MANCHESTER
1824

The University
of Manchester

SpiNNaker

a universal Spiking Neural Network architecture

version 0.1 - DRAFT
16 August 2006

SpiNNaker - a chip multiprocessor for neural network simulation

Features

- *20* ARM968 processors, each with:
  - *64* Kbytes of tightly-coupled data memory;
  - *32* Kbytes of tightly-coupled instruction memory;
  - DMA controller;
  - communications controller;
  - interrupt controller;
  - low-power ‘wait for interrupt’ mode.
- Multicast communications router
  - *6* serial inter-chip receive interfaces;
  - *6* serial inter-chip transmit interfaces;
  - *1024* associative routing entries.
- Interface to external SDRAM
  - over 1 Gbyte/s sustained block transfer rate.
- Fault-tolerant architecture
  - defect detection, isolation, and function migration.
- Boot, test and debug interfaces (to be determined).

Introduction

SpiNNaker is a chip multiprocessor designed specifically for the real-time simulation of large-scale spiking neural networks. Each chip (along with its associated SDRAM chip) forms one node in a scalable parallel system, interconnected to the other nodes through self-timed links.

The processing power is provided through the multiple ARM cores on each chip. Each ARM models multiple (up to 1,000) neurons, with each neuron being a coupled pair of differential equations modelled in continuous ‘real’ time. Neurons communicate through atomic ‘spike’ events, and these are communicated as discrete packets through the on- and inter-chip communications fabric. The packet contains a routing key that is defined at its source and is used to implement multicast routing through an associative router in each chip.

One processor on each SpiNNaker chip will perform system management functions; the communications fabric supports point-to-point packets to enable co-ordinated system management across local regions and across the entire system, and nearest-neighbour packets are used for system flood-fill boot operations and for chip debug.

Background

SpiNNaker was designed at the University of Manchester within an EPSRC-funded project in collaboration with the University of Southampton, ARM Limited and Silistix Limited. The work would not have been possible without EPSRC funding, and the support of the EPSRC and the industrial partners is gratefully acknowledged.

Intellectual Property rights

Disclaimer

Change history

version	date	changes
0.0	27/12/05	first draft
0.1	16/8/06	sundry - document still developing

Contents

1. Chip organization 5
1.1 Block diagram 5
1.2 System-on-Chip hierarchy 6

2. System architecture 7
2.1 Routing 7
2.2 System-level address spaces 8

3. ARM968 processing subsystem 9
3.1 Features 9
3.2 ARM968 subsystem organisation 9
3.3 Fault-tolerance 9
3.4 Test 9

4. ARM 968 10
4.1 Features 10
4.2 Organization 10
4.3 Fault-tolerance 10
4.4 Test 10

5. Counter/timer and interrupt controller 11
5.1 Features 11
5.2 Register summary 11
5.3 Register details 12
5.4 Fault-tolerance 12
5.5 Test 12
5.6 Notes 12

6. DMA controller 13
6.1 Features 13
6.2 Register summary 13
6.3 Register details 13
6.4 Fault-tolerance 13
6.5 Test 13

7. Communications controller 14
7.1 Features 14
7.2 Packet formats 14
7.3 Control byte summary 15
7.4 Register summary 17
7.5 Register details 17
7.6 Fault-tolerance 19
7.7 Test 19
7.8 Notes 19

8. Communications NoC 21
8.1 Features 21
8.2 Block diagram 21
8.3 Arbiter structure 21
8.4 Fault-tolerance 22
8.5 Test 22
8.6 Notes 22

9. Communications Router 23

9.1 Features 23
9.2 Description 23
9.3 Internal organization 23
9.4 Multicast (MC) router 24
9.5 The point-to-point (p2p) router 27
9.6 The algorithmic (ALG) router 27
9.7 Time phase handling 27
9.8 Packet error handler 27
9.9 Emergency routing 28
9.10 Errant packets 28
9.11 Pseudo-code description 28
9.12 Fault-tolerance 31
9.13 Test 31
9.14 Notes 31

10. Inter-chip transmit and receive interfaces 32
10.1 Features 32
10.2 Programmer view 32
10.3 Fault-tolerance 32
10.4 Test 33

11. System NoC 34
11.1 Features 34
11.2 Fault-tolerance 34
11.3 Test 34

12. SDRAM interface 35
12.1 Features 35
12.2 Fault-tolerance 35
12.3 Test 35

13. System Controller 36
13.1 Features 36
13.2 Register summary 36
13.3 Register details 36
13.4 Fault-tolerance 36
13.5 Test 36

14. Router configuration registers 37
14.1 Features 37
14.2 Register summary 37
14.3 Register details 37
14.4 Fault-tolerance 37
14.5 Test 37

15. System RAM 38
15.1 Features 38
15.2 Fault-tolerance 38
15.3 Test 38

16. Boot ROM 39
16.1 Features 39
16.2 Fault-tolerance 39
16.3 Test 39

17. Boot, test and debug support 40
17.1 Features 40

17.2 Issues 40
17.3 Boot algorithm 40
17.4 Fault-tolerance 40
17.5 Test 40

18. Input and Output signals 41
18.1 External SDRAM interface 41
18.2 41

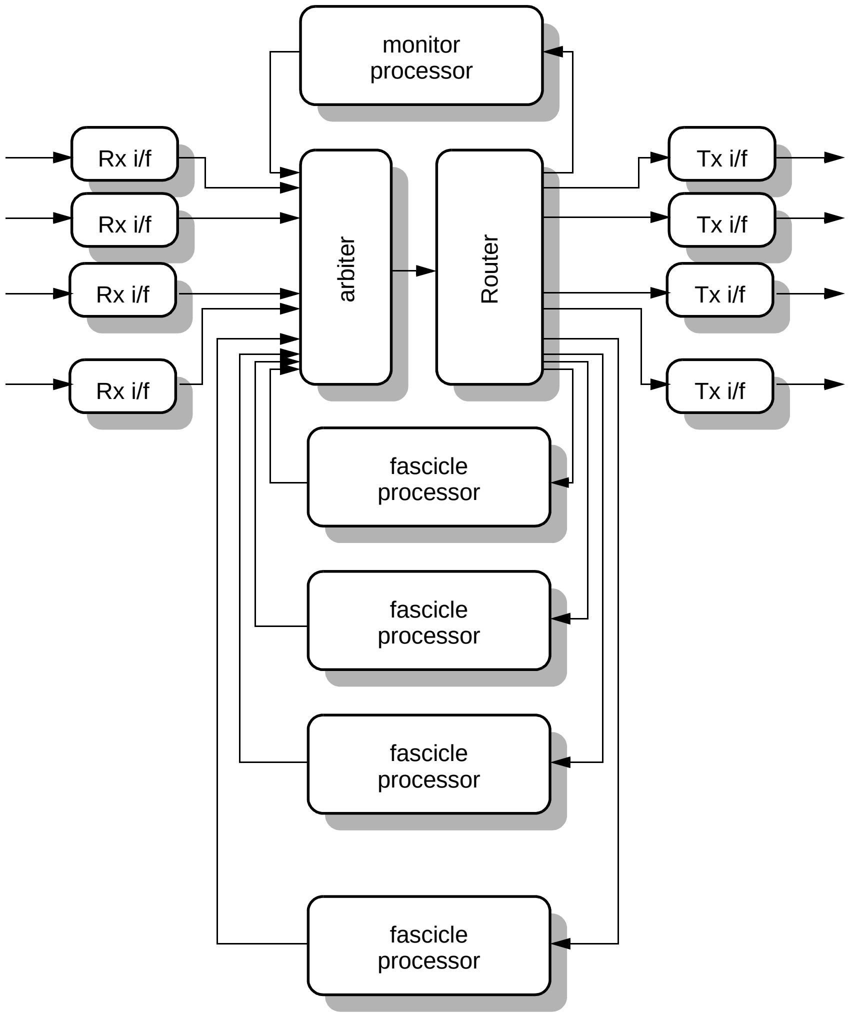
19. Area estimates 42

20. Power estimates 43

1. Chip organization

1.1 Block diagram

The primary functional components of SpiNNaker are illustrated in the figure below, which shows the Communications NoC and its clients.



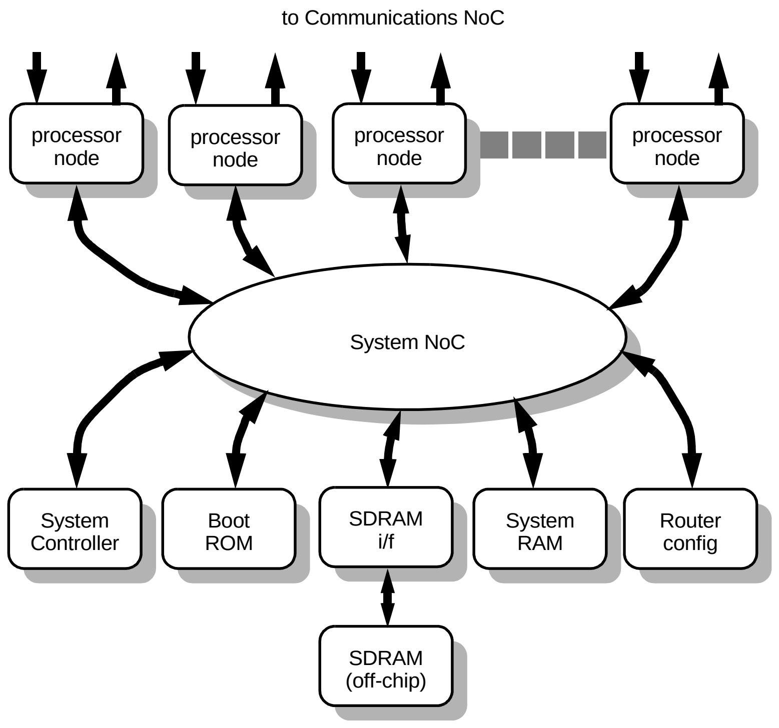
Each chip contains *20* identical processing subsystems each of which is responsible for modelling a number of neurons with associated inputs and outputs - a fascicle.

