HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
http://www.hunteng.co.uk
http://www.hunt-dsp.com

HUNT ENGINEERING

NETWORK FILE SYNTAX

- a common standard for Hunt Engineering tools such as Server/Loader & Heartconf

USER MANUAL

Software Version 4.13
Document Rev B
J.Thie 31/08/05

COPYRIGHT

WARRANTIES LIABILITY and INDEMNITIES

HUNT ENGINEERING warrants the hardware to be free from defects in the material and workmanship for 12 months from the date of purchase. Product returned under the terms of the warranty must be returned carriage paid to the main offices of HUNT ENGINEERING situated at BRENT KNOLL Somerset UK, the product will be repaired or replaced at the discretion of HUNT ENGINEERING.

Exclusions - If HUNT ENGINEERING decides that there is any evidence of electrical or mechanical abuse to the hardware, then the customer shall have no recourse to HUNT ENGINEERING or its agents. In such circumstances HUNT ENGINEERING may at its discretion offer to repair the hardware and charge for that repair.

Limitations of Liability - HUNT ENGINEERING makes no warranty as to the fitness of the product for any particular purpose. In no event shall HUNT ENGINEERING'S liability related to the product exceed the purchase fee actually paid by you for the product. Neither HUNT ENGINEERING nor its suppliers shall in any event be liable for any indirect, consequential or financial damages caused by the delivery, use or performance of this product.

Because some states do not allow the exclusion or limitation of incidental or consequential damages or limitation on how long an implied warranty lasts, the above limitations may not apply to you.

TECHNICAL SUPPORT

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng. co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

TABLE OF CONTENTS

INTRODUCTION 5

THE NETWORK DESCRIPTION FILE (HEART BOARDS) 6

Carrier Board Declaration 6
C6x Processor and Program Declaration 6
FPGA / HERONIO Declaration 8
GDIO Declaration 8
PCIF Declaration 8
Inter-Board Module Declaration (EM1C, EM1, EM2) 9
BDCONN / BDLINK / BDPATH Declaration 9
HEART Declaration 10
BDCAST Declaration 13
LISTEN Declaration 13
UMIRESET Declaration 14
BootSlot Declaration 14
Sample Network Description file for HEPC9 15

THE NETWORK DESCRIPTION FILE (HERON-BASE2) 17

Carrier Board Declaration 17
C6x Processor and Program Declaration 17
FPGA / HERONIO Declaration 18
PCIF Declaration 19
BootLink and BootPath Declaration 19
Host Path Declaration 20
UMIRESET Declaration 20
FIFONOBLOCK Declaration 21
Sample Network Description file for HERON-BASE2 (1) 21
Sample Network Description file for HERON-BASE2 (2) 22
Sample Network Description file for HERON-BASE2 (3) 22

THE NETWORK DESCRIPTION FILE (HEPC8) 24

Carrier Board Declaration 24
C6x Processor and Program Declaration 24
BootLink and BootPath Declaration 25
Host Path Declaration 26
FPGA / HERONIO Declaration 26
Sample Network Description file for HEPC8 27

THE NETWORK DESCRIPTION FILE (C4X SYSTEMS) 28

Carrier Board Declaration 28
C4x Processor and Program Declaration 28
BootLink and BootPath Declaration 29
Host Path Declaration 30
Sample Network Description file for HEPC3 30

NETWORK FILE SYNTAX 32

NETWORK DESCRIPTION FILE SYNTAX 32
Describing Boards 32
BD syntax 32
BDCONN / BDLINK / BDPATH syntax (HEART boards only) 33
C6 nodes syntax (HEPC8, HEART, and HERON-BASE2 boards only) 35
C4 nodes syntax (C4x carrier boards only) 35
FPGA nodes syntax (HEPC8, HEART, and HERON-BASE2 boards only) 36
GDIO nodes syntax (HEART boards only) 37
PCIF (Host Interface) nodes syntax (HEART and HERON-BASE2 boards only) 37

EM2/EM1/EM1C nodes syntax (HEART boards only) *38*
BOOTLINK / BOOTPATH syntax (C4x boards, HEPC8, and HERON-BASE2 only) *38*
HEART syntax (HEART boards only) *39*
BDCAST syntax (HEART boards only) *40*
LISTEN syntax (HEART boards only) *41*
BOOTSLOT syntax (HEART boards only) *42*
HOSTLINK syntax *42*
UMIRESET syntax (HEART and HERON-BASE2 boards only) *43*
FIFONOBLOCK syntax (HERON-BASE2 boards only) *43*
Example Network Description Files *43*

FEATURE SPOTLIGHT **46**
HEART FIFO RESET USING UMI 46
USING INTER-BOARD CONNECTORS 46
SERVER LINKS 48

TECHNICAL SUPPORT **49**

This document describes the network file syntax, as used by tools such as HeartConf and the Server/Loader. A network file is a simple ASCII file that you can create with edit tools such as NotePad (Windows), 'vi' (Linux), or C/C++ compiler IDE's such as Microsoft's Visual Developer (Visual C/C++).

A network file describes board and nodes in a HUNT ENGINEERING system, and their inter-connections (e.g.. comports, fifo's, fibre-links). A network file can also describe what fifo connections to create (for carrier boards that support HEART, such as the HEPC9.

The Network Description File (HEART boards)

The Network Description File is an ASCII file that lists all modules, carrier boards and their inter-connections, as well as HEART connections to be made. (Instead of 'modules' it is better to talk about 'nodes'; some modules may have more than 1 processor or programmable FPGA.) The following information must be present:

- A complete list of carrier boards (HEPC9, etc.).
- A complete list of the nodes (C6x) and programs to be loaded onto them.
- A complete list of the FPGA or HERONIO modules and their bit-streams.
- A complete list of all GDIO modules if HEART connections are to access them.
- A complete list of Host Interfaces (when connected nodes need to be served).
- A complete list of Inter-Board Connectors (EM2, EM1 or EM1C).
- A complete list of connections between boards (e.g.EM2 – EM2 channels).
- HEART connections (HEART boards such as the HEPC9).

Carrier Board Declaration

An example entry for declaring an HEPC9 is as follows:

```
BD API hep9a 0 0
```

The first item, BD, tells the Server/Loader that on this line a board is declared. The second item, API, tells the Server/Loader that the board is to be accessed via the API. (Obviously, the API must have been installed correctly for this to work, eventually.) The remainder is API related: API board name ("hep9a"), board number, and device number, respectively.

Please note that the board number is the number selected by the switch on the HEPC9. It has possible values ranging from 0 to 15. If this switch is set to, for example, 4, then your BD declaration becomes:

```
BD API hep9a 4 0
```

There is an optional keyword 'remote'. This indicates that a board's HSB and RESET should be accessed via another board (connected to it with Inter-Board Connector module links). Typically this is used when a board is not local, i.e. is not in the current PC, but instead, for example, is embedded or is in another PC.

C6x Processor and Program Declaration

For example, if there are two HERON modules inserted on the HEPC9 above, then the processors and programs to boot onto them are defined as follows:

```
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

The first item tells the Server/Loader that a 'C6x processor is declared. The second item tells the Server/Loader via what board this processor is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

To emphasise that the second item is not the board number or board switch, but the BD entry number, consider the following: -

```
BD API hep9a 3 0
BD API hep9a 5 0
c6 1 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

Module HERON1 is on board 1, that is, “hep9a 5”, and module HERON2 is on board 0, that is, “hep9a 3”.

The third item is the name of the processor. You can choose any name you like. The fourth item tells the Server/Loader what type of node it is: ROOT node or NORMAL node. A ROOT node has a direct connection to the PC where the Server/Loader is running and a NORMAL node is a processor at least 1 hop away from the Server/Loader PC. Legacy boards such as the HEPC8 and HEPC3 use such information. With the HEPC9 all nodes are effectively ROOT nodes, as all can be connected to the host. You can choose ROOT or NORMAL as you please, HeartConf and the Server/Loader will ignore this for HEPC9 (and other HEART boards). For network files to maintain some compatibility with the HEPC8 choose the first HERON module in your system to be ROOT, and all others NORMAL, per set of nodes on one board.

The fifth item is the Code Composer Studio ID. Code Composer Studio labels processors as they appear along the "JTAG scan path". This is not necessarily the same as the labels we put on them in the network description file. When you use the `-g` option, the Code Composer Studio ID is used to map program file onto node.

The sixth item is the HERON module's ID. On a HEPC9, the first slot will have ID 1, the second will have ID 2, the third ID3, and the fourth ID 4. But note that the HERON ID is made up of the HEPC9's board number (bits 7..4) and the slot number (bits 3..0). So if the HEPC9 board number switch is set to 4, the HERON IDs become 0x41, 0x42, 0x43 and 0x44. Thus with a HEPC9 board switch set at 0, we have:

```
BD API hep9a 0 0
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

but when the HEPC9 board switch is set to 4, we will have:

```
BD API hep9a 4 0
c6 0 HERON1 ROOT   (1) 0x41 heron1.out
c6 0 HERON2 NORMAL (0) 0x42 heron2.out
```

The last entry is the program name. This should be an executable file produced by the Texas Instruments' C compiler for the 'C6x. With the HEPC9 and other HEART boards, there may be 1 program file declared, but on other boards, such as the HEPC8, 2 program files may be declared.

Please note that for the ***HeartConf*** tool, the processor is not actually loaded. However, the parser still needs a program name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the program name, for example “a” or “no-file”. But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper filename. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***. A file fit for use with ***HeartConf*** may need to have proper program names specified before it works with the ***Server/Loader***.

FPGA / HERONIO Declaration

For example, if there is an FPGA module inserted on the HEPC9 above, then the module and program to boot onto them are defined as follows:

```
fpga 0 FPGA1 NORMAL   00000003 mybitstream.rbt
```

The first item tells the Server/Loader that a FPGA module is declared. The second item tells the Server/Loader via what board this module is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

The third item is the name of the module. You can choose any name you like. The fourth item tells the Server/Loader what type of node it is: always type NORMAL node.

The fifth item is the FPGA module's HERON ID. On a HEPC9, the first slot will have ID 1; the second will have ID 2, the third ID3, and the fourth ID 4. But note that the HERON ID is made up of the HEPC9's board number (bits 7..4) and the slot number (bits 3..0). So if the HEPC9 board number switch is set to 4, the HERON IDs become 0x41, 0x42, 0x43 and 0x44. Thus with a HEPC9 set at 0, we have:

```
BD API hep9a 0 0
fpga 0 FPGA1 NORMAL   0x03 mybitstream.rbt
```

but when the HEPC9 is set to 4, we will have:

```
BD API hep9a 4 0
fpga 0 FPGA1 NORMAL   0x43 mybitstream.rbt
```

The last entry is a bit-stream file name. This should be an rbt file produced by Xilinx tools.

Please note that for the ***HeartConf*** tool, the FPGA is not actually programmed. However, the parser still needs a bit-stream file name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the bit-stream file name, like "a" or "no-rbt". But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper bit-stream. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***.. A network file fit for use with ***HeartConf*** may need to have proper bit-stream file names specified before it works with the ***Server/Loader***.

GDIO Declaration

The GDIO declaration gives you a 'named module' that you can use in a HEART statement to create a HEART connection between the GDIO and another module. The GDIO statement serves no further purpose, as nothing is loaded onto a GDIO module and it cannot be programmed or configured. An example of a GDIO statement is:

```
GDIO 0 HEGD11 NORMAL   00000003
```

In this example the GDIO is situated in slot 3 of a HEPC9 with board switch 0.

PCIF Declaration

The PCIF declaration gives you a 'named entity' (denoting the host interface) that you can use in a HEART statement to create a HEART connection between the host interface and a module. The PCIF statement serves no further purpose, as it cannot be programmed or

configured. An example of a PCIF statement is:

