NumaChip Performance Counters

User Guide

Revision **1.0**, September 03, 2012

Revision History

Rev	Date	Author	Changes
1.0	2012-08-28	Atle Vesterkjær	First version

1 INTRODUCTION

1.1 Abstract

This is a user guide for programming and monitoring the NumaChip performance registers. The registers are availble using CSR accesses towards the NumaChip. The performance counters can be programmed to count 64 different events, using 8 counters at a time. The performance counters can either be programmed and monitored using a commandline tool called *nc_perf* or through a GUI client. *nc_perf* has to run on the master node in the NumaConnect Single Image System. The GUI client, *nc_pstat_gui* fetches data from a server daemon operating on the master node of the NumaConnect Single Image System. The server daemon is called *nc_stat_d*.

1.2 Document overview

In chapter 2 the NumaChip performance registers are described.
In chapter 3 the NumaChip user space library is described.
In chapter 4 the *nc_perf* utility is described.
In chapter 5 the NumaChip performance statistics GUI client is described, while the serverdaemon *nc_stat_d* (operating on the master node of the NumaConnect Single Image System) needed to operate the GUI is described in chapter 6.

1.3 References

Ref#	Document Name	Storage Location on Server	Revision	Date
1	DNC_CSR.pdf	Internal	NA	2012.03.07

1.4 Abbreviations

Throughout this document the following terms might be used:

API	Application Programming Interface
CSR	Control and Status Registers
Numascale AS	The name of the company that develops and owns NumaChip and NumaConnect.
NumaConnect	The technology from Numascale that enable Single Image Systems with cache coherence. See http://www.numascale.com/numa_technology.html
NumaChip	The actual asic that enables the NumaConnect Single Image Systems, see http://www.numascale.com/numa_products.html
HT	Hyper Transport
GUI	Graphical User Interface
nc_perf	NumaChip Performance Statistics commandline tool.
nc_stat_d	Single Image System master node daemon
nc_pstat_gui	The NumaChip Performance Counter Statistics GUI a Single Image System gui client.
Qt	Cross-platform application and UI platform, http://qt.nokia.com/
Qwt	Qt Widgets for Technical Applications, http://qwt.sourceforge.net/

2 NUMACHIP PERFORMANCE REGISTERS

The NumaChip performance registers are reachable through global csr accesses. There are 8 programmable performance counters. They are documented in the DNC_CSR.pdf [1].

The picture below illustrates how to select a source for your performance counter. There are eight different sources to choose from:

Select = 0, REM/SPrb
Select = 1, REM/HReq
Select = 2, LOC/SReq
Select = 3, LOC/HPrb
Select = 4, CData
Select = 5, FTag
Select = 6, MCTag
Select = 7, cHT-Cave

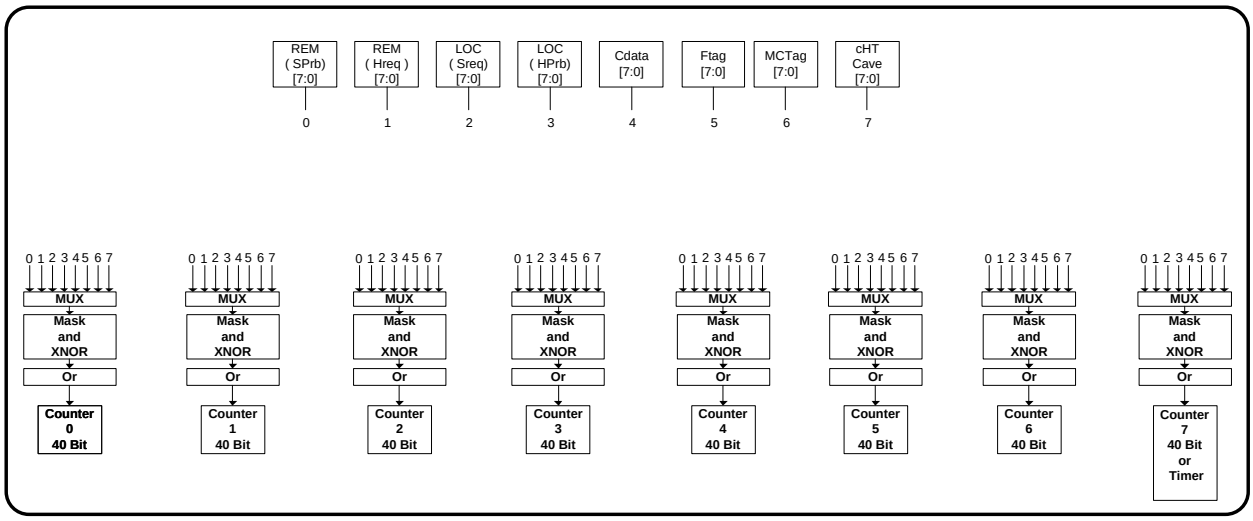

Figure 1: NumaChip Performance Counters

For each of the eight sources you can choose to program a signal:

Select = 0, REM/SPrb :

```
//
//      7 - SCC-Request Invalidate        (shared   => invalid)
//      6 - SCC-Request Read              (modified => shared)
//      5 - SCC-Request Read and Invalidate (modified => invalid)
//      4 - SCC-Request Aliased Invalidate (shared   => invalid)
//      3 - SCC-Request Aliased Read and Invalidate (modified => invalid)
//      2 - SCC-Request with SPrb conflict
//      1 - SCC-Request with HReq conflict
//      0 - Cache data access
//
```

Select = 1, REM/HReq:

```
//      7 - HT-Request start processing
//      6 - HT-Request with ctag miss
//      5 - HT-Request with ctag hit
//      4 - HT-Request with HReq conflict
//      3 - HT-Request with SPrb conflict
//      2 - HT-command unknown
//      1 - Broadcast messages
//      0 - Direct interrupt (no broadcast)
//
```

Select = 2, LOC/SReq:

```
//      7 - Interrupt request
//      6 - Config Space request
//      5 - VictimBlk request
//      4 - VictimBlk conflict
//      3 - SCC conflict
//      2 - SCC discard
//      1 - SCC request (all)
//      0 - Error in interrupt
//
```