Following self-test, at start-up one of the processors is nominated as the Monitor Processor and thereafter performs system management tasks.

The router is responsible for routing neural event packets both between the on-chip fascicle processors and from and to other SpiNNaker chips. The Tx and Rx interface components are used to extend the on-chip communications NoC across to other SpiNNaker chips. The arbiter assembles inputs from the various on- and off-chip sources into a single serial stream which is then passed to the Router.

In addition to the primary function, there are additional resources accessible from the processor systems via the System NoC. Each of the fascicle processors has access to the shared off-chip SDRAM, and various system components also connect through the System NoC in order that, whichever processor is Monitor Processor, it will have access to these components.

The sharing of the SDRAM is an implementation convenience rather than a functional requirement, although it may facilitate function migration in support of fault-tolerant operation.



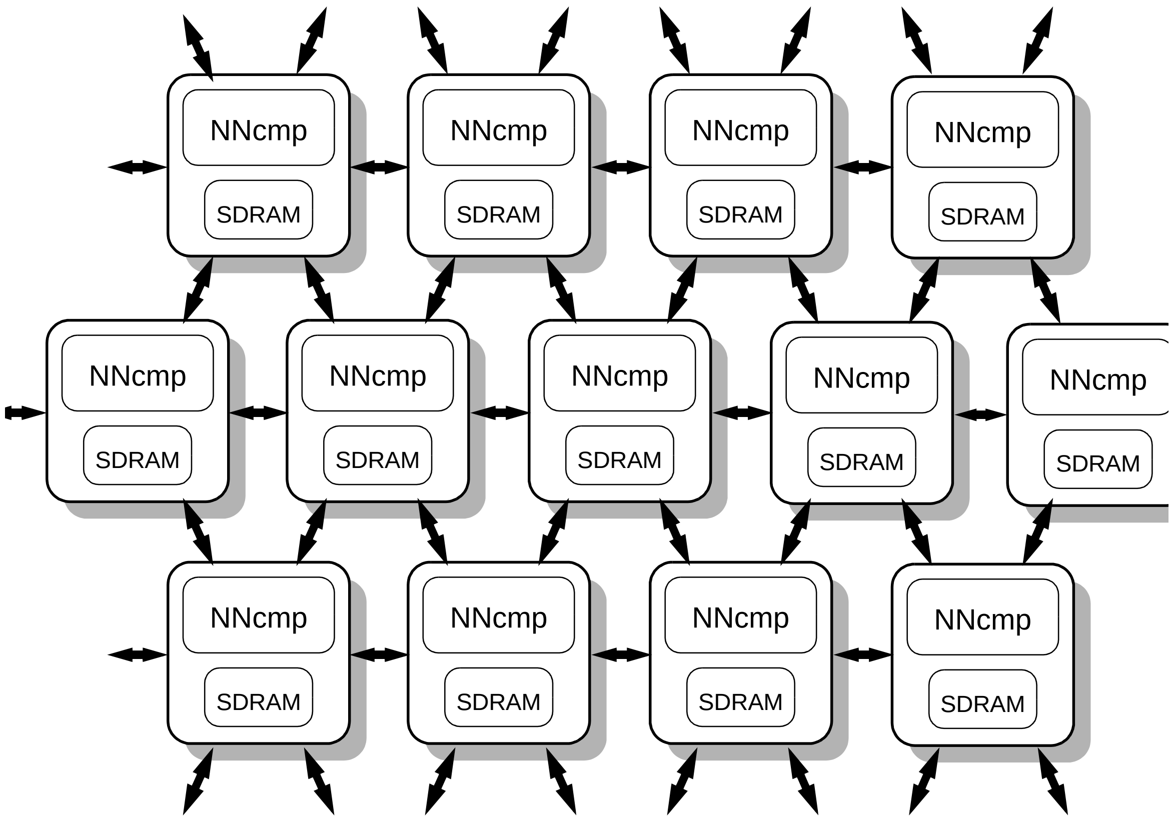
1.2 System-on-Chip hierarchy

The SpiNNaker chip is viewed as having the following structural hierarchy, which is reflected throughout the organisation of this datasheet:

- ARM968 processor subsystem
  - the ARM968, with its tightly-coupled instruction and data memories
  - Timer/counter and interrupt controller
  - DMA controller
  - communications controller, including communications NoC interface
  - System NoC interface
- Communications NoC
  - Router, including multicast, algorithmic, default and emergency routing functions
  - *6* inter-chip transmit interfaces
  - *6* inter-chip receive interfaces
  - communications NoC arbiter and fabric
- System NoC
  - SDRAM interface
  - System Controller
  - Router configuration registers
  - Boot ROM
  - System RAM
  - System NoC arbiter and fabric
- Boot, test and debug
  - central controller for ARM968 JTAG functions

2. System architecture

SpiNNaker is designed to form (with its associated SDRAM chip) a node of a massively parallel system. The system architecture is illustrated below:



2.1 Routing

The nodes are arranged in a *hexagonal* mesh with bidirectional links to *6* neighbours. The system supports multicast packets (to carry neural event information, routed by the associative Multicast Router), point-to-point packets (to carry system management and control information, routed algorithmically) and nearest-neighbour packets (to support boot-time flood-fill and chip debug).

Emergency routing

In the event of a link failing or congesting, traffic that would normally use that link is redirected in hardware around two adjacent links that form a triangle with the failed link. This “emergency routing” is intended to be temporary, and the operating system will identify a more permanent resolution of the problem. The local Monitor Processor is informed of all uses of emergency routing.

Deadlock avoidance

The communications system has potential deadlock scenarios because of the possibility of circular dependencies between links. The policy used here to prevent deadlocks occurring is:

- ***no Router can ever be prevented from issuing its output***.

The mechanisms used to ensure this are the following:

- outputs have sufficient buffering and capacity detection so that the Router knows whether or not an output has the capacity to accept a packet;
- emergency routing is used, where possible, to avoid overloading a blocked output;

- where emergency routing fails (because, for example, the alternative output is also blocked) the packet is ‘dropped’ to the local Monitor Processor;
- the local Monitor Processor is guaranteed to accept the dropped packet (eventually).

The expectation is that the communications fabric will be lightly-loaded so that blocked links are very rare. Where the operating system detects that this is not the case it will take measures to correct the problem by modifying routing tables or migrating functionality to a different part of the system.

Errant packet trap

Packets that get mis-routed could continue in the system for ever, following cyclic paths. To prevent this all packets are time stamped and a coarse global time phase signal is used to trap old packets. To minimize overhead the time stamp is 2 bits, cycling 00 -> 01 -> 11 -> 10, and when the packet is two time phases old (time sent XOR time now = 0b11) it is dropped to the local Monitor Processor and an error flagged. The length of a time phase can be adapted dynamically to the state of the system; normally timed-out packets should be very rare so the time phase can be conservatively long to minimise the risk of packets being dropped due to congestion.

2.2 System-level address spaces

The system incorporates a number of different levels of component that must be enumerated in some way:

- Each Node (where a Node is an SpiNNaker chip plus SDRAM) must have a unique, fixed address which is used as the destination ID for a point-to-point packet, and the addresses must be organised logically for algorithmic routing to function efficiently.
- Processors will be addressed relative to their host Node address, but this mapping will not be fixed as an individual Processor’s role can change over time. Point-to-point packets addressed to a Node will be delivered to the local Monitor Processor, whichever Processor is serving that function. Internal to a Node there will be some hard-wired addressing of each Processor for system diagnosis purposes, but this mapping will be hidden outside the Node.
- Neurons ocuppy an address space that identifies each Neuron uniquely within the domain of its multicast routing path (where this domain must include alternative links that may be taken during emergency routing). Where these domains do not overlap it is possible to reuse the same address, though this must be done with considerable care. Neuron addresses can be assigned arbitrarily, and this flexibility can be exploited to optimize Router utilization (for example by giving Neurons with the same routing requirements related addresses so that they can all be routed by the same Router entries).