```
PCIF 0 host NORMAL   00000005
```

Note that the 'slot id' used (00000005 in the example) must always be 5+(board switch*16). In this example the board switch was 0.

Inter-Board Module Declaration (EM1C, EM1, EM2)

With version 4.05 (released july 4, 2002), 3 inter-board modules are now supported by the network file syntax. They are the EM1C, EM1 and EM2. They can be used in the same fashion as any other node (such as a GDIO or a PCIF). The EM1C is an inter-board module with 2 comports, 1 in (fixed) and 1 out (fixed), without reset or HSB. This module is purely used for connecting a C4x to a C6x system but without any sharing of control between them. The EM1 has 2 comports as well, 1 comport in (fixed) and 1 comport out (fixed), but here reset and HSB are present. It can be used to connect two HEPC9's together. The EM2 is an inter-board module that can connect 2 or more HEPC9's together with high-speed links. There are 6 links in and 6 links out.

The EM1C/EM1/EM2 declaration gives you a 'named entity' (denoting the inter-board connector) that you can use in a BDCONN statement, and in a HEART statement to create a HEART connection between the inter-board connector and a module. The EM1C/EM1/EM2 statement serves no further purpose, as the inter-board connector cannot be programmed or configured. An example of an inter-board module statement is:

```
#keyword  boardno  name     type      heron-id

EM1C      0        myem1c   NORMAL    0x06
EM1       1        myem1    NORMAL    0x16
EM2       2        myem2    NORMAL    0x26
```

Note that the 'heron id' used must always be 6+(board switch*16). There can also be at most 1 inter-board module per board. To define connections you made between inter-board modules, use the BDCONN or BDLINK statement.

BDCONN / BDLINK / BDPATH Declaration

A BDCONN declaration defines a connection between 2 Inter-Board Connector modules. The Inter-Board Connector modules must have been defined earlier in the network file. An example of a BDCONN statement is:

```
EM2       0        em2a    NORMAL    0x16
EM2       1        em2b    NORMAL    0x26
BDCONN  em2a 1   em2b 2
```

This BDCONN statement uses defined Inter-Board Connector modules 'em2a' and 'em2b', both of which are of type EM2. The statement describes channel 1 of 'em2a' connected to channel 2 of 'em2b'. Note that the connection defined is duplex. In case you have hardware that implements a simplex (one way) connection, use the optional ONEWAY keyword:

```
EM1       0        em1a    NORMAL    0x16
EM1       1        em1b    NORMAL    0x26
BDCONN  em1a 1   em1b 2   oneway
```

This defines a connection from 'em1a' channel 1 to 'em1b' channel 2.

Similarly, a BDLINK declaration defines a connection between a HEPC9 and another HEPC9. The actual specifics of the connection depend on the particular Inter-Board Connector that you use. The BDLINK is a general statement that may describe many different Inter-Board Connector modules. It is therefore important that you specify the correct details of the particular Inter-Board Connector that you use. An example of a BDLINK statement is:

```
BDLINK 0 1 2 3
```

This means that an HEPC9, board index 0, channel 1 is connected to another HEPC9, board index 2, channel 3. 'Channel' relates to the cable 'slot' used with the Inter-Board Connector. For EM1's and EM1C's the channel number is always 0. For EM2's you have a choice of 6 channels 0..5.

A BDLINK statement defines a duplex connection. Use BDPATH to define a simplex connection. You would need two BDPATH declarations to emulate one BDLINK declaration. For example:

```
BOOTPATH 0 2 1 0
BOOTPATH 1 0 0 2
```
is equal to:
```
BOOTLINK 0 2 1 0
```

The links between Inter-Board Modules are not only used to create FIFO connections, but also to propagate HSB and reset. By default, the Server/Loader and HeartConf assume that any Inter-Board connection defined propagates HSB and reset. However, between inter-connected boards there may only be at most one HSB connection (path) and at most 1 reset connection (path). Or, in other words, 'loops' are not allowed. Keywords NOHSB and NORESET allow you to select what Inter-Board connections are not to be used to propagate HSB and reset. Simple example:

```
EM2        0        em2a     NORMAL    0x16
EM2        1        em2b     NORMAL    0x26
BDCONN    em2a 0    em2b 0
BDCONN    em2a 1    em2b 1
```

Board 0 is connected to board 1 via channels 0 and 1 of 'em2a'. This means that HSB is propagated through 2 paths from board 0 to board 1. Similarly, reset is propagated through 2 paths from board 0 to board 1. For example, to select channel 0 for HSB and reset:

```
EM2        0        em2a     NORMAL    0x16
EM2        1        em2b     NORMAL    0x26
BDCONN    em2a 0    em2b 0
BDCONN    em2a 1    em2b 1   NOHSB NORESET
```

HEART Declaration

The HEART declaration asks the Server/Loader and HeartConf to create a connection between slots, using the HEART ring. In previous sections it is shown how you can name any possible 'entity' on a carrier board (HERON modules, GDIO modules, host interfaces (PCIF), inter-board connectors (IBC), FPGA modules, and more). Using an entity's name you can define connections between it and another entity. I use the word 'entity' to make include both slots and devices such as the host interface. For example, to have the Server/Loader create a HEART connection between one HERON module and another:

```
#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  c6   0  HERON2 NORMAL (1)  00000002 heron2.out
#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART HERON1    0     HERON2    1         1
```

This will tell the Server/Loader to create a HEART connection from the HERON module in slot 1 to the HERON module in slot 2, using 1 timeslot. To create a duplex connection, you would have to add to this the following line:

```
  HEART HERON2    1     HERON1    0         1
```

As a second example, to have the Server/Loader create a HEART connection between a HERON module and a GDIO module and a host interface:

```
#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  Gdio 0  hegd1  NORMAL      00000002
  pcif 0  host   NORMAL      00000005

#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART hegd1     1     HERON1    2        1
  HEART HERON1    0     host      3        1
  HEART host      3     HERON1    0        1
```

This will tell the Server/Loader to create a HEART connection from the GDIO module in slot 2 to the HERON module in slot 1, using 1 timeslot. The GDIO is to output on fifo 1, and the HERON module can read the GDIO's data from fifo 2. The next two HEART statements will create a duplex connection between the HERON module and the host interface. A PC program can now communicate with the DSP by reading from or writing to FIFO 3. The DSP can communicate with the PC by reading from or writing to FIFO 0.

Specifying timeslots in a HEART statement

The last (sixth) parameter in a HEART statement is the number of timeslots. On a HEPC9 there's a total of 6 timeslots, each timeslot denoting about 63 Mb/sec. Therefore, if your application requires, for example, 100 Mb/sec between a HEGD9 and an FPGA, you can specify in a HEART statement to use 2 timeslots (allowing a maximum of 126 Mb/sec):

```
#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART hegd9     1     FPGA       2        2
```

The Server/Loader will try to 'map' the timeslots you requested onto actual timeslots. Given that resources are limited it may be that it's impossible to 'map' all requests. In that case the Server/Loader will report this and then will then halt execution.

You may also try to map timeslots yourself. There are two ways. One is the 't=' parameter,

and the other is the 'v=' parameter. With the 't=' parameter you specify sequentially what actual timeslots you want to use. Using the same example:

```
#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART hegd9     1     FPGA        2     t=0,2
```

This will tell the Server/Loader that 2 timeslots are required between the HEGD9 and the FPGA, timeslot 0 and timeslot 2. Timeslots 1, 3, 4, and 5 are unused.

With the 'v=' parameter you specify a 'mask' rather than a sequence. Eg timeslot 0 would be 0x1, timeslot 1 is 0x2, timeslot 2 is 0x4 and so on. Thus using two timeslots 0 and 2 would give a 'mask' of 0x5:

```
#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART hegd9     1     FPGA        2     v=5
```

Non Blocking Mode

HEART also supports a non-blocking mode. This means that the 'sender' of data will continue to send data even if when the 'receiver' is not able to read the data (fast enough). To select non-blocking mode, add the keyword NOBLOCK. Example:

```
#-------------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-------------------------------------------------
  HEART hegd1     1     HERON1    2    1   noblock
```

Server connections

The Server/Loader will assume that any processor node that is connected to a host interface want to be 'served'. That is, is assumes that such nodes want to execute functions like 'fwrite', 'fprintf' and other 'stdio' functions over that connection. The Server/Loader will both find direct node – host connections as well as connections that go via Inter-Board Connector modules.

Frequently you will want to create node – host connections for yourself, and you don't want the Server/Loader to use those connections for the Server. In that case, use the optional keyword NOSERVE. For example:

```
  HEART HERON1    0     host      3        1
  HEART host      3     HERON1    0        1

  HEART HERON1    1     host      4        1     NOSERVE
  HEART host      4     HERON1    1        1     NOSERVE
```

Host fifo 3 will then be served by the Server/Loader, executing Server/Loader stdio functions called on module HERON1. Host fifo 4 is free for use by yourself.

It has to be said that the Server/Loader will at most only use 1 link per processor node to serve stdio requests. Therefore, the NOSERVE keyword, in effect, selects a fifo connection that certainly isn't used by the Server/Loader to serve stdio requests. Moreover, if NOSERVE is used for all node – host connections for that node, then no fifo connection is available for serving stdio requests, and even if stdio requests are called on the node, they will not succeed. The Server/Loader is not able to pick up or recognise such 'errors'.

UMI reset of HEART FIFOs

Within HEART, there is a possibility to reset FIFO's. This is done by 'associating' a FIFO

with a UMI line (one or several). FIFO's that are 'associated' with a UMI line will be reset if that UMI line is set to active. To associate a FIFO with a UMI line, you can use the optional keyword UMI with a HEART statement; or use the stand-alone UMIRESET statement. The advantage of using the UMI keyword with a HEART statement is that all FIFO's between two nodes would be reset when the associated UMI line becomes active.

Example:

```
HEART HERON1     0      host       3          1      UMI 0
HEART host       3      HERON1     0          1      UMI 0,1
```

In the first HEART statement, FIFO 0 (out) of node HERON1 and FIFO 3 (in) of the host interface are 'associated' with UMI 0. In the second HEART statement, FIFO 3 (out) of the host interface and FIFO 0 (in) of node HERON1 is 'associated' with both UMI 0 and 1. Now, if UMI line 0 becomes active, all 4 FIFO's discussed are reset. And when UMI line 1 becomes active, FIFO 3 (out) of the host interface and FIFO 0 (in) of node HERON1 are reset.

BDCAST Declaration

The BDCAST declaration defines a one-way connection from a module into a HEART ring. The idea is that any other module can 'listen' to whatever is broadcast onto this ring. The broadcast is a 'named' entity and consumes ones whole ring.

An example of a BDCAST statement is:

```
BDCAST name module 0 1
```

This defines a broadcast called 'name', and the broadcaster is node 'module'. The 'module' node uses fifo 0 to broadcast over, and it uses 1 timeslot.

UMI reset of HEART FIFOs

Within HEART, there is a possibility to reset FIFO's. This is done by 'associating' a FIFO with a UMI line. You can select one or several UMI lines. FIFO's that are 'associated' with a UMI line will be reset if that UMI line is set to active. To associate a FIFO with a UMI line, you can use the optional keyword UMI with a BDCAST statement; or use the stand-alone UMIRESET statement. The advantage of using the UMI keyword with a BDCAST statement is that all FIFO's relating to a BDCAST would be reset when the associated UMI line becomes active.

Example:

```
BDCAST name module 0 1      UMI 0
```

In this BDCAST statement, FIFO 0 (out) of node 'module' is 'associated' with UMI 0. If there are listeners on other boards connected to the board 'module' is on, the FIFO's over which Inter-Board connections are made would also be 'associated' with that UMI line. The actual reset will take place only if UMI line 0 is made active.

LISTEN Declaration

The LISTEN statement makes a module 'listen' to a broadcast. This is done by creating a one-way link from a HEART ring onto a module. In the statement you tell the Server/ Loader what named broadcast to listen to. Eg, assuming a broadcast as defined above:

```
LISTEN name module 0
```

This means that 'module' is listening to broadcast 'name', reading data in over fifo 0.

Non Blocking Mode

HEART also supports a non-blocking mode. This means that the 'sender' of data will continue to send data even if when the 'receiver' is not able to read the data (fast enough). To select non-blocking mode, add the keyword NOBLOCK. Example:

```
#-----------------------------------------------
# LISTEN   bdcast   module   fifo
#-----------------------------------------------
  LISTEN   bbc1     HERON1    2       noblock
```

UMI reset of HEART FIFOs

Within HEART, there is a possibility to reset FIFO's. This is done by 'associating' a FIFO with a UMI line. You can select one or several UMI lines. FIFO's that are 'associated' with a UMI line will be reset if that UMI line is set to active. To associate a FIFO with a UMI line, you can use the optional keyword UMI with a LISTEN statement; or use the stand-alone UMIRESET statement.

Example:

```
LISTEN name module 0  UMI 0
```

In this LISTEN statement, FIFO 0 (in) of node 'module' is 'associated' with UMI 0. The actual reset will take place only if UMI line 0 is made active.

UMIRESET Declaration

Within HEART, there is a possibility to reset FIFO's. This is done by 'associating' a FIFO with a UMI line. You can select one or several UMI lines. FIFO's that are 'associated' with a UMI line will be reset if that UMI line is set to active. To associate a FIFO with a UMI line, you can use the stand-alone UMIRESET statement (or use the optional UMI keyword in a HEART, BDCAST or LISTEN statement). Example:

```
UMIRESET  heronx   0  in    0
UMIRESET  herony   2  out   0,1
```

In the first UMIRESET statement, fifo 0 (in: from HEART to node) of node 'heronx' is associated with UMI line 0. In the second UMIRESET statement, fifo 2 (out: from node to HEART) of node 'herony' is associated with UMI line 1. The actual reset (of heronx fifo 0 (in) and herony fifo 2 (out)) will take place if UMI line 0 is made active. If UMI line 1 is made active, only herony fifo 2 (out) will be reset.

BootSlot Declaration

This is rarely used, as modules are most easily configured using HEART statements. HERON modules have routing jumpers, and when used, one jumper connects the selected timeslot to a module's 'in' FIFO. Another jumper connects the selected timeslot to a module's 'out' FIFO. With a BOOTSLOT statement you define what timeslot you selected on a module ('in' and 'out' are assumed to have been chosen identical).

If the '-j' option is used, the Server/Loader will try to boot a processor module using the specified timeslot. The timeslot value must match the boot jumper as set on the HERON module that you intend to boot. If the '-j' option is **not** used, the boot jumper is ignored; the Server/Loader still uses the timeslot you specified, and boots the HERON module by creating a FIFO link by programming HEART. But the statement is optional in this case;

where it not there, the Server/Loader assumes timeslot 0. The following line shows you how to declare a timeslot value over which a HERON node is to be booted:

```
BOOTSLOT HERON4 1
```

The HERON module named 'HERON4' will be booted over timeslot 1.

Sample Network Description file for HEPC9

For one HEPC9 with one DSP, one FPGA module, a GDIO module, and a HERONIO module, the network description file could look like:

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        hep9a           0               0

#-------------------------------------------------------
# Nodes description
# ND      BD NDNAME   NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6      0  HERON1   ROOT   (0)   00000001 heron1.out
  heronio 0  heronio1 NORMAL       00000002 heronio1.rbt
  fpga    0  FPGA1    NORMAL       00000003 fpga1.rbt
  gdio    0  hegd2    NORMAL       00000004
  pcif    0  host     NORMAL       00000005

#---------------------------------------
# BOOTSLOT  ND_NAME  TIMESLOT
#---------------------------------------
  BOOTSLOT  HERON1   2

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0

#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART HERON1    0     host      3       1
  HEART host      3     HERON1    0       1
  HEART HERON1    1     heronio1  1       1
  HEART heronio1  1     HERON1    1       1
  HEART hegd2     5     FPGA1     4       1
  HEART FPGA1     3     HERON1    2       1
```

This will tell the Server/Loader to create a duplex connection between the DSP module and the host interface. A PC program can now communicate with the DSP by reading/ writing FIFO 3. The DSP can communicate with the PC by reading/writing FIFO 0.

The third and fourth statement will tell the Server/Loader to create a duplex connection between the DSP in slot 1 and the HERONIO in slot 2. The DSP can communicate with the HERONIO by reading/writing FIFO 1. The HERONIO can communicate with the DSP by reading writing FIFO 1.

The fifth statement asks the Server/Loader to create a one-way connection from the GDIO

in slot 4 to the FPGA in slot 3. The GDIO outputs on FIFO 5, and the FPGA reads the GDIO data in at FIFO 4.

The sixth statement asks the Server/Loader to create a one-way connection from the FPGA in slot 3 to the DSP in slot 1. The DSP can read FPGA data from FIFO 2, but cannot write data back to the FPGA (in this example). The FPGA outputs its data onto FIFO number 3.

So in this example we have a GDIO (5) -> (4) FPGA (3) -> (2) DSP 'pipeline'. The DSP has duplex communications with the HERONIO over FIFO 1, and with the PC over FIFO 0. The PC can communicate then with DSP over host FIFO 3.

The Network Description File is an ASCII file that lists all carrier boards, modules and their inter-connections. (Instead of 'modules' it is better to talk about 'nodes'; some modules may have multiple processors or FPGA's.) The following information must be present:

- A complete list of carrier boards (HERON-BASE2)
- A complete list of the nodes (C6x) and programs to be loaded onto them.
- A complete list of the FPGA or HERONIO modules and their bit-streams.
- A complete list of boot link connections.
- Route to host.

Carrier Board Declaration

An example entry for declaring an HERON-BASE2 is as follows:

```
BD API heb2a 0 0
```

The first item, BD, tells the Server/Loader that on this line a board is declared. The second item, API, tells the Server/Loader that the board is to be accessed via the API. (Obviously, the API must have been installed correctly for this to work, eventually.) The remainder is API related: API board name ("heb2a"), board number, and device number, respectively.

Please note that the board number is the number selected by the switch on the HERON-BASE2. It has possible values ranging from 0 to 15. If this switch is set to, for example, 4, then your BD declaration becomes:

```
BD API heb2a 4 0
```

C6x Processor and Program Declaration

For example, if there are two HERON modules inserted on the HERON-BASE2 above, then the processors and programs to boot onto them are defined as follows:

```
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

The first item tells the Server/Loader that a 'C6x processor is declared. The second item tells the Server/Loader via what board this processor is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

The third item is the name of the processor. You can choose any name you like. The third item tells the Server/Loader what type of node it is: ROOT node or NORMAL node. A ROOT node has a direct connection to the PC where the Server/Loader is running and a NORMAL node is a processor at least 1 hop away from the Server/Loader PC. There must be exactly one ROOT node in a network description file.

The fourth item is the Code Composer Studio ID. Code Composer Studio labels processors as they appear along the "JTAG scan path". This is not necessarily the same as the labels we put on them in the network description file. When you use the `-g` option, the Code Composer Studio ID is used to map program file onto node.

The fifth item is the HERON module's ID. On a HERON-BASE2, the first slot will have ID 1, and the second slot will have ID 2. But note that the HERON ID is made up of the HERON-BASE2's board number (bits 7..4) and the slot number (bits 3..0). So if the HERON-BASE2 board number switch is set to 4, the HERON IDs become 0x41 and 0x42. Thus with a HERON-BASE2 board switch set at 0, we have:

```
BD API heb2a 0 0
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

but when the HERON-BASE2 board switch is set to 4, we will have:

```
BD API heb2a 4 0
c6 0 HERON1 ROOT   (1) 0x41 heron1.out
c6 0 HERON2 NORMAL (0) 0x42 heron2.out
```

The last entry is the program name. This should be an executable file produced by the Texas Instruments' C compiler for the 'C6x. With the HERON-BASE2, 2 program files may be declared. In case 2 are declared, the first is executed first, the second program second. You cannot just use any file as the first file. The Server/Loader expects an executable program that sends a specific amount of data back to the Server/Loader, and which then "resets" the processor. Two such programs are present in `\hesl\etc\c6x\eeprom`. They are `eeprom62.out` and `eeprom67.out`. The processor entries would then become:

```
c6 0 HERON1 ROOT   (1) 00000001 eeprom62.out heron1.out
c6 0 HERON2 NORMAL (0) 00000002 eeprom67.out heron2.out
```

assuming that the second processor is a 'C67xx processor, and the first a 'C62xx. You would only use eeprom62/67.out if you use the `-c` option of the Server/ Loader.

Please note that for the ***HeartConf*** tool, the processor is not actually loaded. However, the parser still needs a program name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the program name, for example “a” or “no-file”. But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper filename. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***. A file fit for use with ***HeartConf*** may need to have proper program names specified before it works with the ***Server/Loader***.

FPGA / HERONIO Declaration

For example, if there is an FPGA module inserted on the HERON-BASE2 above, then the module and program to boot onto them are defined as follows:

```
fpga 0 FPGA1 NORMAL   0x1 mybitstream.rbt
```

The first item tells the Server/Loader that a FPGA module is declared. The second item tells the Server/Loader via what board this module is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

The third item is the name of the module. You can choose any name you like. The fourth item tells the Server/Loader what type of node it is: always type NORMAL node.

The fifth item is the FPGA module's HERON ID. On a HERON-BASE2, the first slot will have ID 1 and the second will have ID 2. But note that the HERON ID is made up of the HERON-BASE2's board number (bits 7..4) and the slot number (bits 3..0). So if the

HERON-BASE2 board number switch is set to 4, the HERON IDs become 0x41 and 0x42. Thus with a HERON-BASE2 set at 0, we have:

```
BD API heb2a 0 0
fpga 0 FPGA1 NORMAL   0x03 mybitstream.rbt
```

but when the HERON-BASE2 is set to 4, we will have:

```
BD API heb2a 4 0
fpga 0 FPGA1 NORMAL   0x43 mybitstream.rbt
```

The last entry is a bit-stream file name. This should be an rbt file produced by Xilinx tools.

Please note that for the ***HeartConf*** tool, the FPGA is not actually programmed. However, the parser still needs a bit-stream file name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the bit-stream file name, like "a" or "no-rbt". But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper bit-stream. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***. A network file fit for use with ***HeartConf*** may need to have proper bit-stream file names specified before it works with the ***Server/Loader***.

PCIF Declaration

The PCIF declaration gives you a 'named entity' (denoting the host interface) that you can use in a UMIRESET or FIFONOBLOCK statement. The PCIF statement serves no further purpose, as it cannot be programmed or configured. An example of a PCIF statement is:

```
PCIF 0 host NORMAL   00000005
```

Note that the 'slot id' used (00000005 in the example) must always be 5+(board switch*16). In this example the board switch was 0.

BootLink and BootPath Declaration

The Server/Loader needs to know via what FIFOs it should boot NORMAL nodes. The following line shows you how to declare a connection between processors heron1 and heron2 (as declared in a previous section).

```
BOOTLINK HERON1 1 HERON2 0
```

The first entry, BOOTLINK, tells the Server/Loader that this line declares a connection between one processor and an adjacent processor. In this example, processor HERON1's FIFO 1 is connected to processor HERON2's FIFO 0.

BOOTLINK declares a two-way connection: from HERON1 to HERON2, and from HERON2 to HERON1. Processor HERON 1 can send data to processor HERON2 via FIFO 2, and can also read data from processor HERON2 from FIFO 2. Processor HERON2 can send data to HERON1 via FIFO 0, and can also read data from HERON1 from FIFO 0.

However, this may not always be the case. In case of one-way connections, you can use:

```
BOOTPATH HERON1 1 HERON2 0
```

This declares only a connection from HERON1 to HERON2. You would need two

BOOTPATH declarations to emulate one BOOTLINK declaration. For example:

```
BOOTPATH HERON1 1 HERON2 0
BOOTPATH HERON1 0 HERON2 1
```
is equal to:
```
BOOTLINK HERON1 1 HERON2 0
```

Host Path Declaration

The connection between the ROOT node and the host PC that runs the Server/Loader is declared as follows:

```
HOSTLINK          0
```

The first entry (HOSTLINK) tells the Server/Loader that this line declares what FIFO or link of the ROOT node is connected to the host PC. This declaration defines a two-way link. It tells the Server/Loader that the ROOT node can both read from the host PC and write to the host PC via FIFO or link number 0.