Select = 3, LOC/HPrb:

```
//      7 - HT lock pending
//      6 - VictimBlk conflict
//      5 - HT-probe with next-state=invalidate
//      4 - SCC retries
//      3 - SCC requests
//      2 - HT-probe on own request
//      1 - HT-probe with next-state=shared
//      0 - HT-probe to non-shared memory
//
```

Select = 4, CData:

```
//      7 - CData write request from REM/HReq
//      6 - CData write request from REM/HReq accepted
//      5 - CData read request from REM/HReq
//      4 - CData read request from REM/HReq accepted
//      3 - CData write request from REM/SPrb
//      2 - CData write request from REM/SPrb accepted
//      1 - CData read request from REM/SPrb
//      0 - CData read request from REM/SPrb accepted
//
```

Select = 5, FTag:

```
//      7 - Tag update valid from MCTag
//      6 - Tag read valid from MCTag
//      5 - MCTag request
//      4 - Tag response valid from MCTag to LOC/HPrb
//      3 - Unused
//      2 - Tag response valid from prefetch to LOC/HPrb
//      1 - Unused
//      0 - Tag request from LOC/HPrb
//
```

Select = 6, MCTag:

```
//      7 - Unused
//      6 - Prefetch buffer address hit
//      5 - Prefetch buffer full hit
//      4 - Tag request from REM/HReq
//      3 - CTag cache hit
//      2 - CTag cache miss
//      1 - DRAM read request
//      0 - DRAM read request delayed
//
```

Select = 7, cHT-Cave:

```
//      7 - Outgoing HT-Probe
//      6 - Outgoing HT-Response
//      5 - Outgoing posted HT-Request
//      4 - Outgoing non-posted HT-Request
//      3 - Incoming HT-Probe
//      2 - Incoming HT-Response
//      1 - Incoming posted HT-Request
//      0 - Incoming non-posted HT-Request
//
```

The performance counters themselves are 40-bit registers. The NumaChip user space library gives full access to all of them.

3 NUMACHIP USER SPACE LIBRARY

The NumaChip user space library is located in https://github.com/numascale
To checkout use: git@github.com:numascale/nc-utils.git.
The user space library will then be located in:

user@compileserver:~/github/nc-utils/os/lib$

You can compile the library and master node applications by typing doing:

user@compileserver:~/github/nc-utils/os$ make

The *make* operation will generate a *libnumachip_user.so* file that is linked to by e.g the *nc_perf* application.

The NumaChip user space library enables access to a set of functions that enable communication with all the NumaChips in the NumaConnect Single Image System. The API is defined in:

user@compileserver:~/github/nc-utils/os/numanhip_user.h

For the most relevent combination of performance counters programming there exists a NumaChip performance counter test library. The API is defined in:

user@compileserver:~/github/nc-utils/os/lib/pcounter_test.h

The first version of the NumaChip User Space Library uses the *.json file in order to identify all the NumaChips in the Single Image System. The json file is a configuration file needed by the bootloader to set up the Single Image System over a set of machines. For a small 4 machine setup it may look like this:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_stat_d# cat fabric-loop.json
{"fabric": {
        "x-size": 4,
        "y-size": 0,
        "z-size": 0,
        "nodes": [
                {"desc": "loop",  "uuid": 672, "sciid": "0x000", "partition": 0, "osc": 0,
"sync-only": 0},
                {"desc": "loop-06",  "uuid": 673, "sciid": "0x003", "partition": 0, "osc":
1, "sync-only": 0},
                {"desc": "loop-07",  "uuid": 308, "sciid": "0x001", "partition": 0, "osc":
0, "sync-only": 0},
                {"desc": "loop-08",  "uuid": 309, "sciid": "0x002", "partition": 0, "osc":
1, "sync-only": 0},
        ],
        "partitions": [
                {"master": "0x000", "builder": "0x000"},
        ]
}}
root@loop:/home/user/github/nc-utils/os/nc_test/nc_stat_d#
```

4 NUMACHIP PERFORMANCE STATISTICS COMMANDLINE TOOL – NC_PERF

A NumaChip performance statistics commandline tool, nc_perf has been created for your convienience. It builds on top of the NumaChip user space library and the performance counter test library and provides full programming access to the performance counters described in chapter 2.
You can compile the library and master node applications by typing doing:

```
user@compileserver:~/github/nc-utils/os$ make
```

In order to link with the NUMACHIP user space libarary you have to do:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# export
LD_LIBRARY_PATH=/home/user/github/nc-utils/os/lib:$LD_LIBRARY_PATH
```

A number of test scripts to efficiently operate nc_perf has been added:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ls
fabric-loop.json  Makefile              nc_perf    nc_perf_loop_all_cnt7.sh
nc_perf_probe.sh     nc_perf_short_loop_all.sh  nc_perf_test_node_0.sh
false_params.sh      mask_twice_busy_test.sh  nc_perf.c  nc_perf_loop_all.sh
nc_perf_read_all.sh  nc_perf_stop_all.sh       select-twice-test.sh
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#
```

4.1.1 nc_perf test scripts

If you look at the test scripts you will get an impression on how to program a counter e.g:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# cat
nc_perf_test_node_0.sh
echo "Clear counters - node 0"
./nc_perf fabric-loop.json -counter-clear 0 0
./nc_perf fabric-loop.json -counter-clear 0 1
./nc_perf fabric-loop.json -counter-clear 0 2
./nc_perf fabric-loop.json -counter-clear 0 3

echo "Select them - only node 0"
./nc_perf fabric-loop.json -counter-select 0 0 1
./nc_perf fabric-loop.json -counter-select 0 1 1
./nc_perf fabric-loop.json -counter-select 0 2 6
./nc_perf fabric-loop.json -counter-select 0 3 6

echo "Mask node 0"
./nc_perf fabric-loop.json -counter-mask 0 0 6
./nc_perf fabric-loop.json -counter-mask 0 1 5
./nc_perf fabric-loop.json -counter-mask 0 2 3
./nc_perf fabric-loop.json -counter-mask 0 3 2

echo "read counter - node 0"
./nc_perf fabric-loop.json -counter-read 0 0
./nc_perf fabric-loop.json -counter-read 0 1
./nc_perf fabric-loop.json -counter-read 0 2
./nc_perf fabric-loop.json -counter-read 0 3
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#
```

