappear.

Process and transmit the selected products with

```
uniboard:map(Board, fun evn_bn:runint/3, [{bn0, 124, 2}]).
```

This is the same command as used when reading the products via the Nios, except that the MAC cell and product address parameters are ignored and the data printed to the screen is meaningless. The data are stored in the file set by the capturer (tmp/output.10 in the example above) and can be looked at using Matlab.