In some cases there may not be a two-way connection, but only a one-way connection. In that case you can use:

```
FROMHOST          0
```

when the ROOT node can read data from the host PC over FIFO or link 0. And:

```
TOHOST            0
```

when the ROOT node can write data to the host PC over FIFO or link 0.

UMIRESET Declaration

With the HERON-BASE2, there is a possibility to reset FIFO's. This is done by 'associating' a FIFO with a UMI line. You can select one or several UMI lines. FIFO's that are 'associated' with a UMI line will be reset if that UMI line is set to active. To associate a FIFO with a UMI line, you can use the stand-alone UMIRESET statement (or use the optional UMI keyword in a BOOTLINK or BOOTPATH statement). Example:

```
UMIRESET  heron1   0  in    0
UMIRESET  heron2   1  out   0,1
```

In the first UMIRESET statement, fifo 0 (in: from host to slot 1) of node 'heron1' is associated with UMI line 0. In the second UMIRESET statement, fifo 1 (out: from slot 2 to host) of node 'heron2' is associated with UMI lines 0 and 1. The actual reset (of heron1 fifo 0 (in) and heron2 fifo 2 (out)) will take place if UMI line 0 is made active. If UMI line 1 is made active, only heron2 fifo 2 (out) will be reset.

Alternatively, you can use the optional keyword 'UMI' after a BOOTLINK / BOOTPATH statement. Example:

```
  BOOTLINK HERON1  1  HERON2  0  umi 0
```

In this statement UMI line 0 is associated with the fifo from slot 1 (HERON1) to slot 2 (HERON2), assuming HERON1 and HERON2 definitions as used in previous sections, and also with the fifo from slot 2 to slot 1.

UMIRESET can also be used with the host module fifos. For example:

```
pcif    0  host   NORMAL       00000005
UMIRESET   host   0  in    0
```

There is some degree of redundancy here, as, for example, host fifo 0 out is the same as slot 1 fifo 0 in (please refer to the HERON-BASE2 manual for fifo connections).

FIFONOBLOCK Declaration

The HERON-BASE2 supports fifo non-blocking mode. This means that the 'sender' of data will continue to send data even if when the 'receiver' is not able to read the data (fast enough). To select non-blocking mode, use keyword FIFONOBLOCK or FIFONOB. Example:

```
FIFONOBLOCK heron1 1 in
```

Alternatively, you can use the optional keyword 'NOBLOCK' after a BOOTLINK / BOOTPATH statement. Example:

```
BOOTLINK HERON1  1  HERON2  0  noblock
```

In this statement the fifo from the slot 1 (HERON1) to slot 2 (HERON2) is configured to non-blocking mode, assuming HERON1 and HERON2 definitions as used in previous sections, as well as the fifo from slot 2 to slot 1.

FIFONOBLOCK can also be used with the host module fifos. For example:

```
pcif    0  host   NORMAL       00000005
FIFONOB    host   0  in    0
```

There is some degree of redundancy here, as, for example, host fifo 0 out is the same as slot 1 fifo 0 in (please refer to the HERON-BASE2 manual for fifo connections).

Sample Network Description file for HERON-BASE2 (1)

For a HERON-BASE2 with 2 DSP nodes, the network description file could look like:

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        heb2a           0               0

#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  c6   0  HERON2 NORMAL (1)  00000002 heron2.out
#--------------------------------------
# Bootpath description.
# BOOTLINK  ND_NAME  PORT  ND_NAME  PORT
#--------------------------------------
  BOOTLINK  HERON1   1     HERON2   0

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0
```

This network file describes a HERON-BASE2 with a board switch set to 0, to be accessed

via fifo A. There are two C6x nodes on this board (0). They are named "HERON1" and "HERON2". The heron-id of the first one is 1 (and as this is the first slot is must also be the ROOT node, as only this slot has access to the host PC), the second one is 2, the module being in slot 2 of the HERON-BASE2.

Both HERON modules are connected by a fixed HERON-BASE2 fifo link. This is described in the BOOTLINK statement. The HERON-BASE2 has fixed fifo connections, and if you review the HERON-BASE2 manual you can find out how the HERON-BASE2 slots are connected to each other.

Sample Network Description file for HERON-BASE2 (2)

The HERON-BASE2 can access both slots, via different fifo's. It is possible for the Server / Loader to serve both nodes, as follows.

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        heb2a           0               0
  BD API        heb2a           0               1

#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  c6   1  HERON2 ROOT   (1)  00000002 heron2.out

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0
```

You may still declare the

```
  BOOTLINK  HERON1   1     HERON2   0
```

as in the previous example, but it won't be used. Each node will be booted via 'it's own' fifo connection with the host. Declaring a BOOTLINK or BOOTPATH may still be useful if you want to associate the fifo's involved with a UMI reset, or configure the fifo's to non-blocking mode.

Sample Network Description file for HERON-BASE2 (3)

If you use only a DSP module in slot2, note that you should use a BD API statement with the Device ID set to 1 (and not 0), as the DSP program is to be loaded via FifoB (1).

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        heb2a           0               1

#-------------------------------------------------------
# Nodes description
```

```
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   1  HERON2 ROOT   (0)  00000002 heron2.out
```

The Network Description File is an ASCII file that lists all carrier boards, modules and their inter-connections. (Instead of 'modules' it is better to talk about 'nodes'; some modules may have multiple processors or FPGA's.) The following information must be present:

- A complete list of carrier boards (HEPC8)
- A complete list of the nodes (C6x) and programs to be loaded onto them.
- A complete list of the FPGA or HERONIO modules and their bit-streams.
- A complete list of boot link connections.
- Route to host.

Carrier Board Declaration

An example entry for declaring an HEPC8 is as follows:

```
BD API hep8a 0 0
```

The first item, BD, tells the Server/Loader that on this line a board is declared. The second item, API, tells the Server/Loader that the board is to be accessed via the API. (Obviously, the API must have been installed correctly for this to work, eventually.) The remainder is API related: API board name ("hep8a"), board number, and device number, respectively.

Please note that the board number is the number selected by the switch on the HEPC8. It has possible values ranging from 0 to 15. If this switch is set to, for example, 4, then your BD declaration becomes:

```
BD API hep8a 4 0
```

C6x Processor and Program Declaration

For example, if there are two HERON modules inserted on the HEPC8 above, then the processors and programs to boot onto them are defined as follows:

```
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

The first item tells the Server/Loader that a 'C6x processor is declared. The second item tells the Server/Loader via what board this processor is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

The third item is the name of the processor. You can choose any name you like. The third item tells the Server/Loader what type of node it is: ROOT node or NORMAL node. A ROOT node has a direct connection to the PC where the Server/Loader is running and a NORMAL node is a processor at least 1 hop away from the Server/Loader PC. There must be exactly one ROOT node in a network description file.

The fourth item is the Code Composer Studio ID. Code Composer Studio labels processors as they appear along the "JTAG scan path". This is not necessarily the same as the labels we put on them in the network description file. When you use the `-g` option, the Code Composer Studio ID is used to map program file onto node.

The fifth item is the HERON module's ID. On a HEPC8, the first slot will have ID 1, the second will have ID 2, the third ID3, and the fourth ID 4. But note that the HERON ID is made up of the HEPC8's board number (bits 7..4) and the slot number (bits 3..0). So if the HEPC8 board number switch is set to 4, the HERON IDs become 0x41, 0x42, 0x43 and 0x44. Thus with a HEPC8 board switch set at 0, we have:

```
BD API hep8a 0 0
c6 0 HERON1 ROOT   (1) 00000001 heron1.out
c6 0 HERON2 NORMAL (0) 00000002 heron2.out
```

but when the HEPC8 board switch is set to 4, we will have:

```
BD API hep8a 4 0
c6 0 HERON1 ROOT   (1) 0x41 heron1.out
c6 0 HERON2 NORMAL (0) 0x42 heron2.out
```

The last entry is the program name. This should be an executable file produced by the Texas Instruments' C compiler for the 'C6x. With the HEPC8, 2 program files may be declared. In case 2 are declared, the first is executed first, the second program second. You cannot just use any file as the first file. The Server/Loader expects an executable program that sends a specific amount of data back to the Server/Loader, and which then "resets" the processor. Two such programs are present in `\hesl\etc\c6x\eeprom`. They are `eeprom62.out` and `eeprom67.out`. The processor entries would then become:

```
c6 0 HERON1 ROOT   (1) 00000001 eeprom62.out heron1.out
c6 0 HERON2 NORMAL (0) 00000002 eeprom67.out heron2.out
```

assuming that the second processor is a 'C67xx processor, and the first a 'C62xx. You would only use eeprom62/67.out if you use the `-c` option of the Server/ Loader.

Please note that for the ***HeartConf*** tool, the processor is not actually loaded. However, the parser still needs a program name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the program name, for example “a” or “no-file”. But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper filename. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***.. A file fit for use with ***HeartConf*** may need to have proper program names specified before it works with the ***Server/Loader***.

BootLink and BootPath Declaration

The Server/Loader needs to know via what FIFOs it should boot NORMAL nodes. The following line shows you how to declare a connection between processors heron1 and heron2 (as declared in a previous section).

```
BOOTLINK HERON1 2 HERON2 0
```

The first entry, BOOTLINK, tells the Server/Loader that this line declares a connection between one processor and an adjacent processor. In this example, processor HERON1's FIFO 2 is connected to processor HERON2's FIFO 0.

BOOTLINK declares a two-way connection: from HERON1 to HERON2, and from HERON2 to HERON1. Processor HERON 1 can send data to processor HERON2 via FIFO 2, and can also read data from processor HERON2 from FIFO 2. Processor HERON2 can send data to HERON1 via FIFO 0, and can also read data from HERON1 from FIFO 0.

However, this may not always be the case. In case of one-way connections, you can use:

```
BOOTPATH HERON1 2 HERON2 0
```

This declares only a connection from HERON1 to HERON2. You would need two BOOTPATH declarations to emulate one BOOTLINK declaration. For example:

```
BOOTPATH HERON1 2 HERON2 0
BOOTPATH HERON1 0 HERON2 2
```
is equal to:
```
BOOTLINK HERON1 2 HERON2 0
```

Host Path Declaration

The connection between the ROOT node and the host PC that runs the Server/Loader is declared as follows:

```
HOSTLINK          0
```

The first entry (HOSTLINK) tells the Server/Loader that this line declares what FIFO or link of the ROOT node is connected to the host PC. This declaration defines a two-way link. It tells the Server/Loader that the ROOT node can both read from the host PC and write to the host PC via FIFO or link number 0.

In some cases there may not be a two-way connection, but only a one-way connection. In that case you can use:

```
FROMHOST          0
```

when the ROOT node can read data from the host PC over FIFO or link 0. And:

```
TOHOST            0
```

when the ROOT node can write data to the host PC over FIFO or link 0.

FPGA / HERONIO Declaration

For example, if there is an FPGA module inserted on the HEPC8 above, then the module and program to boot onto them are defined as follows:

```
fpga 0 FPGA1 NORMAL   0x3 mybitstream.rbt
```

The first item tells the Server/Loader that a FPGA module is declared. The second item tells the Server/Loader via what board this module is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is 0; the second BD declaration is 1, and so on.

The third item is the name of the module. You can choose any name you like. The fourth item tells the Server/Loader what type of node it is: always type NORMAL node.