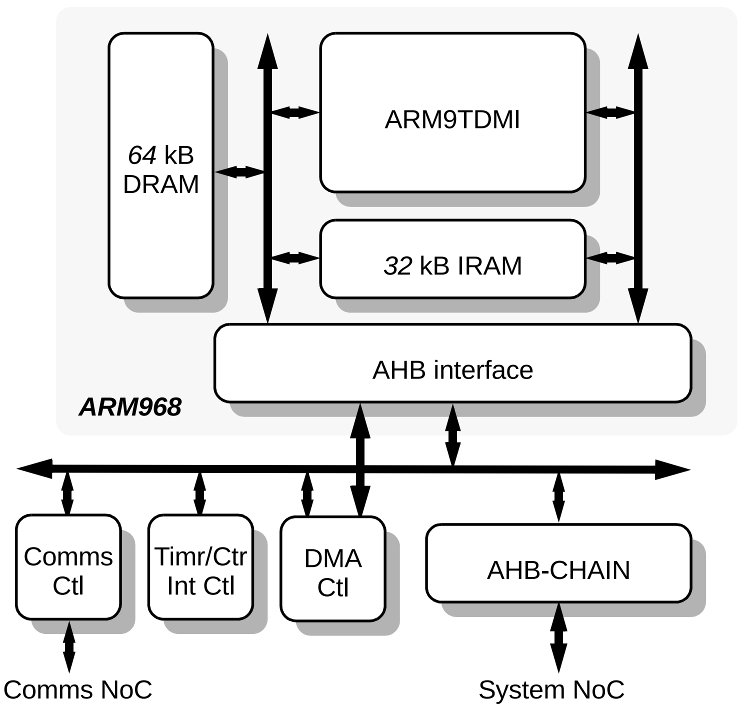
3. ARM968 processing subsystem

SpiNNaker incorporates *20* ARM968 processing subsystems which provide the computational capability of the device. Each of these subsystems is capable of generating and processing neural events communicated via the Communications NoC and, alternatively, of fulfilling the role of Monitor Processor.

3.1 Features

- a synthesized ARM968 module with
  - a 200 MIPS ARM9 processor
  - *32* kB tightly-coupled instruction memory
  - *64* kB tightly-coupled data memory
- a local AHB with
  - communications controller connected to Communications NoC
  - wrapper to interface to the System NoC
  - DMA controller
  - timer/counter and interrupt controller

3.2 ARM968 subsystem organisation



3.3 Fault-tolerance

The fault-tolerance of the ARM968 subsystem is defined in terms of its component parts, described below.

3.4 Test

The test strategies for the ARM968 subsystem are likewise defined in terms of its component parts.

4. ARM 968

The ARM968 (with its associated tightly-coupled instruction and data memories) forms the core processing resource in SpiNNaker. It is a standard synthesizable IP component from ARM Ltd, and as such there is limited scope for customizing it for this application.

4.1 Features

- 200 MIPS ARM9TDMI processor.
- *32* kB tightly-coupled instruction memory (I-RAM).
- *64* kB tightly-coupled data memory (D-RAM).
- AHB interface to external system.

4.2 Organization

See ARM DDI 0311C – the ARM968E-S datasheet.

4.3 Fault-tolerance

Fault insertion

- ARM9TDMI can be disabled.
- Software can corrupt I-RAM and D-RAM to model soft errors. (Can these be detected?)

Fault detection

- The I-RAM and D-RAM are protected by parity bits?
- A watchdog timer can catch runaway software.
- Self-test routines, run at start-up and during normal operation, can detect faults.

Fault isolation

- The ARM968 unit can be disabled from the System Controller.
- Defective locations in the I-RAM and D-RAM can be mapped out of use by software.

Reconfiguration

- Software will avoid using defective I-RAM and D-RAM locations.
- Functionality will migrate to an alternative Processor in the case of permanent faults that go beyond the failure of one or two memory locations.

4.4 Test

production test

start-up test

run-time test

5. Counter/timer and interrupt controller

Each processor node on an SpiNNaker chip has a local counter/timer and interrupt controller that is used to enable and disable interrupts from various sources, and to wake the processor from sleep mode when required. The interrupt controller provides centralised management of IRQ and FIQ sources, and offers an efficient indication of the active sources for vectoring purposes.

5.1 Features

- manages the various interrupt sources to each local processor:
  - arriving multicast packet with/without payload
  - arriving point-to-point packet with/without payload
  - arriving nearest-neighbour packet with/without payload
  - DMA complete
  - timer 1 & timer 2 interrupts
  - interrupt from another processor on the chip
  - packet-error interrupt from the Router
  - system fault interrupt
- the counter/timer unit provides two independent counters, for example for:
  - millisecond interrupts for real-time dynamics
  - watchdog timer for fault trapping

5.2 Register summary

Name	Offset	R/W	Function
r0: FIQ status	0x0	R	active FIQ interrupt sources
r1: FIQ enable	0x4	W	enable FIQ interrupt sources
r2: FIQ inputs	0x8	R	state of FIQ request inputs
r3:			
r4: IRQ status	0x0	R	active IRQ interrupt sources
r5: IRQ enable	0x4	W	enable IRQ interrupt sources
r6: IRQ inputs	0x8	R	state of IRQ request inputs
r7:			
r8:			

5.3 Register details

	7	6	5	4	3	2	1	0
reset to:	0	0	0	0	0	0	0	0

5.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

5.5 Test

production test

start-up test

run-time test

5.6 Notes

- millisecond interrupt could be provided from a centralised C/T unit? But will all processors want to receive the same time interrupts?
- extra interrupt(s) for packet parity failure?

6. DMA controller

Each ARM968 processing subsystem includes a DMA controller. The DMA controller is primarily used for transferring inter-neural connection data from the SDRAM in large blocks in response to an input event arriving at a fascicle processor, and for returning updated connection data during learning.

6.1 Features

•

6.2 Register summary

Name	Offset	R/W	Function

6.3 Register details

	7	6	5	4	3	2	1	0
reset to:	0	0	0	0	0	0	0	0

6.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

6.5 Test

production test

start-up test

run-time test

7. Communications controller

Each processor node on SpiNNaker includes a communications controller which is responsible for generating and receiving packets to and from the communications network.

7.1 Features

- Support for 3 packet types:
  - 40-bit multicast neural event packets routed by a key provided at the source;
  - 40-bit point-to-point packets routed algorithmically by destination address;
  - 40-bit nearest-neighbour packets routed algorithmically by arrival port.
- Packets may optionally carry an additional 32-bit payload.
- 2-bit time stamp (used by Routers to trap errant packets).
- Parity (to detect corrupt packets).

7.2 Packet formats

Neural event multicast (mc) packets (type 0)

Neural event packets include a 32-bit routing key inserted by the source, and a control byte:

32 bits	8 bits
routing key	control

In addition they may include an optional (not normally used) 32-bit payload:

32 bits
payload

The 8-bit control field includes packet type (= 00 for multicast packets), emergency routing and time stamp information, a data payload indicator, and error detection (parity) information:

7	6	5	4	3	2	1	0
0	0	emergency routing		time stamp		data	parity

Point-to-point (p2p) packets (type 1)

Point-to-point packets include 16-bit source and destination chip IDs, plus a control byte and an optional (normally used) 32-bit payload:

16 bits	16 bits	8 bits
source ID	destination ID	control

32 bits
payload

Here the 8-bit control field includes packet type (=01 for p2p packets), a sequence code, time stamp, a data payload indicator and error detection (parity) information:

7	6	5	4	3	2	1	0
0	1	seq code		time stamp		data	parity

Nearest-neighbour (nn) packets (type 2)

Nearest-neighbour packets include a 32-bit address or operation field, plus a control byte and an optional 32-bit payload:

32 bits	8 bits
address/operation	control

32 bits
payload

Here the 8-bit control field includes packet type (= 10 for nn packets), routing information, a data payload indicator and error detection (parity) information:

7	6	5	4	3	2	1	0
1	0	T	route			data	parity

7.3 Control byte summary

Field Name	bits	Function
parity	0	parity of complete packet (including payload when used)
data	1	data payload (1) or no data payload (0)
time stamp	3:2	phase marker indicating time packet was launched
seq code	5:4	p2p only: start, middle odd/even, end of payload
emergency routing	5:4	mc only: used to control routing around a failed link
route	4:2	nn only: information for the Router
T: nn packet type	5	nn only: packet type - normal (0) or direct (1)
packet type	7:6	= 00 for mc; = 01 for p2p; = 10 for nn

parity

The complete packet (including the data payload where used) will have odd parity.

data

Indicates whether the packet has a 32-bit data payload (=1) or not (=0).

time stamp

The system has a global time phase that cycles through 00 -> 01 -> 11 -> 10 -> 00. Global synchronisation must be accurate to within less than one time phase (the duration of which is programmable and may be dynamically variable). A packet is launched with a time stamp equal to the current time phase, and if a Router finds a packet that is two time phases old (time now XOR time launched = 11) it will drop it to the local Monitor Processor. The time stamp is inserted by the local Router, so the Communication Controller need do nothing here.

seq code

p2p packets use these bits to indicate the sequence of data payloads:

- 11 -> start packet: the first packet in a sequence (of >1 packets)
- 10 -> middle even: the second, fourth, sixth, ... packet in a sequence
- 01 -> middle odd: the third, fifth, seventh, ... packet in a sequence
- 00 -> end: the last (or only) packet in a sequence

emergency routing

mc packets use these bits to control emergency routing around a failed or congested link:

- 00 -> normal mc packet;
- 01 -> the packet has been redirected by the previous Router through an emergency route along with a normal copy of the packet. The receiving Router should treat this as a combined normal plus emergency packet.
- 10 -> the packet has been redirected by the previous Router through an emergency route which would not be used for a normal packet.
- 11 -> this emergency packet is reverting to its normal route.

route

These bits are set at packet launch to the values defined in the control register. They enable a packet to be directed to a particular neighbour (0 - 5), to all neighbours (6), or to the local Monitor Processor (7).