4.1.2 nc_perf help menu

nc_perf comes with an extensive help menu:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf
./nc_perf <help>/<json_file>
[-counter-select <node_index>|<'all'> <counterno> <mux value>]
[-counter-mask <node_index>|<'all'> <counterno> <mask value> ]
[-counter-stop <node_index>|<'all'> <counterno>]
[-counter-read <node_index>|<'all'> <counterno> ]
[-counter-clear <node_index>|<'all'> <counterno> ]
[-counter-start <node_index>|<'all'> <counterno> <mux value> <mask value>]
```

4.1.3 Clear counters

In order to learn more about the *counter-clear* command you may type:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-
clear
Argument 2: -counter-clear
----------------------------------------
INVOLVED REGISTERS:---------------------
G3xF78 Select Counter
G3xFA0 Compare and Mask of counter 0
G3xFA4 Compare and Mask of counter 1
G3xFA8 Compare and Mask of counter 2
G3xFAC Compare and Mask of counter 3
G3xFB0 Compare and Mask of counter 4
G3xFB4 Compare and Mask of counter 5
G3xFB8 Compare and Mask of counter 6
G3xFBC Compare and Mask of counter 7
G3xFC0 Performance counter 0 40-Bit (Upper Bits)
G3xFC4 Performance counter 0 40-Bit (Lower Bits)
G3xFC8 Performance counter 1 40-Bit (Upper Bits)
G3xFCC Performance counter 1 40-Bit (Lower Bits)
G3xFD0 Performance counter 2 40-Bit (Upper Bits)
G3xFD4 Performance counter 2 40-Bit (Lower Bits)
G3xFD8 Performance counter 3 40-Bit (Upper Bits)
G3xFDC Performance counter 3 40-Bit (Lower Bits)
G3xFE0 Performance counter 4 40-Bit (Upper Bits)
G3xFE4 Performance counter 4 40-Bit (Lower Bits)
G3xFE8 Performance counter 5 40-Bit (Upper Bits)
G3xFEC Performance counter 5 40-Bit (Lower Bits)
G3xFF0 Performance counter 6 40-Bit (Upper Bits)
G3xFF4 Performance counter 6 40-Bit (Lower Bits)
G3xFF8 Performance counter 7 40-Bit (Upper Bits)
G3xFFC Performance counter 7 40-Bit (Lower Bits)
----------------------------------------
INVOLVED API FUNCTION:--------------------
void NumaChip_clear_pcounter(struct NumaChip_context *cntxt,
                             uint32_t counterno,
                             nc_error_t *error);
----------------------------------------
EXAMPLE:---------------------------------
Clear counter by deleting Performance counter registry values,
deselecting counter and clearing mask by writing api.
Clear counter 0:
----------------------------------------
```

```
OPERERATION:----------------------------
NumaChip_clear_pcounter(cntxt[node],0, &retval);
----------------------------------------
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#
```

4.1.4 Select counters

In order to learn more about the *counter-select* command you may type:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-select
Argument 2: -counter-select
----------------------------------------
INVOLVED REGISTERS:---------------------
G3xF78 Select Counter
----------------------------------------
Reset: 0000 0000h
----------------------------------------
Bits    Description
----------------------------------------
31      RO Reserved
30:28   RW Select Counter 7:
27      RO Reserved
26:24   RW Select Counter 6:
23      RO Reserved
22:20   RW Select Counter 5:
19      RO Reserved
18:16   RW Select Counter 4:
15      RO Reserved
14:12   RW Select Counter 3:
11      RO Reserved
10:8    RW Select Counter 2:
7       RO Reserved
6:4     RW Select Counter 1:
3       RO Reserved
2:0     RW Select Counter 0:
----------------------------------------
ALLOWED VALUES--------------------------
----------------------------------------
Select Counter eventreg:
----------------------------------------
 7 - cHT Cave [7:0]
 6 - MCTag [7:0]
 5 - FLAG [7:0]
 4 - CDATA [7:0]
 3 - LOC (HPrb)
 2 - LOC (SPrb) [7:0] -
 1 - REM (Hreq) [7:0] - Remote (L4) cache
 0 - REM (SPrb) [7:0] - Probes from SCC