The fifth item is the FPGA module's HERON ID. On a HEPC8, the first slot will have ID 1; the second will have ID 2, the third ID3, and the fourth ID 4. But note that the HERON ID is made up of the HEPC8's board number (bits 7..4) and the slot number (bits 3..0). So if the HEPC8 board number switch is set to 4, the HERON IDs become 0x41, 0x42, 0x43 and 0x44. Thus with a HEPC8 set at 0, we have:

```
BD API hep8a 0 0
fpga 0 FPGA1 NORMAL   0x03 mybitstream.rbt
```

but when the HEPC8 is set to 4, we will have:

```
BD API hep8a 4 0
fpga 0 FPGA1 NORMAL   0x43 mybitstream.rbt
```

The last entry is a bit-stream file name. This should be an rbt file produced by Xilinx tools.

Please note that for the ***HeartConf*** tool, the FPGA is not actually programmed. However, the parser still needs a bit-stream file name there. If you use a network file exclusively for use with ***HeartConf***, you may write anything in place of the bit-stream file name, like "a" or "no-rbt". But if you also use the network file for use with the ***Server/Loader***, then you need to specify a proper bit-stream. A network file fit for use with the ***Server/Loader*** will certainly also work with ***HeartConf***.. A network file fit for use with ***HeartConf*** may need to have proper bit-stream file names specified before it works with the ***Server/Loader***.

Sample Network Description file for HEPC8

For a HEPC8 with 2 nodes, 1 FPGA module, the network description file could look like:

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        hep8a           0               0

#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  c6   0  HERON2 NORMAL (1)  00000002 heron2.out
  fpga 0  FPGA1  NORMAL      00000003 mybitstream.rbt

#--------------------------------------
# Bootpath description.
# BOOTLINK  ND_NAME  PORT  ND_NAME  PORT
#--------------------------------------
  BOOTLINK  HERON1   2     HERON2   0

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0
```

This network file describes a HEPC8 with a board switch set to 0, to be accessed via fifo A. There are two C6x nodes on this board (0). They are named "HERON1" and "HERON2". The heron-id of the first one is 1 (and as this is the first slot is must also be the ROOT node, as only this slot has access to the host PC), the second one is 2, the module being in slot 2 of the HEPC8. There's also an FPGA module in slot 3.

Both HERON modules are connected by a fixed HEPC8 fifo link. This is described in the BOOTLINK statement. The HEPC8 has fixed fifo connections, and if you review the HEPC8 manual you can find out how the HEPC8 slots are connected to each other.

Finally, we specify that the ROOT node is connected to the host via fifo A (0), by means of the HOSTLINK statement. Note that when using HERON processor modules on the HEPC8, routing jumpers are used to make the module boot from a certain fifo. The selected fifo then becomes fifo 0.

The Network Description File is an ASCII file that lists all carrier boards, modules, and their inter-connections. Instead of 'modules' it is better to talk about 'nodes'; for example an HETWIN module has two C4x processors, i.e. two 'nodes'. The following information must be present:

- A complete list of carrier boards (HEPC2E, HEPC3, HEPC4, etc.).
- A complete list of the nodes (C4x processors) and programs to be loaded onto them.
- A complete list of boot link connections.
- Route to host.

Carrier Board Declaration

An example entry for declaring an HEPC3 is as follows:

```
BD API hep3b 0 0
```

The first item, BD, tells the Server/Loader that on this line a board is declared. The second item, API, tells the Server/Loader that the board is to be accessed via the API. (Obviously, the API must have been installed correctly for this to work, eventually.) The remainder is API related: API board name ("hep3b"), board number, and device number, respectively.

If you have just 1 HEPC3 in your system this will work. If you have 2 HEPC3's in your system, you need to define one HEPC3 as "`BD API hep3b 0 0`" and the other as "`BD API hep3b 1 0`". Which of the two is "0" and which of the two is "1" depends entirely on which comes first seen from your PC's PCI bus. If you need this information, do some testing with the confidence checks (e.g. see what led's light up when you do a reads confidence check with board 0).

The board number has possible values ranging from 0 to 3 (0 to 2 for the HEPC2E).

C4x Processor and Program Declaration

For example, if there are two one processor TIM-40 modules inserted on a HEPC4 board, then the processors and programs to boot onto them are defined as follows:

```
BD API HEP3B  0  0
# Node descriptions.
# ND BD name  type  CCID GBCW     LBCW     IACK     filename(s)
  ND 0  NODE0 ROOT   (1) 00000000 00000000 002ff800 idrom.out sample1.out
  ND 0  NODE1 NORMAL (0) 00000000 00000000 002ff800 idrom.out sample2.out
```

(N.B. The HEPC4 and HCPCI1 boards have an identical PCI interface with the HEPC3, and typically 'hep3b' is used for BD API declarations even if the actual board is HEPC4 or HCPCI1).

The BD line declares a HEPC4 board to be accessed via the API. In the ND lines, the first item (ND itself) tells the Server/Loader that a 'C4x processor is declared. Instead of ND, you may also use C4.

The second item tells the Server/Loader via what board this processor is accessed. The number is the number in the list of BD declarations you made. The first BD declaration is

0; the second BD declaration is 1, and so on.

The third item is the name of the processor. You can choose any name you like.

The third item tells the Server/Loader what type of node it is: ROOT node or NORMAL node. A ROOT node has a direct connection to the PC where the Server/Loader is running and a NORMAL node is a processor at least 1 hop away from the Server/Loader PC. There must be exactly one ROOT node in a network description file.

The fourth item is the Code Composer Studio ID. This is no longer used for C4x nodes, and you can ignore this. It makes no difference whether the Code Composer Studio IDs are there or not, for C4x nodes.

The fifth item is the GBCW (Global Bus Control Word). If the module in question is a TIM-40, the Server/Loader will retrieve the GBCW from the TIM-40's IDROM - if you specify 0 for the GBCW. If you use a value different than 0 for GBCW, the Server/Loader will use that value rather than the IDROM supplied one.

The sixth item is the LBCW (Local Bus Control Word). If the module in question is a TIM-40, the Server/Loader will retrieve the LBCW from the TIM-40's IDROM - if you specify 0 for the LBCW. If you use a value different than 0 for LBCW, the Server/Loader will use that value rather than the IDROM supplied one.

The seventh item is the IACK. This should always be set to 2ff800.

The last entries are program names. These should be executable files produced by the Texas Instruments' C compiler for the 'C4x. There should be 2 program files declared; you can do with only 1 if your program does the work that the `idrom.out` program usually does (sending the idrom's contents to the host). You cannot just use any file as the first file. The Server/ Loader expects an executable program that sends a specific amount of data back to the host (Server/Loader), and which then "resets" the processor. These programs are present in `\hesl\etc\c4x\idrom`. They are `idrom.out`, `hequad40.out`, `hequad50.out` and `hequad60.out`. For TIM-40 nodes, please use `idrom.out`. For non TIM-40 nodes (HEQUAD) use one of the `hequadxx.out` files (xx = the HEQUAD's processor speed, 40/50/60 Mhz).

BootLink and BootPath Declaration

For all C4x carrier boards such as HEPC3 and HEPC2, the Server/Loader needs to know via what comports it should boot NORMAL nodes. The following line shows you how to declare a connection between processors ‘node0’ and ‘node1’ (as declared in a previous section).

```
BOOTLINK NODE0 2 NODE1 0
```

The first entry, BOOTLINK, tells the Server/Loader that this line declares a connection between one processor and an adjacent processor. In this example, processor NODE0's comport 2 is connected to processor NODE1's comport 0.

BOOTLINK declares a two-way connection: from NODE0 to NODE1, and from NODE1 to NODE0. Processor NODE0 can send data to processor NODE1 via comport 2, and can also read data from processor NODE1 from comport 2. Processor NODE1 can send data to NODE0 via comport 0, and can also read data from NODE0 from comport 0.

However, this may not always be the case. In case of one-way connections, you can use:

```
BOOTPATH NODE0 2 NODE1 0
```

This declares only a connection from NODE0 to NODE1. You would need two BOOTPATH declarations to emulate one BOOTLINK declaration. For example:

```
BOOTPATH NODE1 2 NODE0 0
BOOTPATH NODE0 0 NODE1 2
```
is equal to:
```
BOOTLINK NODE0 2 NODE1 0
```

Host Path Declaration

The connection between the ROOT node and the host PC that runs the Server/Loader is declared as follows:

```
HOSTLINK          0
```

The first entry (HOSTLINK) tells the Server/Loader that this line declares what comport of the ROOT node is connected to the host PC. This declaration defines a two-way link. It tells the Server/Loader that the ROOT node can both read from the host PC and write to the host PC via comport number 0.

In some cases there may not be a two-way connection, but only a one-way connection. In that case you can use:

```
FROMHOST          0
```

when the ROOT node can read data from the host PC over comport 0. And:

```
TOHOST            0
```

when the ROOT node can write data to the host PC over comport 0.

Sample Network Description file for HEPC3

For a HEPC3 with 2 nodes, the network description file could look like:

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        hep3b           0               0

#-------------------------------------------------------
# Nodes description
# ND BD name  type   GBCW     LBCW      IACK     filename(s)
#-------------------------------------------------------
ND 0 NODE0 ROOT   00000000 00000000 002ff800 idrom.out root.out
ND 0 NODE1 NORMAL 00000000 00000000 002ff800 idrom.out slave.out

#-------------------------------------
# Bootpath description.
# BOOTLINK ND_NAME PORT   ND_NAME PORT
#-------------------------------------
BOOTLINK   NODE0   5      NODE1   2

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK PORT
#-----------------------------------------------
HOSTLINK   3
```

This network file describes a HEPC3, board number0, to be accessed via comport A. There

are two C4x nodes on this board. They are named “NODE0” and “NODE1”. There 1 ROOT node (i.e. this node is connected to the host via a comport connection), the other node is NORMAL (i.e. this node is not connected to the host, and is reached only via other nodes from the host). The global bus control word (GBCW) and local bus control word (LBCW) are extracted from the IDROM using idrom.out (value 0 indicates that).

Both C4x modules are connected by a comport connection. This is described in the BOOTLINK statement. The HEPC2E, HEPC3, HEPC4 and HCPCI1 connect comports in certain ways, depending on module type and size. Please review the manuals of these boards for more information how modules are or maybe inter connected via comports.

Finally, we specify that the ROOT node is connected to the host via comport 3, by means of the HOSTLINK statement.

Network Description File Syntax

The following section covers the syntax for commands in the network description file. This file is passed to the Server/Loader and is parsed to create a representation of the network. During the parser stage, an unrecognised command will cause the loader to abort, reporting the command at the point it failed. Finally, an example network description file is shown, illustrating how to represent a 2-node network.

Describing Boards

The BD command informs the loader that a motherboard or host adapter description will follow. Each occurrence of a board description has a reference number assigned to it. This reference number is required by the ND node description command. The numbering is sequential, starting from 0 for the first BD entry.

There are two forms of the BD command - one for the HUNT ENGINEERING API and one for direct I/O. Since C6x boards have no support for direct I/O, you must use the HUNT ENGINEERING API.

BD syntax

```
BD    [type] [Board_type] [Board_Id] [Device_Id] <options>
Options: REMOTE
```

`[type]` Field indicating the motherboard type. For the HUNT ENGINEERING API all boards are of type "API".

`[Board_type]` Field indicating the API code for the motherboard type:

"hep2d"	For the HEPC2M rev D
"hep2e"	For the HEPC2E
"hep3b"	For the HEPC3 rev B, HEPC4, and HECPCI1
"hep6a"	For the HEPC6 rev A
"hep8a"	For the HEPC8 rev B
"hep9a"	For the HEPC9 rev B

See the API manual for a list of supported boards on your platform.

`[Board_Id]` Field indicating which board in the system is to be used. In the case of a HEPC9 or HEPC8 this number is the value of red switch on the carrier board. For a full description see the API documentation. For legacy boards, such as the HEPC3 or HEPC2E sometimes the Board Id is a sequence number (1st board is 0, 2nd is 1, ...) and sometimes it corresponds to the address of the board (e.g. 0 corresponds to 0x150 for the HEPC2M default address and 0x200 for the HEPC2E default address). For a full description of the use of this field see the API documentation.

`[Device_Id]` Field indicating the device, in case the board supports multiple devices.

