

## IP Address Configuration

initialization in CPU code for any network application

To set the local IP address and netmask:

```
max_ip.config (dh, cn, ip, nm)
```

where *nm* is the netmask, a pointer to type struct in\_addr as declared in #include <netinet/in.h>.

## TCP

for working with persistent and reliable connections

### Network Manager Interface

To create a pair *T* of TCP streams, a TCP stream *x* receiving, and a TCP stream *y* transmitting to the network:

```
TCPStream T = addTCPStream (st, co)
DFELink x = T.getReceiveStream ()
DFELink y = T.getTransmitStream ()
```

### Standard Manager Interface

Connect TCP streams named in the Kernel to the network:

```
mg.setIO(
    Manager.link (st, Manager.TCP (co)),
    Manager.link (st, Manager.TCP (co)))
```

### CPU Interface

To create a TCP socket *ts*, to find its socket number *sn*, and to close the socket:

```
ts = max_tcp.create_socket (dh, st)
sn = max_tcp.get_socket_number (ts)
max_tcp.close (ts)
```

To connect to a remote server:

```
max_tcp.connect (ts, ip, pn)
```

To accept connections from remote clients:

```
max_tcp.listen (ts, pn)
```

To monitor connections from remote clients by waiting for a specified connection state:

```
rs = max_tcp.await_state (ts, cs, to)
```

with

```
cs the awaited connection state
to the timeout value (of type struct timeval*)
rs zero iff the state is reached before timeout
```

where connection states are

```
MAX_TCP_STATE_CLOSED
MAX_TCP_STATE_LISTEN
MAX_TCP_STATE_ESTABLISHED
MAX_TCP_STATE_CLOSE_WAIT
MAX_TCP_STATE_CLOSED_DATA_PENDING
```

## UDP

for transferring packets statelessly

### Network Manager Interface

To create a pair *U* of UDP streams, a UDP stream *x* receiving and a UDP stream *y* transmitting to the network:

```
UDPStream U = addUDPStream (st, co, cm, sm)
DFELink x = U.getReceiveStream ()
DFELink y = U.getTransmitStream ()
```

with *sm* either DropBadFrames, FlagOnEOF or Disabled

### Standard Manager Interface

Connect UDP streams named in the Kernel to the network:

```
mg.setIO(
    Manager.link (st, Manager.UDP (co, cm)),
    Manager.link (st, Manager.UDP (co, cm)))
```

### CPU Interface

To create a UDP socket *us*, to find its socket number *sn*, and to close the socket:

```
us = max_udp.create_socket (dh, st)
sn = max_udp.get_socket_number (us)
max_udp.close (us)
```

To let a UDP socket receive data:

```
max_udp.bind (us, pn)
```

To let a OneToOne mode UDP socket send data:

```
max_udp.connect (us, ip, pn)
```

## Ethernet

for handling network traffic on a low level

### Network Manager Interface

To create a pair *E* of Ethernet streams, an Ethernet stream *x* receiving and an Ethernet stream *y* transmitting to the network:

```
EthernetStream E = addEthernetStream (st, co, em)
DFELink x = E.getReceiveStream ()
DFELink y = E.getTransmitStream ()
```

with *em* either DropBadFrames or FlagOnEOF

### Standard Manager Interface

Connect Ethernet streams named in the Kernel to the network:

```
mg.setIO(
    Manager.link (st, Manager.ETHERNET (co)),
    Manager.link (st, Manager.ETHERNET (co)))
```

### CPU Interface

To read the default MAC addresses from hardware:

```
max_eth.get_default_mac_address (dh, cn, mac)
```

where *mac* is a pointer to a struct ether\_addr as defined in the standard #include <net/ethernet.h>

## Framed Streams

for transferring framed data from the DFE to the CPU

### Network Manager Interface

To declare a stream *S* in the Manager:

```
DFELink S = addFramedStreamToCPU (st, ty, as, bs)
```

with

```
ty a FramedBusType
as the alignment size (optional integer parameter)
bs the buffer size (optional integer parameter)
```

### CPU Interface

#### initialization/reclamation

To create a handle *fh* of type max\_framed\_stream\_t\*, and to free the handle:

```
fh = max_framed_stream_setup (dh, st, bu, bs)
max_framed_stream_release (fh)
```

with

```
bu a pointer to a buffer storage area
bs the size of the buffer in bytes
```

#### usage

Transfer data by alternately requesting and acknowledging receipt of a number of frames.

```
fr = max_framed_stream_read (fh, rf, fp, fs)
max_framed_stream_discard (fh, af)
```

with

```
rf the requested number of frames
fp a pointer to an array of pointers to frame buffers
fs a pointer to an array of the frame buffer sizes
fr the number of frames actually read
af the number of frames acknowledged (≤ fr)
```

## Framed Kernels

for transparent marshalling and tunneling of framed data

### Creating Framed Stream Formats

Define a class extending FrameFormat. In the constructor method, call one of:

```
super (ByteOrder.LITTLE_ENDIAN)
super (ByteOrder.BIG_ENDIAN)
```

Define any number of fixed sized fields:

```
fr = addField ("id", ty)
```

with

```
"id" the name of the field
ty the type of the field as any Kernel type
```

Define any number of variable sized fields:

```
addVariableLengthField ("id", ty, min, max[, gr])
fd.setSizeForVariableField ("id", sz)
```

where "*fd*" is an input or output FrameData instance, "*id*" and *ty* are as above, and also:

```
min minimum number of elements
max maximum number of elements
gr number of elements per transfer (optional)
sz DFEVar where the number of elements is stored
```

### Creating Framed Streams

To create a framed input *fdi* and a framed output *fdo*:

```
FrameData<F> fdi = io.frameInput(st, new F(· · ·), bt)
FrameData<F> fdo = new FrameData<F>(this, new F(· · ·))
```

where class *F* extends FrameFormat, and *bt* is a bus type:

```
TCPType
UDPOneToOneRXType
UDPOneToOneTXType
UDPOneToManyRXType
UDPOneToManyTXType
```

### Using Framed Streams

Methods on an input FrameData *fdi*:

```
fdi["id"] value of frame field "id"
fdi.isStart() implies data is available
fdi.linkfield["id"] readable field from the bus
```

Methods on an output FrameData *fdo*:

```
fdo["id"] <== x assign a frame field
fdo.linkfield["id"] <== y writable field from the bus
io.frameOutput (st, fdo, ev) transmit
```

where *ev* is a boolean indicating frames are initialized, typically *fdi.isStart()*, if the output derives from an input *fdi*

## Common Parameters

```
dh a device handle returned by max_open_device
st a stream name as a character string
ts a TCP socket of type max_tcp.socket_t*
ip an IP address of type struct in_addr*
us a UDP socket of type max_udp.socket_t*
cm a connection mode, OneToOne or OneToMany
pn a local or remote port number of type uint_16
sn a socket number from 0 to 63 for TCP
or from 0 to 15 for UDP
mg a standard Manager object
co Max3NetworkConnection.CH2_SFP1
or Max3NetworkConnection.CH2_SFP2
cn MAX_NET_CONNECTION.CH2_SFP1
or MAX_NET_CONNECTION.CH2_SFP2
```