----------------------------------------
INVOLVED API FUNCTION:------------------
void NumaChip_select_pcounter(struct NumaChip_context *cntxt,
                              uint32_t counterno,
                              uint32_t eventreg,
                              nc_error_t *error);
----------------------------------------
EXAMPLE:--------------------------------
```

Select counter 0 for mux: 1 - REM (Hreq):

OPERERATION:----------
NumaChip_select_pcounter(cntxt[node],0,0x1, &retval);
----
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#

4.1.5 Mask counters

In order to learn more about the *counter-mask* command you may type:

root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-mask
Argument 2: -counter-mask

INVOLVED REGISTERS:----------
G3xF9C Timer for ECC / Counter 7 (if you select
counter 7, then we will set this register for you.)
G3xFA0 Compare and Mask of counter 0
G3xFA4 Compare and Mask of counter 1
G3xFA8 Compare and Mask of counter 2
G3xFAC Compare and Mask of counter 3
G3xFB0 Compare and Mask of counter 4
G3xFB4 Compare and Mask of counter 5
G3xFB8 Compare and Mask of counter 6
G3xFBC Compare and Mask of counter 7

Reset: 0000 0000h

Bits Description

31:16 RO Reserved
15:8 RW Compare of performance counter
7:0 RW Mask of performance counter

ALLOWED VALUES----------

NumaChip_mask_counter - Apply counter mask

Select = 0, REM/SPrb :

7 - SCC-Request Invalidate (shared => invalid)
6 - SCC-Request Read (modified => shared)
5 - SCC-Request Read and Invalidate (modified => invalid)
4 - SCC-Request Aliased Invalidate (shared => invalid)
3 - SCC-Request Aliased Read and Invalidate (modified => invalid)
2 - SCC-Request with SPrb conflict
1 - SCC-Request with HReq conflict
0 - Cache data access

Select = 1, REM/HReq :

7 - HT-Request start processing
6 - HT-Request with ctag miss
5 - HT-Request with ctag hit
4 - HT-Request with HReq conflict
3 - HT-Request with SPrb conflict
2 - HT-command unknown

1 - Broadcast messages
0 - Direct interrupt (no broadcast)

Select = 2, LOC/SReq :

7 - Interrupt request
6 - Config Space request
5 - VictimBlk request
4 - VictimBlk conflict
3 - SCC conflict
2 - SCC discard
1 - SCC request (all)
0 - Error in interrupt

Select = 3, LOC/HPrb :

7 - HT lock pending
6 - VictimBlk conflict
5 - HT-probe with next-state=invalidate
4 - SCC retries
3 - SCC requests
2 - HT-probe on own request
1 - HT-probe with next-state=shared
0 - HT-probe to non-shared memory

Select = 4, CData :

7 - CData write request from REM/HReq
6 - CData write request from REM/HReq accepted
5 - CData read request from REM/HReq
4 - CData read request from REM/HReq accepted
3 - CData write request from REM/SPrb
2 - CData write request from REM/SPrb accepted
1 - CData read request from REM/SPrb
0 - CData read request from REM/SPrb accepted

Select = 5, FTag :

7 - Tag update valid from MCTag
6 - Tag read valid from MCTag
5 - MCTag request
4 - Tag response valid from MCTag to LOC/HPrb
3 - Unused
2 - Tag response valid from prefetch to LOC/HPrb
1 - Unused
0 - Tag request from LOC/HPrb

Select = 6, MCTag :

7 - Unused
6 - Prefetch buffer address hit
5 - Prefetch buffer full hit
4 - Tag request from REM/HReq

3 - CTag cache hit
2 - CTag cache miss
1 - DRAM read request
0 - DRAM read request delayed

Select = 7, cHT-Cave :

7 - Outgoing HT-Probe
6 - Outgoing HT-Response
5 - Outgoing posted HT-Request
4 - Outgoing non-posted HT-Request
3 - Incoming HT-Probe
2 - Incoming HT-Response
1 - Incoming posted HT-Request
0 - Incoming non-posted HT-Request

Documentation is in hdl:
assign prfMask0 = CSR_H2S_G3xFA0[`prfMask0Range];
assign prfCom0 = CSR_H2S_G3xFA0[`prfCom0Range];
assign pc0Event = |(prfMask0 & (prfCom0 ~^ Sel0Event));

INVOLVED API FUNCTION:----------
Select mask by writing api:
*void NumaChip_mask_pcounter(struct NumaChip_context *cntxt,*
uint32_t counterno,
uint32_t mask,
*nc_error_t *error);*

EXAMPLE:----------
For selected counter 0 mux: 1 - REM (Hreq).
Mask 6 - HT-Request with ctag miss:

OPERERATION:----------
NumaChip_mask_pcounter(cntxt[node],0,6, &retval);
----
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#

4.1.6 Read counters

In order to learn more about the *counter-read* command you may type:

root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-read
Argument 2: -counter-read

INVOLVED REGISTERS:----------
G3xFC0 Performance counter 0 40-Bit (Upper Bits)
G3xFC4 Performance counter 0 40-Bit (Lower Bits)
G3xFC8 Performance counter 1 40-Bit (Upper Bits)
G3xFCC Performance counter 1 40-Bit (Lower Bits)
G3xFD0 Performance counter 2 40-Bit (Upper Bits)
G3xFD4 Performance counter 2 40-Bit (Lower Bits)
G3xFD8 Performance counter 3 40-Bit (Upper Bits)
G3xFDC Performance counter 3 40-Bit (Lower Bits)

G3xFE0 Performance counter 4 40-Bit (Upper Bits)
G3xFE4 Performance counter 4 40-Bit (Lower Bits)
G3xFE8 Performance counter 5 40-Bit (Upper Bits)
G3xFEC Performance counter 5 40-Bit (Lower Bits)
G3xFF0 Performance counter 6 40-Bit (Upper Bits)
G3xFF4 Performance counter 6 40-Bit (Lower Bits)
G3xFF8 Performance counter 7 40-Bit (Upper Bits)
G3xFFC Performance counter 7 40-Bit (Lower Bits)

INVOLVED API FUNCTION:----------
*uint64_t NumaChip_get_pcounter(struct NumaChip_context *cntxt,*
uint32_t counterno,
*nc_error_t *error);*

EXAMPLE:----------
Read the counters Performance counter registry values,e.g of counter 0:

OPERERATION:----------
uint64_t val=NumaChip_read_pcounter(cntxt[node],0 &retval);
----
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#

4.1.7 Stop counters

In order to learn more about the *counter-stop* command you may type:

root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-stop
Argument 2: -counter-stop

INVOLVED REGISTERS:----------
G3xF78 Select Counter
G3xFA0 Compare and Mask of counter 0
G3xFA4 Compare and Mask of counter 1
G3xFA8 Compare and Mask of counter 2
G3xFAC Compare and Mask of counter 3
G3xFB0 Compare and Mask of counter 4
G3xFB4 Compare and Mask of counter 5
G3xFB8 Compare and Mask of counter 6
G3xFBC Compare and Mask of counter 7
G3xF9C Timer for ECC / Counter 7 (if you select
counter 7, then we will set this register for you.)

INVOLVED API FUNCTION:----------
Stop counter by deselecting counter
and clearing mask by writing api:
*void NumaChip_stop_pcounter(struct NumaChip_context *cntxt,*
uint32_t counterno,
*nc_error_t *error);*

EXAMPLE:----------
Stop counter 0 by clearing the select and mask register
and corresponding counter register without clearing the number of counts:

OPERERATION:----------
NumaChip_stop_pcounter(cntxt[node],0, &retval);
----
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#

4.1.8 Start counters

In order to learn more about the *counter-start* command you may type:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf help -counter-
start
Argument 2: -counter-start
----------------------------------------
INVOLVED REGISTERS:---------------------
G3xF78 Select Counter
G3xF9C Timer for ECC / Counter 7 (if you select
counter 7, then we will set this register for you.)
G3xFA0 Compare and Mask of counter 0
G3xFA4 Compare and Mask of counter 1
G3xFA8 Compare and Mask of counter 2
G3xFAC Compare and Mask of counter 3
G3xFB0 Compare and Mask of counter 4
G3xFB4 Compare and Mask of counter 5
G3xFB8 Compare and Mask of counter 6
G3xFBC Compare and Mask of counter 7
G3xFC0 Performance counter 0 40-Bit (Upper Bits)
G3xFC4 Performance counter 0 40-Bit (Lower Bits)
G3xFC8 Performance counter 1 40-Bit (Upper Bits)
G3xFCC Performance counter 1 40-Bit (Lower Bits)
G3xFD0 Performance counter 2 40-Bit (Upper Bits)
G3xFD4 Performance counter 2 40-Bit (Lower Bits)
G3xFD8 Performance counter 3 40-Bit (Upper Bits)
G3xFDC Performance counter 3 40-Bit (Lower Bits)
G3xFE0 Performance counter 4 40-Bit (Upper Bits)
G3xFE4 Performance counter 4 40-Bit (Lower Bits)
G3xFE8 Performance counter 5 40-Bit (Upper Bits)
G3xFEC Performance counter 5 40-Bit (Lower Bits)
G3xFF0 Performance counter 6 40-Bit (Upper Bits)
G3xFF4 Performance counter 6 40-Bit (Lower Bits)
G3xFF8 Performance counter 7 40-Bit (Upper Bits)
G3xFFC Performance counter 7 40-Bit (Lower Bits)
----------------------------------------
INVOLVED API FUNCTION:-------------------
----------------------------------------
NumaChip_start_pcounter is just doing clear, select and mask in one step.
Checkout ./nc_perf help -counter-clear, -counter-select and -counter-mask for
more details for eventreg and mask.
----------------------------------------
void NumaChip_start_pcounter(struct NumaChip_context *cntxt,
                             uint32_t counterno,
                             uint32_t eventreg,
                             uint32_t mask,
                             nc_error_t *error);
----------------------------------------
EXAMPLE:---------------------------------
Clear counter by deleting Performance counter registry values,
deselecting counter and clearing mask by writing api:
Clear counter 0:
Select counter 0 for mux: 1 - REM (Hreq):
Select mask 6 - HT-Request with ctag miss:
----------------------------------------
OPERERATION:-----------------------------
NumaChip_start_pcounter(cntxt[node],0,1,6 &retval);
----------------------------------------
```