This is a numeric code:
0 denotes FIFO A 6 denotes FIFO C 8 denotes FIFO E
1 denotes FIFO B 7 denotes FIFO D 9 denotes FIFO F

`REMOTE` Optional. This declares that a board should be accessed via other boards. Typically used for boards outside a PC case. But it can also be used if the board defined as REMOTE is actually still within the same PC. Its PCI interface is still accessible, but all HSB and RESET will go via another board connected to the REMOTE board. The REMOTE board must be accessible via Inter-Board Connector module links.

For the HEPC3, HEPC2E and HEPC2M please note that boards that are purely slaves of other boards, i.e. their reset is connected to a master board not their own host interface, need not be declared here. The HEPC8 cannot have slave boards, so defining more than 1 HEPC8 will denote separate systems. With HEPC9 systems all boards in the system should be defined.

BDCONN / BDLINK / BDPATH syntax (HEART boards only)

Boards (e.g. two HEPC9's) that are connected by means of two inter-board connectors (e.g. EM2) can define the connection with a BDCONN statement. This defines a duplex connection between the two inter-board modules (i.e. the boards).

```
BDCONN [fromnode] [fromchan] [tonode] [tochan] <options>

Options: NOHSB, NORESET, ONEWAY
```

`[fromnode]` Name of an Inter-Board Connector module, such as an EM2 or EM1 module. The module must have been defined earlier in the network file.

`[fromchan]` Integer value indicating the channel that is used by the Inter-Board Connector module 'fromnode' connect to the other board. For an EM1 this is always 0, for an EM2 you have a choice between 0..5.

`[tonode]` Name of an Inter-Board Connector module, such as an EM2 or EM1 module. The module must have been defined earlier in the network file.

`[tochan]` Integer value indicating the channel that is used by the Inter-Board Connector module 'tonode' to connect to the other board. For an EM1 this is always 0, for an EM2 you have a choice between 0..5.

`NOHSB` Optional. In HEART systems with more than one board, and whose boards are connected via Inter-Board Connector modules, and at least one board is a 'remote' board, HSB and RESET connections are propagated over Inter-Board Connector channels. This so that the remote board can be accessed by HSB and so that it can be reset (via another board). However, if there are multiple channel paths via which a 'remote' board can be reached, the keyword NOHSB must be used to define links that must not propagate HSB until just one link is left propagating HSB to the 'remote' board.

`NORESET` Optional. In HEART systems with more than one board, and whose boards are connected via Inter-Board Connector modules, and at least one board is a 'remote' board, HSB and RESET connections are propagated over Inter-Board Connector channels. This so that the remote board can be accessed by HSB and so that it can be reset (via

another board). However, if there are multiple channel paths via which a 'remote' board can be reached, the keyword NORESET must be used to define links that must not propagate RESET until just one link is left propagating RESET to the 'remote' board.

`ONEWAY` Optional. Declares that this link between Inter-Board Modules is simplex (one way) only.

As an alternative, you can also declare links between boards directly. However, it assumes that Inter-Board Connector modules are defined later in the network file. And the actual connection is still assumed to be a link between to Inter-Board Connector modules.

To declare links between boards directly, use a BDLINK or BDPATH statement. BDLINK and BDPATH define a board-to-board connection, assuming that inter-board modules are define later in the network file. BDLINK defines a one-way connection, BDPATH a duplex connection between the two carrier boards.

```
BDLINK [frombd] [fromfifo] [tobd] [tofifo] <options>
BDPATH [frombd] [fromfifo] [tobd] [tofifo] <options>

Options: NOHSB, NORESET
```

`[frombd]` Integer value indicating from what board the connection starts. The integer value is one of the boards defined with a BD statement earlier. The first BD (API) statement defines board #0, and so on.

`[fromfifo]` Integer value indicating the fifo number that is (to be) used by the inter-board module on 'frombd' to connect to the other board.

`[tobd]` Integer value indicating at what board the connection ends. The integer value is one of the boards defined with a BD statement earlier. The first BD statement defines board #0, and so on.

`[tofifo]` Integer value indicating the fifo number that is (to be) used by the inter-board module on 'tobd' to connect to the other board.

`NOHSB` Optional. In HEART systems with more than one board, and whose boards are connected via Inter-Board Connector modules, and at least one board is a 'remote' board, HSB and RESET connections are propagated over Inter-Board Connector channels. This so that the remote board can be accessed by HSB and so that it can be reset (via another board). However, if there are multiple channel paths via which a 'remote' board can be reached, the keyword NOHSB must be used to define links that must not propagate HSB until just one link is left propagating HSB to the 'remote' board.

`NORESET` Optional. In HEART systems with more than one board, and whose boards are connected via Inter-Board Connector modules, and at least one board is a 'remote' board, HSB and RESET connections are propagated over Inter-Board Connector channels. This so that the remote board can be accessed by HSB and so that it can be reset (via another board). However, if there are multiple channel paths via which a 'remote' board can be reached, the keyword NORESET must be used to define links that must not propagate RESET until just one link is left propagating RESET to the 'remote' board.

C6 nodes syntax (HEPC8, HEART, and HERON-BASE2 boards only)

The node declaration informs the loader that a node description will follow. A 'C6x processor node can be declared with:

`C6 [host_bd] [nd_name] [nd_type] [(cc-id)] [heron_id] [filenames..]`

Instead of "C6" you may also use "c62", "c62x", "c6201", "c67", "c67x", and "c6701". In the current version of the Server/Loader there are no differences between these entries.

`[host_bd]`	Field indicating the number of the motherboard controlling this processor. (note that if a module is on a slave board, this field should contain the reference to its master.)
`[nd_name]`	Character string uniquely identifying the processor.
`[nd_type]`	Field indicating whether this processor is the root node or a slave node. The `nd_type` must be one of the following two values. "`ROOT`" indicating that this is the root node. "`NORMAL`" indicating that this is a regular network node. The value ROOT can only be applied to the first node, or `root' node in the network.
`[(cc-id)]`	Optional. Denotes a Code Composer Studio processor ID. Code Composer Studio labels processors as it finds them on the JTAG scan path. This is not necessarily the same as how we see the system as expressed in the network file.
`[heron-id]`	The ID of the HERON module. This ID has two parts: bits 7..4 that denote the board number (hex 0 to f) and bits 3..0 that denote the slot number (1, 2, 3 or 4 on a HEPC8). The board number is set by the red switch on the HEPC8. (This is not necessarily 0!)
`[Filenames]`	The names of the file(s) to load onto this node.[1]

C4 nodes syntax (C4x carrier boards only)

The ND command informs the loader that a node description will follow.

`ND [host_bd] [nd_name] [nd_type] [GBCW] [LBCW] [IACK] [filenames..]`

Instead of "ND" you may also use "c4", "c4x", "c40", or "c44". In the current version of the Server/Loader there are no differences between these entries.

[1] If two filenames are provided the first is taken as the node-specific initialisation routine, and the second is taken as the target program for the node. For HEART boards, such as HEPC9, only 1 filename may be specified.

ND	Keyword indicating the description of a 'C4x DSP node.
[host_bd]	Field indicating the number of the motherboard controlling this processor. (note that if a module is on a slave board, this field should contain the reference to its master.)
[nd_name]	Character string uniquely identifying the processor.
[nd_type]	Field indicating whether this processor is the root node or a slave node. The nd_type must be one of the following two values. ROOT Keyword indicating that this is the root node. NORMAL Keyword indicating that this is a regular network node. The value ROOT can only be applied to the first node, or `root' node in the network.
[GBCW]	Global Bus Control Word for this DSP node.[2]
[LBCW]	Local bus control word for this DSP node.[3]
[IACK]	Iack address for this DSP node.[4]
[Filenames]	The names of the file(s) to load onto this C4x.[5]

FPGA nodes syntax (HEPC8, HEART, and HERON-BASE2 boards only)

The FPGA declaration informs the loader that a FPGA or HERONIO description will follow. An FPGA node can be declared with:

```
FPGA [host_bd] [nd_name] [nd_type] [heron_id] [filename]
```

Instead of "FPGA" you may also use "HERONIO". In the current version of the Server/Loader there are no differences between these entries. It denotes any HERON module that has a programmable FPGA on board that can be downloaded via HSB.

[host_bd]	Field indicating the number of the motherboard controlling this processor. (Note that if a module is on a slave board, this field should contain the reference to its master, ie the carrier board via which PCI interface you wish to download the FPGA's bootstream.)
[nd_name]	Character string uniquely identifying the module.
[nd_type]	Field indicating whether this module is the root node or a slave node. The nd_type must be one of the following two values. "ROOT" indicating that this is the root node. "NORMAL" indicating that this is a regular network node. The value ROOT can only be applied to the first node, or `root' node in

[2] These values are ignored if two filenames are provided

[3] These values are ignored if two filenames are provided

[4] Note that the IACK value is required in both cases

[5] If two filenames are provided the first is taken as the node-specific initialisation routine, and the second is taken as the target program for the node.

the network.

`[heron-id]` The ID of the HERON module. This ID has two parts: bits 7..4 that denote the board number (hex 0 to f) and bits 3..0 that denote the slot number (1, 2, 3 or 4 on a HEPC8 or 9). The board number is set by the red switch on the HEPC8 or 9. (This is not necessarily 0!)

`[Filename]` The names of the bit stream to load onto this node.

GDIO nodes syntax (HEART boards only)

The GDIO statement defines a HEGDx module. A GDIO module can be declared with:

```
GDIO [host_bd] [nd_name] [nd_type] [heron_id]
```

You can use this description to describe any GDIO module, such as the HEGD1, HEGD2, HEGD3 and so on. The reason why you might want to describe a GDIO module is to give you a 'named handle' that you can use later, in a HEART statement, to define a HEART FIFO connection between a GDIO module and another module. For **non**-HEART boards there's no reason to use this statement: you can but the Server/ Loader won't use the information for anything so you may just as well omit it.

`[host_bd]` Field indicating the number of the motherboard controlling this processor. (note that if a module is on a slave board, this field should contain the reference to its master.)

`[nd_name]` Character string uniquely identifying the processor.

`[nd_type]` Field indicating whether this processor is the root node or a slave node. The `nd_type` must be one of the following two values.

"`ROOT`" indicating that this is the root node.
"`NORMAL`" indicating that this is a regular network node.

The value ROOT can only be applied to the first node, or `root' node in the network.

`[heron-id]` The ID of the HERON module. This ID has two parts: bits 7..4 that denote the board number (hex 0 to f) and bits 3..0 that denote the slot number (1, 2, 3 or 4 on a HEPC8). The board number is set by the red switch on the HEPC8. (This is not necessarily 0!)

PCIF (Host Interface) nodes syntax (HEART and HERON-BASE2 boards only)

The PCIF declaration informs the loader that a host interface description will follow. A host interface can be declared with:

```
PCIF [host_bd] [nd_name] [nd_type] [heron_id]
```

You can use this description to describe a host interface, such as the PCI interface on a HEPC9 board. The reason why you might want to describe a PCIF module is to give you a 'named handle' that you can use later, in a HEART statement, to define a HEART FIFO connection between the PCI interface and a module. For **non**-HEART boards there's no reason to use this statement: you can, but the Server/ Loader won't use the information for anything so you may just as well omit it.

[host_bd] Field indicating the number of the motherboard controlling this processor. (note that if a module is on a slave board, this field should contain the reference to its master.)

[nd_name] Character string uniquely identifying the processor.

[nd_type] Field indicating whether this processor is the root node or a slave node. The nd_type must be one of the following two values.

"ROOT" indicating that this is the root node.
"NORMAL" indicating that this is a regular network node.

The value ROOT can only be applied to the first node, or `root' node in the network.

[heron-id] The ID of the HERON module. This ID has two parts: bits 7..4 that denote the board number (hex 0 to f) and bits 3..0 that denote the hsb number (**5** on a HEPC9). The board number is defined by the red switch on the HEPC9. (This is not necessarily 0!)