T (nn packet type)

This bit specifies whether an nn packet is ‘normal’, so that it is delivered to the Monitor Processor on the neighbouring chip(s), or ‘direct’, so that performs a read or write access to the neighbouring chip’s System NoC resource.

packet type

These bits indicate whether the packet is a multicast (00), point-to-point (01) or nearest-neighbour (10) packet. Packet type 11 is reserved for future use.

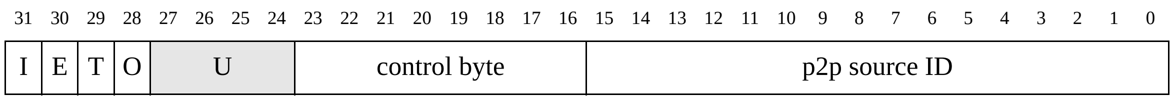
7.4 Register summary

Name	Offset	R/W	Function
r0: Tx control	0x0	R/W	Controls packet transmission
r1: Rx status	0x4	R/W	Indicates packet reception status
r2: send data	0x8	W	32-bit data for transmission
r3: send key	0xC	W	Send mc key/p2p dest ID & seq code
r4: receive data	0x10	R	32-bit received data
r5: receive key	0x14	R	Received mc key/p2p source ID & seq code

A packet will contain data if r2 is written before r3; this can be performed using an ARM STM instruction.

7.5 Register details

r0: transmit control

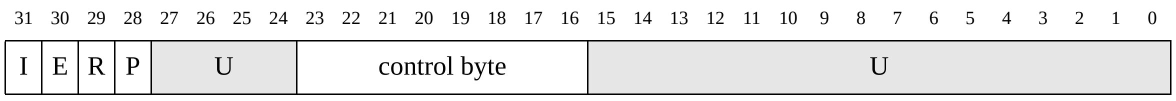


The functions of these fields are described in the table below:

Name	bits	R/W	Function
p2p source ID	15:0	W	16-bit chip source ID for p2p packets
control byte	23:16	W	control byte of next sent packet
U: unused	27:24	-	-
O: Tx overflow	28	R	transmit buffer overflow
T: Tx full	29	R	transmit buffer full
E: int. enable	30	W	enable interrupt by Tx overflow
I: int. status	31	R	enabled interrupt caused by Tx overflow

The p2p source ID is expected to be configured once at start-up. The parity and sequence code fields of the control byte will be replaced by automatically-generated values when the packet is launched. The time stamp (where applicable) will be inserted by the local Router.

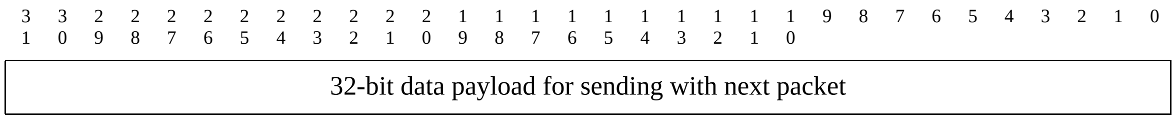
r1: receive status



The functions of these fields are described in the table below:

Name	bits	R/W	Function
U: unused	15:0	-	-
control byte	23:16	R	control byte of last received packet
U: unused	27:24	-	-
P: parity	28	R	received packet parity error (=1)
R: received	29	R	packet received
E: int. enable	30	W	enable interrupt by received packet
I: int. status	31	R	enabled interrupt caused by received packet

r2: send data

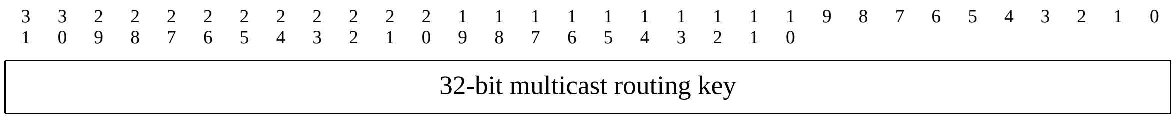


If data is written into r2 before a send key or dest ID is written into r3, the packet initiated by writing to r3 will include the contents of r2 as its data payload. If no data is written into r2 before a send key or dest ID is written into r3 the packet will carry no data payload.

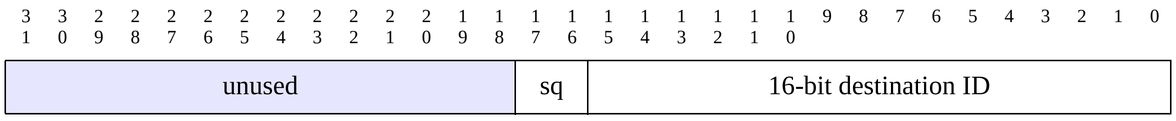
r3: send mc key/p2p dest ID & sequence code

Writing to r3 will cause a packet to be issued (with a data payload if r2 was written previously).

If bits[23:22] of the control register are 00 the Communication Controller is set to send multicast packets and a 32-bit routing key should be written into r3. The 32-bit routing key is used by the associative multicast Routers to deliver the packet to the appropriate destination(s).



If bits[23:22] of the control register are 01 the Communication Controller is set to send point-to-point packets and the value written into r3 should include the 16-bit address of the destination chip in bits[15:0] and a sequence code in bits[17:16]. (See ‘seq code’ on page 16.)



If bits[23:22] of the control register are 10 the Communication Controller is set to send nearest neighbour packets and the 32-bit nn address field should be written in r3.

r4: received data

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32-bit received data payload																															

If a received packet carries a data payload the payload will be delivered here and will remain valid until r5 is read.

r5: received mc key/p2p source ID & sequence code

A received packet will deliver its mc routing key, nn address or p2p source ID and sequence code to r5. For an mc or nn packet this will be the exact value that the sender placed into its r3 for transmission; for a p2p packet the sequence number will be that placed by the sender into its r3, and the 16-bit source ID will be that in the sender's r0.

The register is read sensitive - once read it will change as soon as the next packet arrives.

7.6 Fault-tolerance

Fault insertion

Software can cause the Communications Controller to misbehave in several ways including inserting dodgy routing keys, source IDs, destination IDs.

Do we need to be able to force parity errors in transmit packets?

Fault detection

Parity of received packet?

Fault isolation

The Communications Controller is mission-critical to the local processing subsystem, so if it fails the subsystem should be disabled and isolated.

Reconfiguration

The local processing subsystem is shut down and its functions migrated to another subsystem on this or another chip. It should be possible to recover all of the subsystem state and to migrate it, via the SDRAM, to a functional alternative.

7.7 Test

production test

start-up test

run-time test

7.8 Notes

- time phase accuracy: if we assume that the system time phase is F and the skew is K (that is, all parts of the system transition from one phase to its successor within a time K), then a packet has at least F-K to reach its destination and will be killed after at most 2F+K.
  Thus, if we want to allow for a maximum packet transit time of F-K = T and can achieve a minimum phase skew of K, then T and K are both system constants and we should choose F = T+K. The longest packet life is then 2T+3K.

- transmit flow control needs further thought. Support for polling is already provided, as is transmit buffer overrun interupt, but do we also want hardware stall on send when buffer full?

8. Communications NoC

The communications NoC has the primary role of carrying neural event packets between Fascicle Processors on the same or different chips.

8.1 Features

- On- and inter-chip links
- Router with associative multicast, algorithmic, default and emergency routing functions.
- Arbiter to merge all sources into a sequential packet stream into the Router.
- Individual links can be reset to clear blockages and deadlocks.