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#
```

4.1.9 nc_perf example

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# ./nc_perf_probe.sh
Clear all counters - node 0
Select them - only node 0
Mask node 0
read counter - node 0
Reading counter node 0 counterno 6 = 2733
Reading counter node 0 counterno 7 = 15156
root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf# cat nc_perf_probe.sh
echo "Clear all counters - node 0"
./nc_perf fabric-loop.json -counter-clear 0 6
./nc_perf fabric-loop.json -counter-clear 0 7

echo "Select them - only node 0"
./nc_perf fabric-loop.json -counter-select 0 6 7
./nc_perf fabric-loop.json -counter-select 0 7 7

echo "Mask node 0"
./nc_perf fabric-loop.json -counter-mask 0 6 7
./nc_perf fabric-loop.json -counter-mask 0 7 3

echo "read counter - node 0"
./nc_perf fabric-loop.json -counter-read 0 6
./nc_perf fabric-loop.json -counter-read 0 7

root@loop:/home/user/github/nc-utils/os/nc_test/nc_perf#
```

5 NUMACHIP PERFORMANCE STATISTICS GUI – NC_PSTATS_GUI

The NumaChip Performance Statistics GUI is written on top of Qt Widgets for Technical Applications, Qwt, http://qwt.sourceforge.net/. This means that you need to link to both Qt libaries, http://qt.nokia.com/ and Qwt libaries.

The NumaChip Performance Statistics GUI typically runs on machine that is seperated from the actual NumaConnect Single System Image Cluster.

Currently the the NumaChip Performance Statistics GUI operates on Windows 7 and X11 systems. As the NumaChip Performance Statistics GUI is written using Qwt it will run on any platform that Qwt supports, e.g Linux, Mac and Windows.

For the NumaChip Performance Statistics GUI simply type

c:\nc-utils\os\nc_gui\nc_pstat_gui\Debug> nc_pstat_gui.exe -cache <IP address of Single Image System Master Node Daemon>:<portno used by Single Image System Master Node Daemon>

like this

c:\nc-utils\os\nc_gui\nc_pstat_gui\Debug> nc_pstat_gui.exe -cache 172.16.100.186 7070

If you will like to play with the tool without having a Single Image System Master Node Daemon you may do

c:\nc-utils\os\nc_gui\nc_pstat_gui\Debug> nc_pstat_gui.exe –simulate <number of nodes>

If you do not want to start the application from commandline you can put this information in a bat-file, e.g nc_pstat_cache.bat.

The NumaChip Performance Statistics GUI shows 5 different graphs:

- NumaChip Cache Hitrate (%) snapshot (per second)/time
- NumaChip Cache Hitrate (%) distribution pr. NumaChip. Both the average Cache Hitrate and the current snapshot (last second) is displayed for each NumaChip in the system.
- The NumaChip Total Number of Transactions In/Out pr. NumaChip is displayed. By NumaChip Number of Transactions In/Out we mean cHT-Cave Incoming and Outgoing non-posted HT-Request
- The NumaChip Number of Transactions In/Out snapshot (per second)/time. By NumaChip Number of Transactions In/Out we mean cHT-Cave Incoming and Outgoing non-posted HT-Request
- The NumaChip Total Number of Probes pr. NumaChip is displayed. By NumaChip Number of Probes we mean Incoming/Outgoing probe HT-Request.