EM2/EM1/EM1C nodes syntax (HEART boards only)

The EM2/EM1/EM1C declaration informs the loader that an Inter-Board Connector module description will follow.

```
EM2   [host_bd] [nd_name] [nd_type] [heron_id]
EM1   [host_bd] [nd_name] [nd_type] [heron_id]
EM1C  [host_bd] [nd_name] [nd_type] [heron_id]
```

The reason why you might want to describe a Inter-Board Connector module is to give you a 'named handle' that you can use in a BDCONN statement to declare connections (links) between 2 Inter-Board Connector modules. Or it can be used in a HEART statement, to define a HEART FIFO connection between an Inter-Board Connector module and another module. For **non**-HEART boards there's no reason to use this statement: you can but the Server/ Loader won't use the information for anything so you may just as well omit it.

[host_bd] Field indicating the number of the motherboard controlling this module. (note that if a module is on a slave board, this field should contain the reference to its master.)

[nd_name] Character string uniquely identifying the processor.

[nd_type] Field indicating whether this processor is the root node or a slave node. The nd_type must be one of the following two values.

"ROOT" indicating that this is the root node.
"NORMAL" indicating that this is a regular network node.

The value ROOT can only be applied to the first node, or `root' node in the network.

[heron-id] The ID of the HERON module. This ID has two parts: bits 7..4 that denote the board number (hex 0 to f) and bits 3..0 that denote the hsb number (**6** on a HEPC9). The board number is set by the red switch on the HEPC9. (This is not necessarily 0!)

BOOTLINK / BOOTPATH syntax (C4x boards, HEPC8, and HERON-BASE2

only)

The BOOTLINK declaration informs the loader that a boot path description will follow. For every node in the network other than the root node, one `BOOTLINK` entry is required. In the case of the root node, the `HOSTLINK` command must be used. For HEART based boards, such as the HEPC9, no BOOTLINK or BOOTPATH statements are used though. This is because the Server/Loader can automatically create temporarily HEART FIFO links over which to boot processors or FPGA modules.

```
BOOTLINK [parent_nd] [parent_lnk] [target_nd] [target_lnk]
```

Instead of BOOTLINK you may also use the keyword BOOT.

`[parent_nd]`	The name of the processor node from which to boot the target node.
`[parent_lnk]`	Parent's node fifo through which the parent will boot the target node.
`[target_node]`	The name of the target node.
`[target_lnk]`	Target node's fifo through which the target node will be booted.

The BOOTPATH declaration informs the loader that a boot path description will follow. It is different from the `BOOTLINK` command in that the `BOOTPATH` defines a one-way path from the "parent" to the "target". The `BOOTLINK` declaration defines a two-way path: from "parent" to "target", and from "target" to "parent".

```
BOOTPATH [parent_nd] [parent_lnk] [target_nd] [target_lnk]
```

Instead of BOOTPATH you may also use the keyword PATH.

`[parent_nd]`	The name of the processor node from which to boot the target node.
`[parent_lnk]`	Parent node's FIFO number through which the parent will boot the target node.
`[target_node]`	The name of the target node.
`[target_lnk]`	Target node's FIFO or comport through which the target node will be booted.

HEART syntax (HEART boards only)

A HEART statement tells the Server/Loader to create a point-to-point FIFO connection between two modules. It thus doesn't describe a situation, is tells the Server/Loader to perform an action. The connection created is one way.

```
HEART [fromnode] [fromfifo] [tonode] [tofifo] [timeslots] <options>
Options: NOSERVE, UMI 0|1|2|3
```

`[fromnode]`	The name of the node where the FIFO connection starts. This node must have been defined earlier with a C6, FPGA/HERONIO, GDIO, PCIF or IBC definition.
`[fromfifo]`	The number of the fifo (0..5) on fromnode from where the FIFO connection starts.
`[tonode]`	The name of the node where the FIFO connection ends. This node must have been defined earlier with a C6, FPGA/HERONIO, GDIO,

PCIF or IBC definition.

`[tofifo]` The number of the fifo (0..5) on tonode from where the FIFO connection ends.

`[timeslots]` The number of timeslots that must be used for this connection (1..6). It is also possible to specify precisely what timeslots to use: "t=0,3" tells the Server/Loader to use timeslots 0 and 3 (of 0..5). You can also use "v=0x81" to do the same. The "v=" tells the Server/Loader that you wish to use a 6-bit mask to specify what timeslots should be used. If no "t=" or "v=" specifier is used, the number specified is interpreted as the number of timeslots you wish to use. The Server/ Loader will allocate automatically proper timeslots. This is the usual way of using HEART statements.

`NOSERVE` The Server/Loader will check the network file to find all processor nodes that are connected to a host interface. This connection may be a direct HEART connection between a processor node and a host interface, or a connection via Inter-Board Connectors. The Server will then serve all processor nodes that have a duplex connection to the host. However, frequently you want a processor – host connection for yourself and not have the Server use it. For such situations you can use the keyword NOSERVE to indicate that the Server should ignore this connection and not serve it.

`UMI 0|1|2|3` HEART boards have a reset FIFO feature. A HEART FIFO can be reset via UMI lines. For this to work, you have to specify what UMI line a FIFO is associated with. In the case that you want to be able to reset all HEART FIFO's related to a HEART connection, use the UMI keyword, and then specify the UMI line. You can specify more than 1 UMI line, separated by commas. The actual reset is then done by twiggling a UMI line. You can also associate single FIFO's with a UMI line by using a UMIRESET statement.

BDCAST syntax (HEART boards only)

A BDCAST statement tells the Server/Loader to create a one way FIFO connection from a module onto the HEART ring. The statement defines a module that 'broadcasts' data onto the timeslot you specify in this statement. Modules can 'listen' to a 'broadcast' by reading from the ring at that timeslot. A 'listen' connection can be created with the LISTEN statement. It thus doesn't describe a situation, is tells the Server/Loader to perform an action.

```
BDCAST [broadcast][node][outfifo][timeslots] <options>
Options: UMI 0|1|2|3
```

`[broadcast]` Give the broadcast a name. This allows you to later 'name' the broadcast to listen to in a LISTEN statement.

`[node]` The name of the node that will broadcast. This node must have been defined earlier with a C6, FPGA/HERONIO, GDIO, PCIF or IBC definition.

`[outfifo]` The number of the fifo (0..5) on node to broadcast with.

`[timeslots]` The number of timeslots that must be used for this connection (1..6). It is also possible to specify precisely what timeslots to use: "t=0,3" tells the Server/Loader to use timeslots 0 and 3 (of 0..5). You can also use "v=0x81" to do the same. The "v=" tells the Server/Loader that you wish to use a 6-bit mask to specify what timeslots should be used. If no "t=" or "v=" specifier is used, the number specified is interpreted as the number of timeslots you wish to use. The Server/ Loader will allocate automatically proper timeslots. This is the usual way of using BDCAST statements.

`UMI 0|1|2|3` HEART boards have a reset FIFO feature. A HEART FIFO can be reset via UMI lines. For this to work, you have to specify what UMI line a FIFO is associated with. In the case that you want to be able to reset all HEART FIFO's related to a HEART connection, use the UMI keyword, and then specify the UMI line. You can specify more than 1 UMI line, separated by commas. The actual reset is then done by twiggling a UMI line. You can also associate single FIFO's with a UMI line by using a UMIRESET statement.

LISTEN syntax (HEART boards only)

A LISTEN statement tells the Server/Loader to create a one way FIFO connection from the HEART ring to a module. The statement defines a module that 'listens' to the time-slot you specify in this statement. A modules can 'broadcast' by sending data to the ring at that timeslot. A 'broadcast' connection can be created with the BDCAST statement. A LISTEN statement thus doesn't describe a situation, is tells the Server/Loader to perform an action.

```
LISTEN [broadcast][node][infifo][timeslots] <options>
Options: UMI 0|1|2|3
```

`[broadcast]` The broadcast to listen to. The broadcast must have been defined in an earlier BDCAST statement.

`[node]` The name of the node that will listen. This node must have been defined earlier with a C6, FPGA/HERONIO, GDIO, PCIF or IBC definition.

`[infifo]` The number of the fifo (0..5) on node to listen with.

`[timeslots]` The number of timeslots that must be used for this connection (1..6). It is also possible to specify precisely what timeslots to use: "t=0,3" tells the Server/Loader to use timeslots 0 and 3 (of 0..5). You can also use "v=0x81" to do the same. The "v=" tells the Server/Loader that you wish to use a 6-bit mask to specify what timeslots should be used. If no "t=" or "v=" specifier is used, the number specified is interpreted as the number of timeslots you wish to use. The Server/ Loader will allocate automatically proper timeslots. This is the usual way of using BDCAST statements.

`UMI 0|1|2|3` HEART boards have a reset FIFO feature. A HEART FIFO can be reset via UMI lines. For this to work, you have to specify what UMI line a FIFO is associated with. In the case that you want to be able to

reset all HEART FIFO's related to a HEART connection, use the UMI keyword, and then specify the UMI line. You can specify more than 1 UMI line, separated by commas. The actual reset is then done by twiggling a UMI line. You can also associate single FIFO's with a UMI line by using a UMIRESET statement.

BOOTSLOT syntax (HEART boards only)

The BOOTSLOT declaration informs the loader via what HEART timeslot to boot the defined module. By default, the Server/Loader will use the timeslot defined by this statement to boot the defined module. The Server/Loader will program HEART to create the desired FIFO connection. It is not necessary to set boot jumpers. In fact, any boot jumper setting will be ignored. You may even omit a BOOTSLOT statement at all. If there's no BOOTSLOT statement for a module, the Server/Loader assumes timeslot 0. As before, the Server/Loader will program HEART to create the desired FIFO connecti-on using timeslot 0. Thus, the only influence of a BOOTSLOT statement is to define over what timeslot a module is booted.

The behaviour is different with the '-j' option. If this is used, then the Server/Loader will not create any HEART FIFO connection. Instead, it assumes that you have set boot jumpers on the modules to create on-reset HEART FIFO connections. In this case you **must** use BOOTSLOT statements and the timeslot defined **must** match the boot jumpers on the module.

```
BOOTSLOT [module][timeslot]
```

`[module]`	The name of the node to boot.
`[timeslot]`	Timeslot over which to boot 'module'.

HOSTLINK syntax

The HOSTLINK declaration informs the loader that the host link will follow. The host link command should appear once only, describing the FIFO through which the root node is connected to the host.

```
HOSTLINK    [fifo_no]
[fifo_no]       Root node's fifo connected to the host system.
```

The FROMHOST declaration informs the loader there is a one-way connection between the host PC and the ROOT node in the DSP network. The ROOT DSP can read data sent by the host PC over this FIFO, but cannot write over this FIFO.

```
FROMHOST    [fifo_no]
[fifo_no]       Root node's FIFO connected to the host system.
```

The TOHOST declaration informs the loader there is a one-way connection between the ROOT node in the DSP network and the host PC. The ROOT DSP can write data to the host PC over this FIFO, but cannot read from this FIFO.

```
TOHOST      [fifo_no]
[fifo_no]       Root node's fifo connected to the host system.
```

UMIRESET syntax (HEART and HERON-BASE2 boards only)

HEART offers the possibility of being able to reset a (HEART) FIFO. This is done by 'associating' a FIFO with one or more UMI lines. The actual FIFO reset occurs each time the UMI line is made 'active'. Only FIFO's 'associated' with that UMI line will be reset. There are two ways you can 'associate' a FIFO with a UMI line: by using the optional UMI keyword in a HEART statement (see earlier), or using the UMIRESET statement. In the latter case, individual FIFO's are 'associated' with a UMI line.

```
UMIRESET [node][fifo][in|out][0|1|2|3]
```

`[node]`	The node on which there's a FIFO you want to 'associate' a UMI line with.
`[fifo]`	The FIFO you want to 'associate' a UMI line with.
`[in|out]`	Specify if the FIFO is an incoming FIFO (i.e. from HEART to the node) or an outgoing FIFO (i.e. from the node to HEART).
`[0|1|2|3]`	The UMI line you want to 'associate' the FIFO with. There are 4 UMI lines and you may select more than 1, separated by commas.