8.2 Block diagram

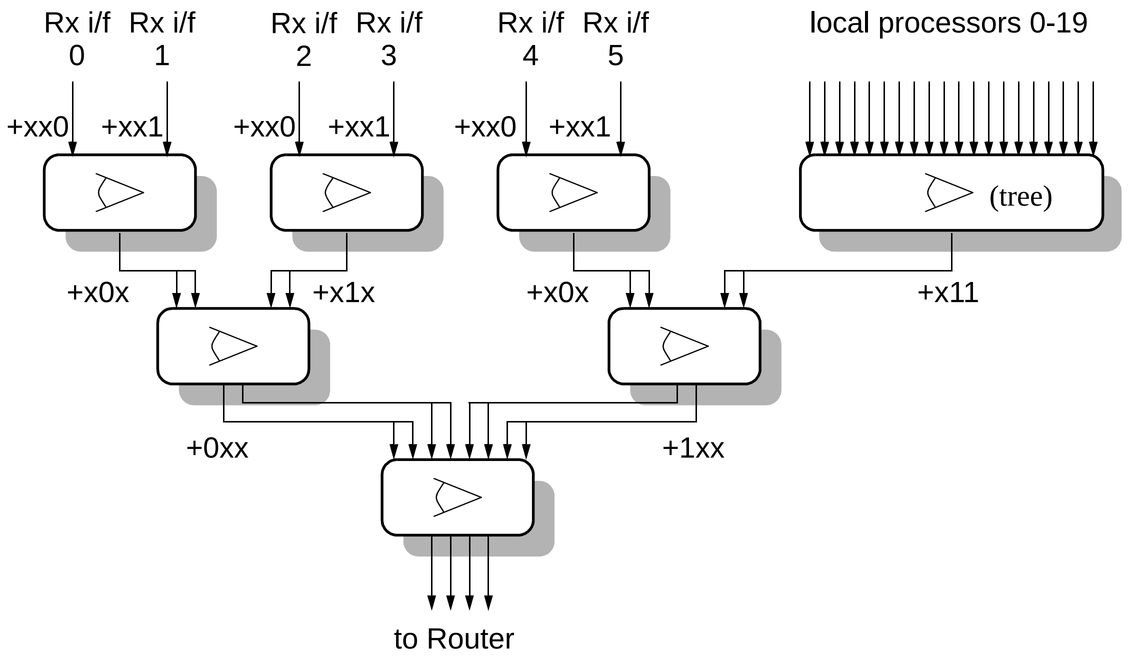
A block diagram of the Communications NoC was given in section 1.1 on page 5.

8.3 Arbiter structure

As the input links converge on the Router they must merge through 2-way CHAIN arbiters, and the link width must increase to absorb the bandwidth. The following hierarchy is proposed:

- the local processor links can all be merged through a single-link arbiter tree as the local bandwidth is low, e.g. at most 20 processors x 1,000 neurons x 100Hz x 40 bits = 80 Mbit/s.
- the Rx interfaces can each carry up to 1 Gbit/s, about half the on-chip single-link bandwidth, so the first layer of arbiters can be single-link, the 2nd layer dual-link and the 3rd layer quad-link (i.e. 8-bits or 48 wires wide).
- buffering is required wherever the link width increases to ensure that the full arbiter bandwidth is used. Each buffer must be at least half a packet long - 36 bits?
- at each arbiter merging Rx interfaces the packet must pick up 1 bit to indicate its source, for default routing [unless the source tagging is done by the Rx interface?]

The Arbiter structure is illustrated below. Each doubling of the wires represents a doubling of the CHAIN link width. The numbers indicate source tagging of the packets.



8.4 Fault-tolerance

Fault insertion

There is little direct control of the Communications NoC fabric except at the periphery as noted in the sections below.

Fault detection

Most failures will cause local asynchronous deadlock, which is readily detected at both the transmitting and receiving ends of the link.

Fault isolation

If links fail their clients will have to be disabled and their functions migrated.

Reconfiguration

Client functional migration is required.

8.5 Test

production test

start-up test

run-time test

8.6 Notes

- must decide whether to add source tags for default routing in arbiters or in Rx interfaces.

9. Communications Router

The Communications Router is responsible for routing all packets that arrive at its input to one or more of its outputs. Its primary function is to route multicast neural event packets, which it does through an associative multicast router subsystem. But it is also responsible for routing point-to-point packets (for which it uses algorithmic routing), for nearest-neighbour routing (which is a simple algorithmic process), for default routing (when a multicast packet does not match any entry in the multicast router) and for emergency routing (when an output link is blocked due to congestion or hardware failure).

Various error conditions are identified and handled by the Communications Router, for example packet parity errors, time-out, and output link failure.

9.1 Features

- *1024* programmable associative multicast routing entries.
  - associative routing based on source 'key'.
  - with flexible "don't care" masking.
- algorithmic routing of point-to-point and nearest-neighbour packets.
- support for 40- and 72-bit multicast, point-to-point and nearest neighbour packets.
- default routing of unmatched multicast packets.
- automatic re-routing around failed links.
- failure detection and handling:
  - packet parity error
  - time-expired packet
  - output link failure
  - corrupt (wrong length) packet

9.2 Description

We assume that messages arrive from other nodes via the link receiver interfaces and from internal clients and are presented to the router one-at-a-time. The Arbiter is responsible for determining the order of presentation of the messages, but as each message is handled independently the order is unimportant (though it is desirable for packets following the same route to stay in order).

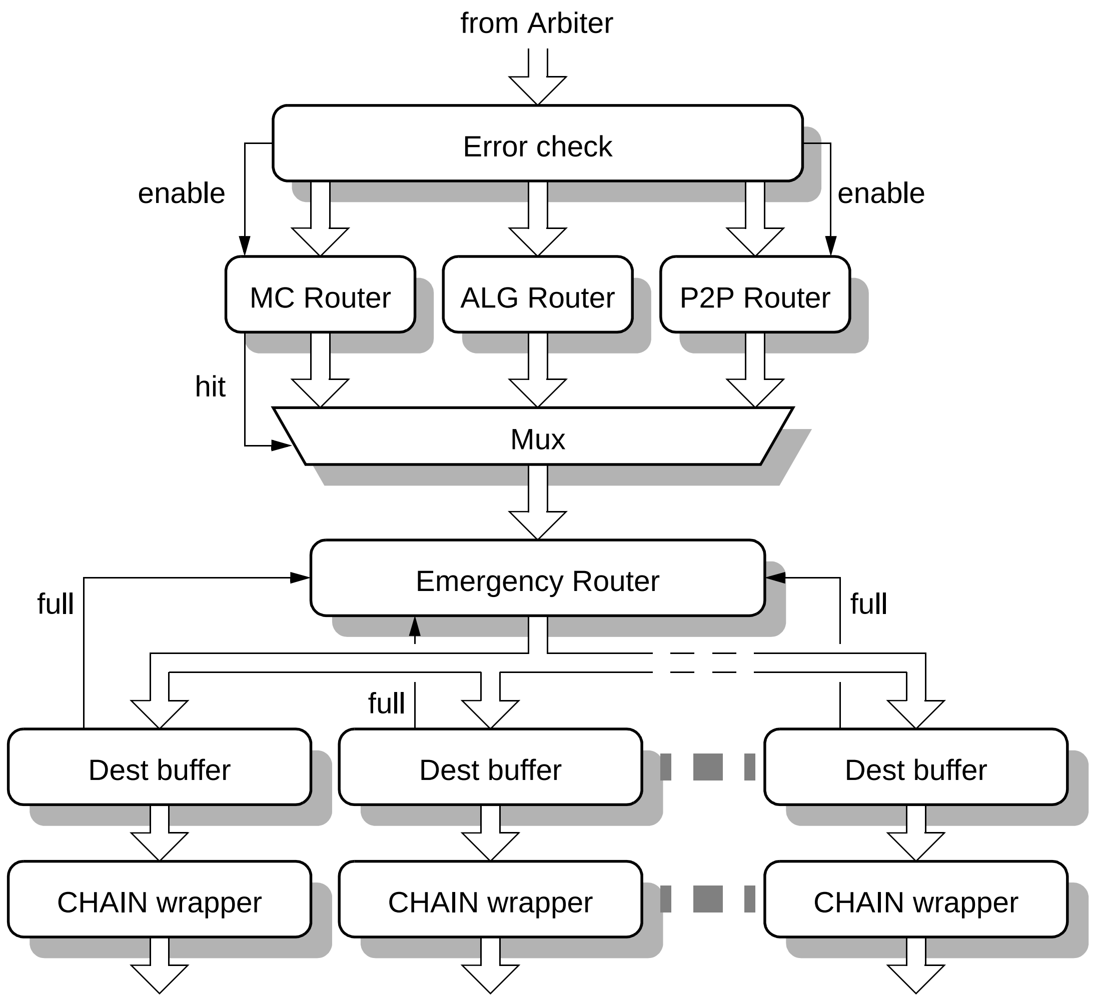
Each message contains an identifier that is used by the Communications Router to determine which of the outputs the message is sent to. These outputs may include any subset of the output links, where the message may be sent via the respective link transmitter interface, and/or any subset of the internal processor nodes, where the message is sent to the respective Communications Controller.

For the neural network application the identifier can be simply a number that uniquely identifies the source of the message – the neuron that generated the message by firing. This is 'source address routing'. In this case the message need contain only this identifier, as a neural spike is an "event" where the only information is that the neuron has fired.

The Router then functions simply as a look-up table where for each identifier it looks up a routing word, where each routing word contains 1 bit for each destination (each link transmitter interface and each local processor) to indicate whether or not the message should be passed to that destination.

9.3 Internal organization

The internal organization of the Communications Router is illustrated in the figure opposite.