5.1.1 NumaChip Cache Hitrate snapshot (per second)/time

The tab shows a graph displaying cache hitrate (%) per second over time. The number of accesses to the cache per update is displayed in the legend text for each remote cache. The graph monitors the cache hit ratio for all remote caches (L4 NumaConnect type caches) in the NumaConnect Single Image System. The example below shows this information on a NumaConnect Single Image System using four NumaChips:

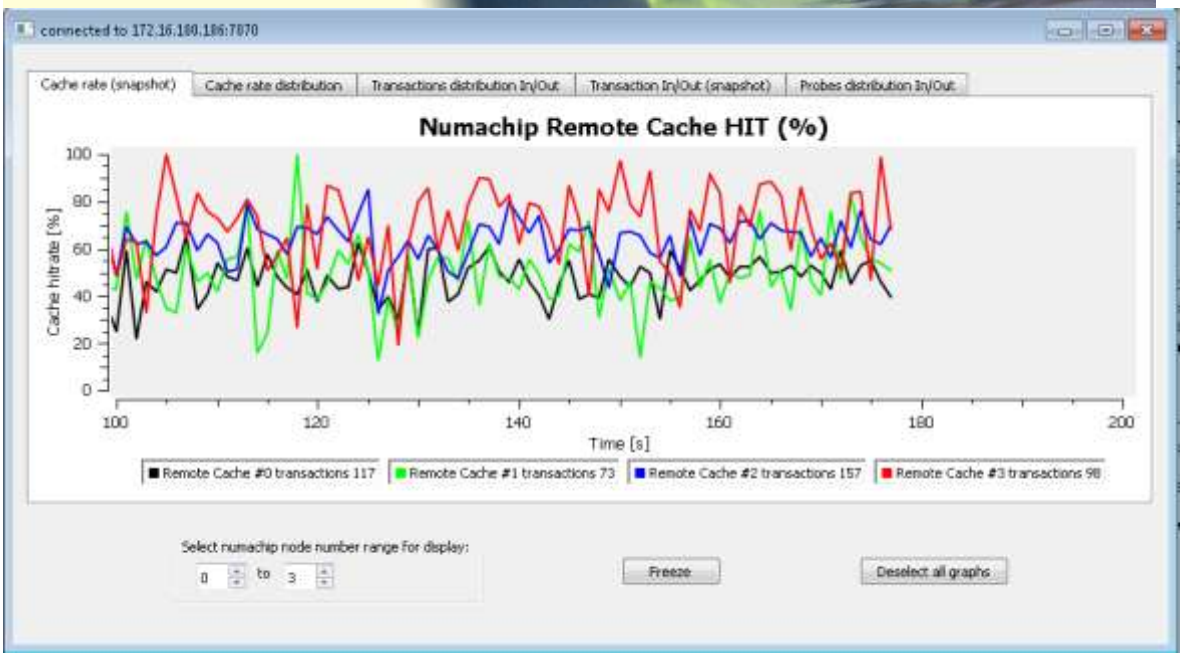

Figure 2: NumaChip Cache Hitrate (per second)/time

5.1.2 NumaChip Cache Hitrate distribution pr. NumaChip

This tab shows a histogram displaying a cache hitrate (%) snapshot (shown in blue color) on top of an average (shown in red) cache hitrate over time (red over blue gives purple). The number of accesses to the cache per update is displayed in the legend text for each remote cache. The histogram monitors the cache hit ratio for all remote caches (L4 NumaConnect type caches) in the NumaConnect Single Image System.

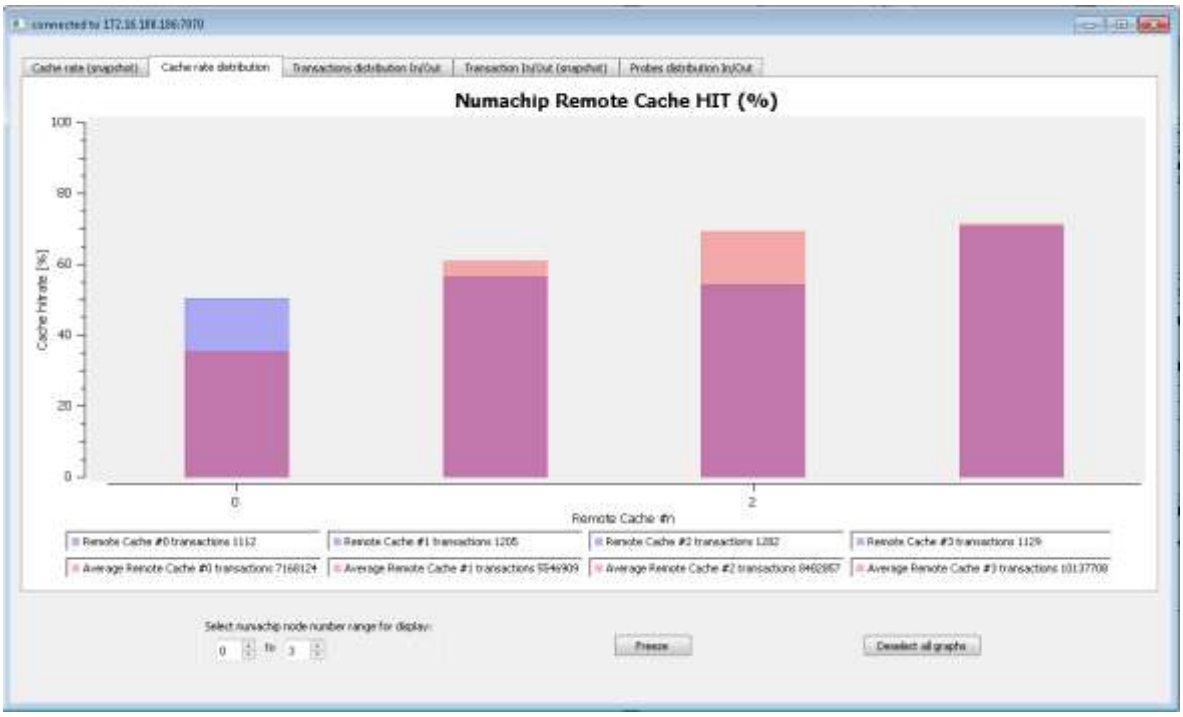

Figure 3: NumaChip Cache Hitrate distribution pr. NumaChip

5.1.3 NumaChip Total Number of Transactions In/Out

This tab shows a histogram displaying all Incoming (shown in blue color) non-posted HT-Request and Outgoing (shown in red color) non-posted HT-Request (red over blue gives purple) to the NumaChip (Cave). The non-posted HT-Requests gives an impression of the amount of actual hyper transport traffic in and out of each NumaChip. The histogram shows the total accumulated number of non-posted HT-Requests for each NumaChip in the NumaConnect Single Image System.