FIFONOBLOCK syntax (HERON-BASE2 boards only)

The HERON-BASE2 offers the possibility to configure a fifo in non-blocking mode.

```
FIFONOBLOCK [node][fifo][in|out]
FIFONOB     [node][fifo][in|out]
```

`[node]`	The node on which there's a FIFO you want to configure to non-blocking mode.
`[fifo]`	The FIFO you want to configure.
`[in|out]`	Specify if the FIFO is an incoming FIFO or an outgoing FIFO.

Example Network Description Files

Hepc9

For one HEPC9 with one DSP, one FPGA module, a GDIO module, and a HERONIO module, the network description file could look like:

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        hep9a           0               0

#-------------------------------------------------------
# Nodes description
# ND      BD NDNAME   NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
  c6      0  HERON1   ROOT   (0)   00000001 heron1.out
  heronio 0  heronio1 NORMAL       00000002 heronio1.rbt
  fpga    0  FPGA1    NORMAL       00000003 fpga1.rbt
  gdio    0  hegd2    NORMAL       00000004
```

```
  pcif    0  host     NORMAL       00000005

#--------------------------------------
# BOOTSLOT  ND_NAME  TIMESLOT
#--------------------------------------
  BOOTSLOT  HERON1   2

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0

#-----------------------------------------------
# HEART from:slot fifo  to:module fifo timeslots
#-----------------------------------------------
  HEART HERON1    0     host      3       1
  HEART host      3     HERON1    0       1
  HEART HERON1    1     heronio1  1       1
  HEART heronio1  1     HERON1    1       1
  HEART hegd2     5     FPGA1     4       1
  HEART FPGA1     3     HERON1    2       1
```

This will tell the Server/Loader to create a duplex connection between the DSP module and the host interface. A PC program can now communicate with the DSP by reading/ writing FIFO 3. The DSP can communicate with the PC by reading/writing FIFO 0.

The third and fourth statement will tell the Server/Loader to create a duplex connection between the DSP in slot 1 and the HERONIO in slot 2. The DSP can communicate with the HERONIO by reading/writing FIFO 1. The HERONIO can communicate with the DSP by reading writing FIFO 1.

The fifth statement asks the Server/Loader to create a one-way connection from the GDIO in slot 4 to the FPGA in slot 3. The GDIO outputs on FIFO 5, and the FPGA reads the GDIO data in at FIFO 4.

The sixth statement asks the Server/Loader to create a one-way connection from the FPGA in slot 3 to the DSP in slot 1. The DSP can read FPGA data from FIFO 2, but cannot write data back to the FPGA (in this example). The FPGA outputs its data onto FIFO number 3.

So in this example we have a GDIO (5) -> (4) FPGA (3) -> (2) DSP 'pipeline'. The DSP has duplex communications with the HERONIO over FIFO 1, and with the PC over FIFO 0. The PC can communicate then with DSP over host FIFO 3.

Hepc8

Example for a HEPC8 with 2 nodes, 1 FPGA module.

```
#-------------------------------------------------------
# Board description
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------
  BD API        hep8a           0               0

#-------------------------------------------------------
# Nodes description
# ND BD NDNAME NDType CC-id HERON-ID filename(s)
#-------------------------------------------------------
```

```
  c6   0  HERON1 ROOT   (0)  00000001 heron1.out
  c6   0  HERON2 NORMAL (1)  00000002 heron2.out
  fpga 0  FPGA1  NORMAL      00000003 mybitstream.rbt
#--------------------------------------
# Bootpath description.
# BOOTLINK  ND_NAME  PORT  ND_NAME  PORT
#--------------------------------------
  BOOTLINK  HERON1   2     HERON2   0

#-----------------------------------------------
# Number of the link connected to the host system
# HOSTLINK  PORT
#-----------------------------------------------
  HOSTLINK  0
```

The above network description describes a DSP network that has two C6x processors, int slot 1 and 2, and a FPGA module in slot 3, on the same HEPC8. The HEPC8 motherboard is to be accessed using the Hunt Engineering API as board number 0, device 0. The C6x in slot 1 of the HEPC8 is connected to the host via FIFO 0. And it is connected via FIFO 2 with FIFO 0 of the C6x in slot 2 of the HEPC8. (Note that this FIFO mapping may be different in your case, since FIFOs may be selected using jumpers on the HERON module).

Hepc3

```
# Server/Loader. Example Network Description File.
# Lines can be commented out with a `#`.
#
# Board description.
# BD  API    board   board number     device number
  BD API     hep3b        0                 0
#
# Nodes description
#  BD NAME  Type  CCid GBCW     LBCW     IACK     Program(s)
ND  0 NODE0 ROOT   (2) 00000000 00000000 002ff800 idrom.out root.out
ND  0 NODE1 NORMAL (1) 00000000 00000000 002ff800 idrom.out slave1.out
ND  0 NODE2 NORMAL (0) 00000000 00000000 002ff800 idrom.out slave2.out
#
# Bootpath description.
# BOOTLINK ND_NAME PORT   ND_NAME PORT
  BOOTLINK  NODE0   5      NODE1   2
  BOOTLINK  NODE1   5      NODE2   2
#
# Number of the link connected to the host system
# HOSTLINK PORT
  HOSTLINK   3
```

The above network description describes a DSP network that has 3 C4x processors, all situated on a HEPC4, HEPC3 or HECPCI1. The motherboard is to be accessed using the Hunt Engineering API as board number 0, device 0. The C1x in slot 1 is connected to the host via comport 3. And it is connected via comport 5 to comport 2 of the C4x in slot 2. In its turn, the second C4x is connected to a third C4x via its comport 5 to the third C4x's comport 2. (Note that this comport mapping may be different in your case.)

HEART fifo reset using UMI

HEART has a feature that allows you to reset a HEART fifo. This is done by attaching a UMI line to 1 or more HEART fifo's. By setting the UMI line to active, the HEART fifo's attached to that line are then reset.

The network file specification allows to ways two define how a UMI line is to be attached to a fifo. First, there's a specific keyword (UMIRESET) that specifically selects a FIFO to be attached to a UMI line: -

UMIRESET node fifo in umi

for a HEART to node fifo, and

UMIRESET node fifo out umi

for a node to HEART fifo. Here 'node' is the name of a node, fifo is the fifo number (0..5), in or out are keywords, and umi is the umi line number (0..3). However, you may want to be able to reset all fifo's in a connection between two nodes. In that case, it is easier / simpler to use the optional UMI keyword in a HEART, MCAST, BDCAST or LISTEN statement. For example: -

HEART nodea 2 nodeb 4 1 umi 2

defines a HEART connection between 'nodea' fifo 2 and 'nodeb' fifo 4, 1 timeslot, and defines all fifo's involved to be attached to UMI line 2.

Finally, note that the actual fifo reset is done by setting the UMI line to active. In the network file you only define (associate) certain fifo's with certain UMI lines.

Using Inter-Board Connectors

Some HEART boards, such as the HEPC9, can accept Inter-Board Connectors, such as the EM2, EM1 and EM1C. Using Inter-Board Connectors boards can be connected. HSB and reset may be propagated over board-to-board links, allowing HSB exchange between nodes on different boards and allowing a reset on 1 board to reset all connected boards as well. Using HEART statements it is also possible to wire node-to-node fifo connections over board-to-board links. This can be done explicitly, for example:

```
BD API hep9a 0 0
BD API hep9a 1 0

c6   0      nodea    ROOT     (1)    0x01  module1.out
em2  0      em2a     normal          0x06
c6   1      nodeb    ROOT     (0)    0x01  module2.out
em2  1      em2b     normal          0x06

heart  nodea  0   em2a   1   1
heart  em2b   1   nodeb  0   1
```

This describes a connection over an EM2 link (channel 1 (em2a) to channel 1 (em2b)) from nodea fifo 0 to nodeb fifo 0, using 1 timeslot.

However, you can also declare board-to-board connections using BDCONN, and in that case you can connect any nodes. HeartConf or Server/Loader will then automatically route via Inter-Board Connectors, using the information provided in BDCONN statements. The same example as above would become: -

```
BD API hep9a 0 0
BD API hep9a 1 0

c6   0      nodea    ROOT     (1)    0x01  module1.out
em2  0      em2a     normal          0x06
c6   1      nodeb    ROOT     (0)    0x01  module2.out
em2  1      em2b     normal          0x06

BDCONN em2a 1 em2b 1

heart  nodea 0   nodeb 0   1
```

Instead of a BDCONN you could also use a BDLINK statement. This statement declares board-to-board connections without using Inter-Board Connectors explicitly named. It uses two board indices instead. For example, with the above statement one could replace the BDCONN with an equivalent: -

```
BDLINK 0 1 1 1
```

thus, essentially, the em2a node is replaced by the board index of the board it's on, and the em2b node is replaced by the board index of the board the em2b node is on. Why would you use one form above the other? No reason really, just choose what fits you best. In both cases you must have defined Inter-Board Connectors; with a BDLINK statement the Server /Loader or HeartConf will trace back what Inter-Board Connector is used on that board.

Note that 1 board-to-board connection can carry only one node-to-node connection. Thus, if you have 2 node-to-node connections where the connections will have to pass Inter-Board Connectors, you need 1 link per node-to-node connection. For example:

```
BD API hep9a 0 0
BD API hep9a 1 0

c6   0      nodea    ROOT     (3)    0x01  module1.out
c6   0      nodeb    normal   (2)    0x02  module1.out
em2  0      em2a     normal          0x06
c6   1      nodec    ROOT     (0)    0x01  module2.out
c6   1      noded    normal   (0)    0x02  module2.out
em2  1      em2b     normal          0x06

BDCONN em2a 1 em2b 1

heart  nodea 0   nodec 0   1
heart  nodeb 2   noded 3   1
```

In this example, HeartConf or Server/Loader will return an error indicating it can't route 2 connections over 1 board-to-board connection. In this example, you would need 2 channel connections between the two boards: -

```
BDCONN em2a 1 em2b 1
BDCONN em2a 2 em2b 2
```

I.e. we have specified there are now two cables between em2a and em2b (channel 1 (em2a) to channel 1 (em2b) and channel 2(em2a) to channel 2(em2b)).

If you need more bandwidth between two nodes, you can increase the number of timeslots. For example, continuing on the earlier examples: -

```
heart  nodea 0   nodec 0   2
```

However, note that the maximum bandwidth between two Inter-Board Connectors may be less than the number of timeslots you specified. For example, the maximum bandwidth of an EM2 to EM2 connection per cable is 125 Mb/sec, whereas two timeslots on HEART would give you 132 Mb/sec.

Server links

Server links are only used by the Server/Loader, not by HeartConf. The Server part of the Server/Loader must try to find out what nodes need or want to be served. With non-HEART boards, such as the HEPC8 or HEPC3, this was easy. With such boards only one node, the ROOT node is connected to the host; so finding the ROOT node also tells you what node needs to be served. With HEART boards, however, all nodes can be connected to the host.

The server will search the HEART statements, and will detect all nodes that have a duplex connection with a host. This includes node-to-host connections that go via Inter-Board Connectors. It is thus possible to serve nodes on remote boards.

If there is more than 1 duplex node-to-host connection, the server chooses one connection to serve. This is effectively a random choice. In case that you wish to use the Server together with an API program that accesses the same host interface, you may want to be able to create a duplex connection of which you can be sure it isn't used by the Server. That can be done by using the optional NOSERVE keyword. For example:

```
heart  nodea 0   host  0   1
heart  host  0   nodea 0   1
heart  nodea 1   host  1   1   NOSERVE
heart  host  1   nodea 1   1   NOSERVE
```

This will force the Server/Loader to use fifo a to serve nodea, while ensuring that fifo b is free for use for nodea-to-host communications for your own API application. Of course, if you don't use the Server all of this doesn't matter.

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng.demon.co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

If you are in North America, South America or Canada, contact our strategic partner Traquair Data Systems at www.traquair.com/company/support.html for support information and contact details.