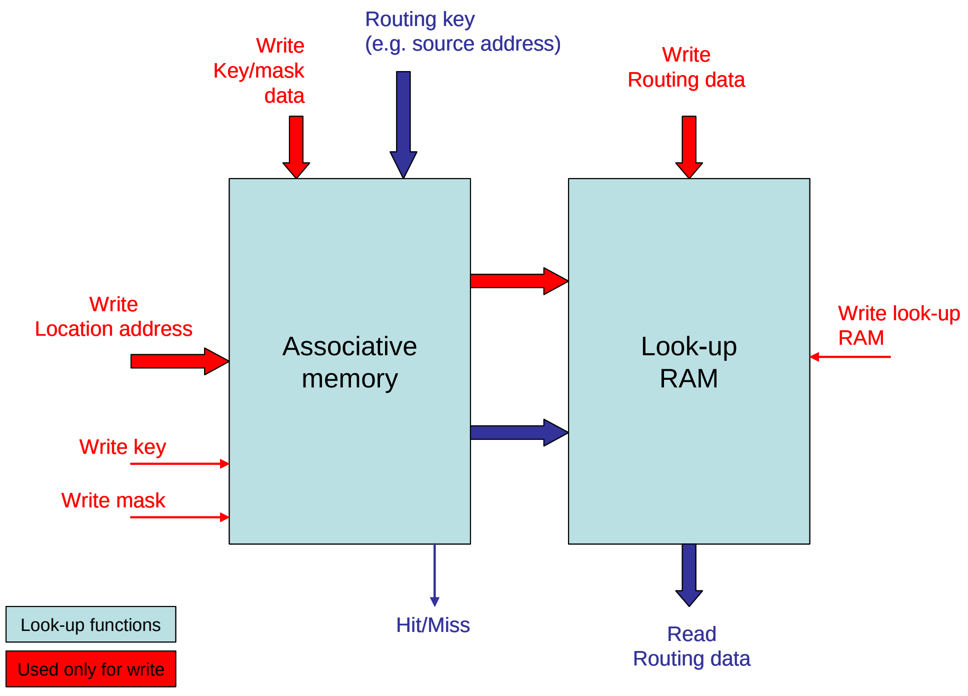
Packets are passed as complete 40- or 72-bit units from the Arbiter, together with an identifier of the Rx interface that the packet arrived through (for nearest-neighbour and default routing). The first stage of processing here is to identify errors. The second stage passes the packet to the appropriate routing engines – the multicast (MC) router is activated only if the packet is error-free and of multicast type, but the algorithmic (ALG) router is activated for every packet as it handles default routing in addition to point-to-point algorithmic and error routing. The output of the router stage is a vector of destinations to which the packet should be relayed. The third stage is the emergency routing mechanism for handling failed or congested links, which it detects using 'full' signals fed back from the individual destination output buffers.

Notes

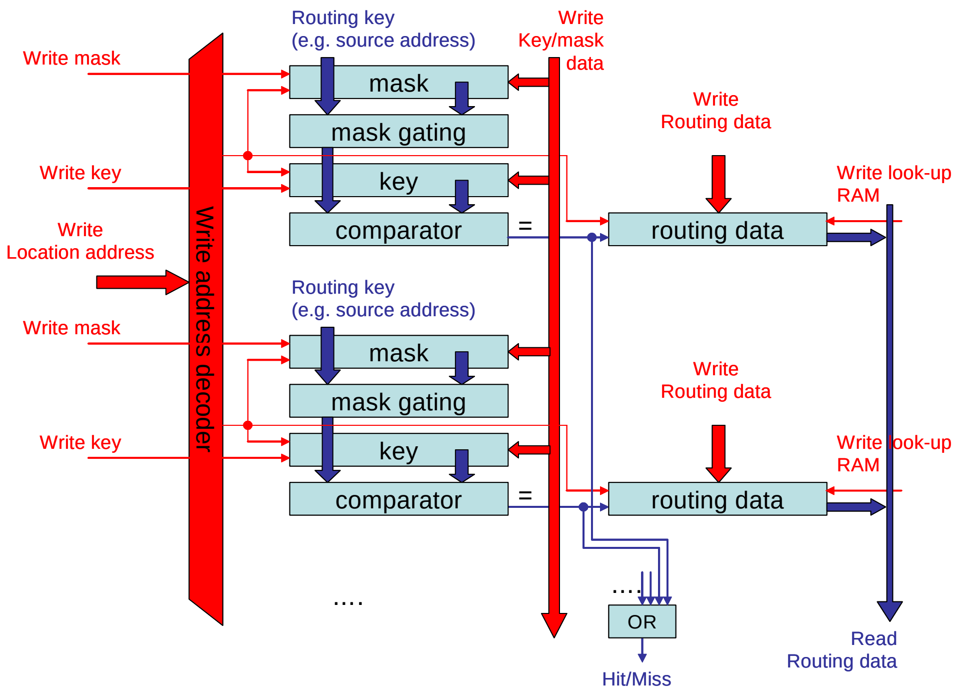
- the Router needs to know which is the Monitor Processor for routing terminating p2p, nn, error, and dropped packets
- how are details of errors communicated to the Monitor Processor?
- Emergency routing may cause two packets to be issued onto the same output link (one for the MC routed data and the second for the alternative route for the blocked link). These are merged into a single packet with a different emergency-routing type to indicate its dual purpose.

9.4 Multicast (MC) router

The internal organisation of the multicast router is illustrated in the figure below.



Implementation



Multicast router optimisations

The simple look-up table as described above works in principle but in practice would be too large to fit onto a chip. However, several optimisations address this problem, reducing the table to a practical size:

- The table within a particular node need contain entries only for message identifiers whose routes from their source to all of their destinations pass through, are generated in, or end in that node.

- Default routing can be supported that passes messages from a link receiver interface to the diametrically opposite link transmitter interface when the message identifier is not found in the look-up table.
- Groups of message identifiers can be routed using the same look-up table entry by making some of the identifier bits “don’t care” as far as the look-up process is concerned.

The logical structure of the Router is then an associative (content-addressed) memory, with programmable masking on a per-entry basis to support the “don’t care” optimization, connected to a conventional memory that holds the per-entry output routing word. The associative memory can be implemented using any of the usual techniques such as VLSI CAM cells or hash-addressed RAM.

Additional mechanisms can support the default routing mentioned above when there is no match for the message identifier in the associative memory and, through partitioning the identifier address space, provision can be made for conventional 1-to-1 destination address routing and broadcast mechanisms.

Illustrative example

By way of illustration, let us assume a neural modelling system where each neuron has a unique 14-bit number, and when it fires it transmits this number prepended by two zero bits as the message identifier (and the message has no other content). Further, we assume that the neurons are handled in groups of 256, where each group of 256 is assigned to a particular processor on a particular node and each neuron in a group has the same set of inputs and the same output destinations as every other neuron in the same group.

The message identifiers can then be processed by the Routers as the 16-bit binary number: 00nnnnnnXXXXXXXXX, where:

- 00 indicates that this is a source address message identifier that should be routed according to the routing table;
- nnnnnn is the neuron group identifier, so at most $2^6 = 64$ routing look-up table entries are required;
- XXXXXXXX indicates that the bottom 8 bits of the message identifier can be treated as “don’t care” and play no role in the routing. They will be used only by the destination processor to identify the neuron that fired within the group.

Messages beginning with 01, 10, and 11 may be used for 1-to-1 destination address routing, broadcast, and some other purpose respectively, using conventional routing algorithms.

It can be seen from this example that the optimisations are a vital aspect of the invention, reducing the size of the look-up table in this very small example from 16,384 (= 2^{14}) to at most 64 entries. For larger systems the benefits of the optimisations are likely to be even more significant.

The “don’t care” bits are programmable independently for each look-up table entry on each node, and they can be distributed anywhere across the message identifier for maximum flexibility, so a single look-up table entry may route several groups together at one node, whereas at the next node they may be routed independently to different destinations.

Route set up

The routing look-up tables may be configured using external software that takes a neural “netlist”, describing the way the neurons interconnect, and then maps the neurons onto processors and determines the routing table values using algorithms similar to those used to configure an FPGA. As with FPGA configuration, resource constraints such as the routing table size and link bandwidth limitations must be taken into account during the mapping process.

The routing table configuration is then loaded into the local Router by a local processor that follows instructions from a control system using the 1-to-1 message routing mechanism.

For static neural network modelling the routing is fixed after initialisation. It is possible to allow local processors to modify the routing tables while the system is running, if this is required to model developmental processes (for example), provided this is done with due care.

9.5 The point-to-point (p2p) router

The p2p router uses the 16-bit destination ID in a point-to-point packet to determine which output(s) the packet should be routed to. A 64K entry x 8-bit SRAM lookup table directs the p2p packet to:

- the local Monitor Processor, and/or
- adjacent chips via the appropriate links.

Each 8-bit entry has one bit which determines whether the packet is delivered to the local Monitor Processor, one bit for each of the six output links, plus a parity bit. Thus there is a form of broadcast capability available here.

9.6 The algorithmic (ALG) router

nn routing

Nearest-neighbour packets are used to initialise the system and to perform run-time flood-fill and debug functions. The routing function here is to send ‘normal’ nn packets that arrive from outside the node (i.e. via an Rx link) to the monitor processor and to send nn packets that are generated internally to the appropriate output (Tx) link(s). This is to support a flood-fill OS load process.