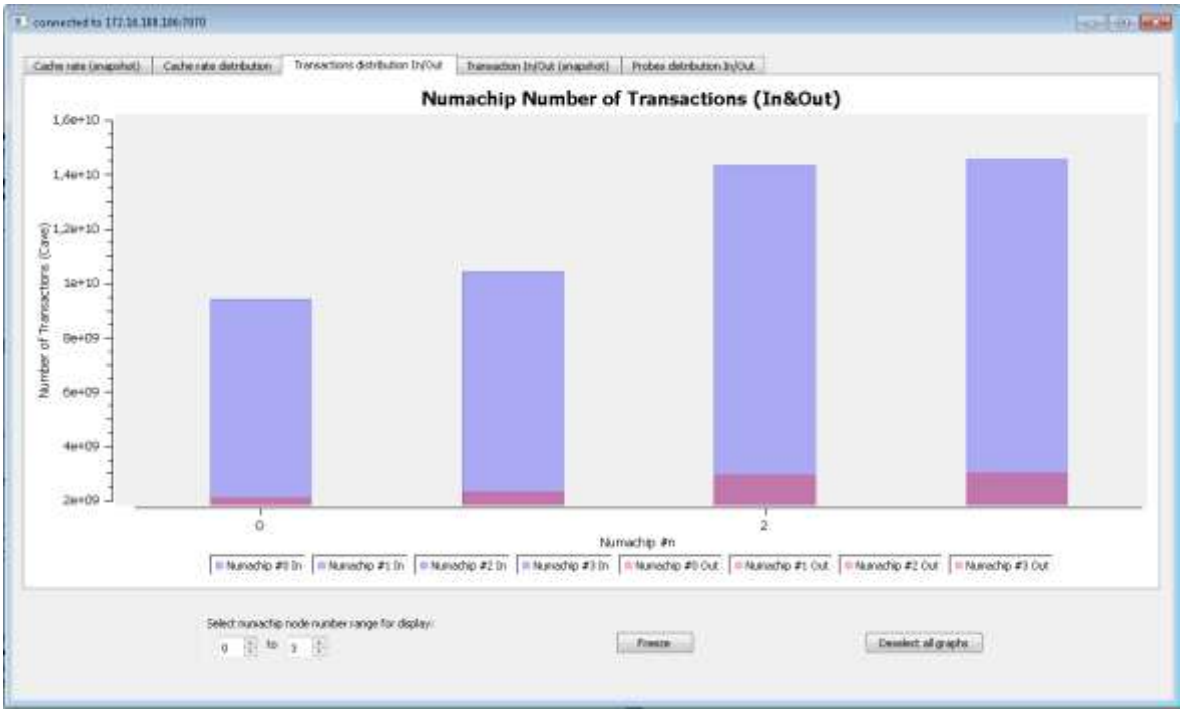

Figure 4: NumaChip Total Number of Transactions In/Out distribution pr. NumaChip

If you are just interested in the NumaChip Outgoing HT-Non Posted Requests you may deselect the Incoming legends and the axis will autoscale.

Figure 5: NumaChip Total Number of Outgoing Transactions distribution pr. NumaChip

5.1.4 NumaChip Number of Transactions In/Out snapshot (per second)/time.

This tab shows a histogram displaying all Incoming non-posted HT-Requests and Outgoing non-posted HT-Request to the NumaChip (Cave). The non-posted HT-Requests give an impression of the amount of actual hyper transport traffic in and out of each NumaChip. The histogram shows a snapshot of the total accumulated number of non-posted HT-Requests per second for each NumaChip in the NumaConnect Single Image System.

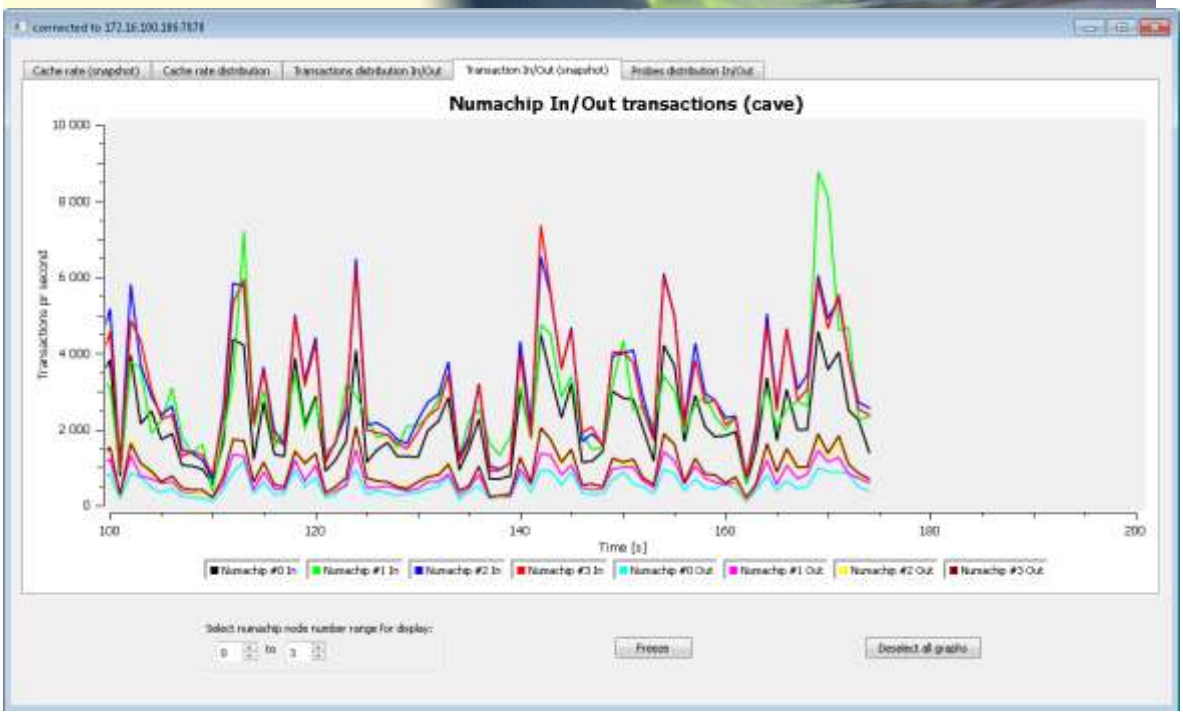

Figure 6: NumaChip Number of Transactions In/Out snapshot (per second)/time

If you are just interested in the NumaChip Outgoing HT-Non Posted Requests you may deselect the Incoming legends and the axis will autoscale.

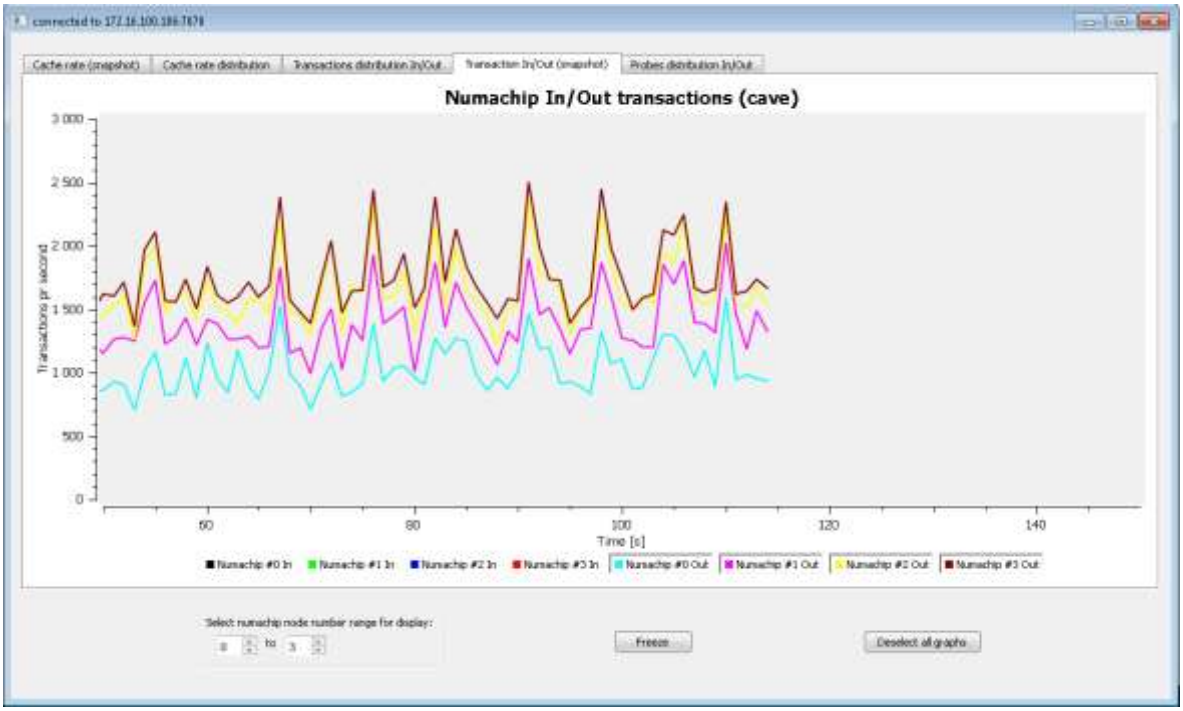

Figure 7: NumaChip Number of Transactions Out snapshot (per second)/time

5.1.5 NumaChip Total Number of Probes

This tab shows a histogram displaying all Incoming (shown in red color) HT-Probe and Outgoing (shown in blue color) HT-Probe (red over blue gives purple) to the NumaChip (Cave). The coherent hyper transport probes are necessary for maintaining the cache

coherency in L1, L2, L3 and L4. The histogram shows the total accumulated number of HT-Probes for each NumaChip in the NumaConnect Single Image System.

Figure 8: NumaChip Number of Probes In/Out snapshot (per second)/time

6 NC_STAT_D – SINGLE IMAGE SYSTEM MASTER NODE DAEMON

The nc*_stat_d* is a Linux application that will run on the master node of the NumaConnect Single Image System. It uses a set of the most popular NumaChip Performance Statistics sources and signals and runs on top of the NumaChip User Space Library. These are defined in a *struct* that is passed to the NumaChip Performance Statistics GUI.

```
struct cachestats_t {
    uint64_t hit; //counter_0 - Select = 1, REM/HReq value 6 - HT-Request with ctag
miss
    uint64_t miss; //counter_1 - Select = 1, REM/HReq value 5 - HT-Request with ctag
hit

    //From totmiss and tothit we can calculate avg hit/miss.
    uint64_t tothit; //counter_1 - Select = 1, REM/HReq value 5 - HT-Request with
ctag hit
    uint64_t totmiss; //counter_0 - Select = 1, REM/HReq value 6 - HT-Request with
ctag miss
    uint64_t cave_in; //counter_2 - Select = 7, cHT-Cave value 0 - Incoming non-
posted HT-Request
    uint64_t cave_out; //counter_3 - Select = 7, cHT-Cave value 4 - Outgoing non-
posted HT-Request
    uint64_t tot_cave_in; //counter_4 - Select = 7, cHT-Cave value 0 - Incoming non-
posted HT-Request
    uint64_t tot_cave_out; //counter_5 - Select = 7, cHT-Cave value 4 - Outgoing
non-posted HT-Request
    uint64_t tot_probe_in; //counter_6 - Select = 7, cHT-Cave value 3 - Incoming
probe HT-Request
    uint64_t tot_probe_out; //counter_7 - Select = 7, cHT-Cave value 7 - Outgoing
probe HT-Request
};
```

A smart reader might notice that these are 10 values, while there are only 8 performance counters. This is because that we do not use the performance counters to calculate *tothit* and *totmiss*, but just calculate *tothit* and *totmiss* from *hit* and *miss*. By doing so we lose a little accuracy, but since we are only interested in the ratio between (*tothitrate=tothit/tothit+totmiss*) them and not their accurate number we still get a relevant result. All the other numbers are accurate.

In order to start the *nc_stat_d* you may type:

```
root@loop:/home/user/github/nc-utils/os/nc_test/nc_stat_d# ls
fabric-loop.json  Makefile  nc_stat_d  nc_stat_d.c
root@loop:/home/user/github/nc-utils/os/nc_test/nc_stat_d# ./nc_stat_d
error, no port provided
root@loop:/home/user/github/nc-utils/os/nc_test/nc_stat_d# ./nc_stat_d 7070
```