In addition, the ‘direct’ form of nn packet can be used by neighbouring systems to access System NoC resources. Here an nn ‘write’ packet (which is a direct type with a 32-bit payload) is used to write the 32-bit data defined in the payload to a 32-bit address defined in the address/operation field. An nn ‘read’ packet (which is a direct type without a 32-bit payload) uses the 32-bit address defined in the address/operation field to read from the System NoC and returns the result (as a ‘normal’ nn packet) to the neighbour that issued the original packet using the Rx link ID to identify that source. This ‘direct’ access to a neighbouring chip’s principal resources can be used to investigate a non-functional chip, to re-assign the Monitor Processor from outside, and generally to get good visibility into a chip for test and debug purposes.

default and error routing

In addition, the algorithmic router performs default and error routing functions.

9.7 Time phase handling

The Router maintains a 2-bit time phase signal that is used to delete packets that are out-of date. The time phase logic operates as follows:

- locally-generated packets will have the current time phase inserted (where appropriate);
- a packet arriving from off-chip will have its time phase checked, and if it is two phases old it will be deleted (dropped to the local Monitor Processor).

9.8 Packet error handler

The packet error handler is a routing engine that simply flags the packet for routing to the local Monitor Processor if it detects any of the following:

- a packet parity error;
- a packet that is two time phases old;
- a packet that is the wrong length.

There must be a means for the Monitor Processor to recognise packets passed to it with errors. Rather than complicating the Communications Controller, this is probably better done by providing error information via the Router configuration registers.

9.9 Emergency routing

If a link fails (temporarily, due to congestion, or permanently, due to component failure) action will be taken at two levels:

- the blocked link will be detected in hardware and subsequent packets rerouted via the other two sides of one of the routing triangles of which the suspect link was an edge.
- the Monitor Processor will be informed. It will assess the problem, and take appropriate action:
  - if the problem was due to transient congestion, it will note the congestion but do nothing further;
  - if the problem was due to recurring congestion, it will negotiate and establish a new route for some of the traffic using this link;
  - if the problem appears permanent, it will reset the link (incurring some packet loss) and then, if this does not clear the problem, negotiate and establish new routes for all of the traffic using this link.

The hardware support for these processes include:

- default routing processes in adjacent nodes that are invoked by flagging the packet as an emergency type;
- mechanisms to inform the monitor processor of the problem;
- mechanisms the monitor processor can use to reset the link;
- means of inducing the various types of fault for testing purposes.

Emergency rerouting around the triangle requires additional emergency packet types for mc packets. p2p packets will find their own way to their destination following emergency routing.

9.10 Errant packets

In order to ensure that packets cannot circulate for ever within the system each packet includes a time phase field. This is set when the packet is launched, and if a packet arrives at a Router two time phases after it was launched it will be routed directly (and only) to the local monitor processor for error-handling purposes.

9.11 Pseudo-code description

The following pseudo-code describes the detailed operation of the Communications Router:

```
Pipeline stage 1: Error Checking
inputs:       72-bit Packet    p;
              3-bit  SourceID src;
local info:   2-bit TimePhase timePhase

Begin stage 1=========================================

% check error conditions

PPerr = (packetParity(p)  == EVEN);           % parity error
TPerr = (src < 6) AND (p.timeStamp = timePhase EOR 0b11)
              AND (p.type == 0b0x);            % time phase error
LNerr = (p.lastSymbol != EOP);                % packet length error

error = PPerr OR TPerr OR LNerr;

% update counters

if (PPerr) incPacketParityErrorCounter();
if (TPerr) incPacketTimeStampErrorCounter();
if (LNerr) incPacketLengthErrorCounter();

incPacketCounter();
```

```
% insert Time Phase

if (src == 7) AND (p.type == 0b0x) {          % local p2p or mc packet
  p.timeStamp = timePhase;
  ParityFix(p.parity);

% engage appropriate Router

enMC = (not error) AND (p.type == MC) AND (p.emergencyRouting == 0b0x);
enP2P = (not error) AND (p.type == P2P);

End stage 1==========================================

Pipeline stage 2: Routing
inputs:      72-bit Packet   p;
             3-bit  SourceID src;
             Booleans PPerr, TPerr, LNerr, error;
local info: 5-bit MonitorProcessorID mpID;

Begin stage 2========================================

% enable relevant Router

if (enMC) {hit, MCvect} = MCrouter(p.MCkey);
in (enP2P) {P2Pvect}    = P2Prouter(p.destID);

% default emergency routing vector

erVect = 0;

% send all errors to Monitor Processor

if (error)              vect = 2^(mpID+6);
else {

% routing depends on packet type

case (p.type) {

MC:          if (hit) vect = MCvect;
             else if (p.emergencyRouting == 0b0x)
                      vect = 2^[(src+3)mod6];    % normal default
             else     vect = 0;                  % ER only
             if (p.emergencyRouting == 0b01 or 0b10)
                      erVect = 2^[(src-1)mod6]; % ER 1st stage
             else if (p.emergencyRouting == 0b11)
                      erVect = 2^[(src+2)mod6]; % ER 2nd stage

P2P:                  vect = P2Pvect;

NN:          if (src < 6) {                      % local source
               if (p.route == 7)                 % local MP
                      vect = 2^(mpID+6);
               else if (p.route == 6)            % all neighbours
                      vect = 0b000000000000000000111111;
               else   vect = 2^(packet.route);   % one neighbour
             } else {                            % external source
               if (p.T) {                        % direct NN
                 if (p.data) write.SystemNoC(p.address,p.data);
                      vect = 0;                  % packet goes nowhere
                 } else {
                 p.payload = read.SystemNoC(p.address);
                 p.data    = TRUE;
                 p.T       = 0;                  % change to normal
                      vect = 2^src;              % return to sender!
               } else vect = 2^(mpID+6);         % normal NN
             }
} % end case
} % end else (¬error)
```

```
End stage 2==========================================

Pipeline stage 3: Emergency Routing
inputs:       72-bit Packet    p;
              26-bit Vector    vect;
              6-bit  ERvector erVect;
local info: Booleans buffFull bFull[0..25];
              5-bit MonitorProcessorID mpID;


Begin stage 3========================================

% check for output contention & wait fixed max time to resolve

clockCycles = 0;
do {
  blocked = FALSE;
  for (i = 0; i++; i<26) {
    if (bFull[i] AND (vect.bit[i] OR erVect.bit[i]) blocked = TRUE;
  }
  clockCycles++;
} while (blocked AND (clockCycles < MaxWaitBeforeER));

% now look into Emergency Routing options & wait fixed max time

clockCycles = 0;
do {
  blocked = FALSE;
  for (i = 0; i++; i<26) {
    if ((bFull[i] AND (vect.bit[i] OR erVect.bit[i]))
        AND ((i>5) OR bFull[(i-1)mod6]
        OR (p.emergencyRouting != 0b00))) blocked = TRUE;
  }
  clockCycles++;
} while (blocked AND (clockCycles < MaxWaitForER));

% if Emergency Routing has failed...

if (blocked) {
  sendPacketTo(p, buff[mpID+6]);       % send to Monitor Proc
  incDroppedPacketCounter();           % record packet loss
} else {

% can now proceed

  for (i = 0; i++; i<26) {
    p2 = p;                                    % copy packet
    if (vect.bit[i] OR erVect.bit[i]
       OR ((i<6) AND bFull[(i+1)mod6] AND vect.bit[(i+1)mod6]
           AND (p.emergencyRouting == 0b00))) {
      if (erVect.bit[i]) {
        if (p.emergencyRouting == 0b11)
             p2.emergencyRouting = 0b00;      % ER finished
        else p2.emergencyRouting = 0b11;      % ER 2nd stage
      } else if (i<6) {
        if (bFull[(i+1)mod6] AND vect.bit[(i+1)mod6]
           AND (p.emergencyRouting == 0b00))
             p2.emergencyRouting = 0b01;      % ER 1st stage
           incERpacketCounter();              % record ER
        else p2.emergencyRouting = 0b00;
        if (¬vect.bit[i])
             p2.emergencyRouting = 0b10;      % ER only
      } else p2.emergencyRouting = 0b00;
      ParityFix(p2.parity);
      sendPacketTo(p2, buff[i]);
    }
  }
}

End stage 3==========================================
    outputs:     72-bit PacketBuffer buff[0..25]
```

9.12 Fault-tolerance

The Communications Router has limited fault-tolerance capacity, mainly coming down to mapping out a failed multicast router entry. This is a useful mechanism as the multicast router dominates the silicon area of the Communications Router.

Fault insertion

- enable Router to flip packet parity bits?

Fault detection

- packet parity errors
- packet time-phase errors
- packet unroutable errors (e.g. a locally-sourced multicast packet which doesn’t match any entry in the multicast router).
- wrong packet length.

Fault isolation

- a mechanism is required to disable a multicast router entry if it fails. Possible just an ‘entry valid’ bit?

Reconfiguration

- since all multicast router entries are identical the function of any entry can be relocated to a spare entry (within the same segment of the router if segmentation is used to save power).
- if a router (segment) becomes full a global reallocation of resources can move functionality to a different router (segment)

9.13 Test

production test

start-up test

run-time test

9.14 Notes

- The Router will require a number of traffic monitor features, e.g. packet counters, congestion indicators, count packet under match & mask, dropped packet count, emergency routing count, count on each output link, ...

10. Inter-chip transmit and receive interfaces

Inter-chip communication is implemented by extending CHAIN links from chip to chip. In order to sustain CHAIN link throughput, there is a protocol conversion at each chip boundary from standard CHAIN 1-of-5 (including EOP) return-to-zero to 2-of-7 non-return-to-zero. Each conversion maps two 2-bit CHAIN symbols to a single 4-bit 2-of-7 symbol.

10.1 Features

- transmit (Tx) interface:
  - converts two on-chip 1-of-5 RTZ symbols into one off-chip 2-of-7 NRZ symbol;
  - control input to induce a fault;
  - failure detection output.
  - fault reset input.
- receive (Rx) interface:
  - converts one off-chip 2-of-7 NRZ symbol into two on-chip 1-of-5 RTZ symbols;
  - control input to induce a fault;
  - failure detection output.
  - fault reset input.
  - adds source tag to packet for default routing [unless this is done in the Communications NoC arbiter?]

10.2 Programmer view

There are no programmer-accessible features implemented in these interfaces. In normal operation these interfaces provide transparent connectivity between the routing network on one chip and those on its neighbours.

10.3 Fault-tolerance

The fault inducing, detecting and resetting functions are controlled from the System Controller (see 'System Controller' on page 36).

Fault insertion

- an input controlled by the System Controller causes the interface to deadlock

Fault detection

- an output to the System Controller indicates deadlock

Fault isolation

- the interface can be disabled to isolate the chip-to-chip link. This may be the same input from the System Controller that is used to insert a fault.

Reconfiguration

- the link interface can be reset by the System Controller to attempt recovery from a fault
- the link interface can be isolated and an alternative route used

10.4 Test

production test

start-up test

run-time test

11. System NoC

The System NoC has a primary function of connecting the Fascicle Processors to the SDRAM interface. It is also used to connect the Monitor Processor to system control and test functions, and for a variety of other purposes.

11.1 Features

- supports full bandwidth block transfers between the SDRAM and the Fascicle Processors.
- the Router is an additional initiator for system debug purposes.
- can be reset (in subsections?) to clear deadlocks.

11.2 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

11.3 Test

production test

start-up test

run-time test

12. SDRAM interface

The SDRAM interface connects the System NoC to an off-chip SDRAM device. It will be the ARM PL340.

12.1 Features

- lots of bandwidth, please!

12.2 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

12.3 Test

production test

start-up test

run-time test

13. System Controller

The System Controller incorporates a number of functions used by the Monitor Processor for system start-up, fault-tolerance testing (invoking, detecting and resetting faults), general performance monitoring, and such like.

13.1 Features

- lots of fiddly bits
- input & output ports to connect to the communications NoC Tx & Rx interface fault invocation, detection and reset functions.

13.2 Register summary

Name	Offset	R/W	Function

13.3 Register details

	7	6	5	4	3	2	1	0
reset to:	0	0	0	0	0	0	0	0

13.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

13.5 Test

production test

start-up test

run-time test

14. Router configuration registers

The Router is highly configurable, and the Monitor Processor is responsible for initialising it and updating it when necessary. The Router configuration registers are accessed via the system NoC.

14.1 Features

- used to set up the associative routing tables.
- give read/write access to the Router tables for test purposes.
- access for Monitor Processor to Router packet error information and traffic counters.

14.2 Register summary

Name	Offset	R/W	Function

14.3 Register details

	7	6	5	4	3	2	1	0
reset to:	0	0	0	0	0	0	0	0

14.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

14.5 Test

production test

start-up test

run-time test

15. System RAM

The System RAM is an additional *128* kByte block of on-chip RAM used primarily by the Monitor Processor to enhance its program and data memory resources as it will be running more complex (though less time-critical) algorithms than the Fascicle Processors.

As the choice of Monitor Processor is made at start-up (and may change during run-time for fault-tolerance purposes) the System RAM is made available to whichever processor is Monitor Processor via the System NoC. It is probably important that accesses by the Monitor Processor to the System RAM are non-blocking as far as SDRAM accesses by the Fascicle Processors are concerned, so the System NoC should ensure this is the case.

The System RAM may also be used by the Fascicle Processors to communicate with the Monitor Processor and with each other, should the need arise.

15.1 Features

- *128* kB of SRAM, available via the System NoC.
- can be disabled to model complete failure for fault-tolerance testing.
- can we include parity or ECC to improve fault-tolerance?

15.2 Fault-tolerance

Fault insertion

- It is straightforward to corrupt the contents of the System RAM to model a soft error – any processor can do this. It is not clear how this would be detected.
- The System RAM can be disabled to model a total failure.

Fault detection

- The Monitor Processor may perform a System RAM test at start-up, and periodically thereafter.
- It is not clear how soft errors can be detected without some sort of parity or ECC system.

Fault isolation

- Faulty words in the System SRAM can be mapped out of use.

Reconfiguration

- For hard failure of a single bit, avoid using the word containing the failed bit.
- If the System RAM fails completely the only option is to use the SDRAM instead, which will probably result in compromised performance for the Fascicle Processors due to loss of SDRAM bandwidth. An option then would be to relocate some of the Fascicle Processors' workload to another chip.

15.3 Test

production test

- run standard memory test patterns from one of the processing subsystems.

start-up test

run-time test

16. Boot ROM

16.1 Features

- a small on-chip ROM to provide minimal support for:
- initial self-test, and Monitor Processor selection
- Router initialisation for bootstrapping
- system boot.

16.2 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

16.3 Test

production test

start-up test

run-time test

17. Boot, test and debug support

17.1 Features

- means of booting system and flood-filling the distributed operating system efficiently at start-up.
- back-up boot mechanism in case Boot ROM fails.
- access to ARM968 EmbeddedICE features.
- sundry features to facilitate production, start-up and run-time testing.

17.2 Issues

At system power-up we can make few assumptions about what is and isn't working within the system. What is the minimum that must work for each chip to run internal self-tests, appoint a Monitor Processor, and then participate with its peers in an efficient bootstrap process that loads a distributed operating system into every node?

The inter-chip communication system is very soft and must be initialised before any mc or p2p communication can take place. But each node has no initial knowledge of where it is in the system, so how can it initialise the Router?

The ultimate system is large, so the bootstrap process must be efficient and employ flood-fill algorithms.

17.3 Boot algorithm

- Following power-on reset, each chip will perform internal self-tests and a Monitor Processor will be selected, probably as a result of asynchronous arbitration processes. The node will go into receptive mode, relying on the default boot routing process to communicate.
- The host system will begin sending OS load packets in nearest-neighbour format, tagged with sequence numbers, to the node to which it is directly connected. All nodes receive all incoming nn packets and, if they have not been seen before, retransmit them to all neighbours. Any packet which has been seen before will be dropped and not retransmitted.
- Once all sequence numbers have been received a node will perform a CRC check and, if this is correct, begin executing the loaded OS code.

17.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

17.5 Test

production test

start-up test

run-time test

18. Input and Output signals

18.1 External SDRAM interface

Signal	Type	Function

18.2

19. Area estimates

We are targetting a UMC 130nm process 10mm x 10mm die. (Europractice runs on this process are multiples of 5mm x 5mm. The test chip will be 5mm x 5mm.)

Assumptions

- RAM is around 2µm^2/bit = 3M T/mm^2.
- logic is 0.2 x the density of RAM = 100k gates/mm^2.
- The pad ring occupies 0.25 mm all round the chip, so the core is 9.5 x 9.5 = 90.25 mm^2.

Using these assumptions we total up the core logic area as follows:

- The processor nodes = 20 x 3.8 = 76 mm^2.
  - An ARM968 with *32* kByte I-RAM and *64* kByte D-RAM is 3.5 mm^2.
  - DMA, interrupt, counter/timer, communications controllers: 20 k gates = 0.2 mm^2.
  - Communications and Systems NoC interfaces = 0.1 mm^2.
- The Communications NoC = 9.7 mm^2.
  - The associative router with *1024* associative entries is ~ 9mm^2.
  - The algorithmic router with 64k x 3 entries is 0.2 mm^2.
  - The Arbiter is small ~ 0.1 mm^2.
  - The Tx and Rx interfaces are small: altogether ~ 0.2 mm^2.
  - Communications network fabric ~ 0.2 mm^2.
- The Systems NoC = 4 mm^2.
  - The *128* kByte System RAM is 2 mm^2.
  - The Boot ROM is small ~ 0.2 mm^2.
  - The System Controller with 20k gates is 0.2 mm^2.
  - The SDRAM controller with 60k gates is 0.6 mm^2.
  - The network fabric is ~ 100kgates = 1 mm^2.
- Boot, test and debug = 0.5 mm^2.

Total area

The total core logic area is thus 76 + 9.7 + 4 + 0.5 = 90.2 mm^2.

Notes

Associative Router = *1024* x 96 latches + 96 gates = 500k gates = 5 mm^2?

20. Power estimates

Processor

ARM968 (from ARM web site) consumes 0.12 to 0.23 mW/MHz on a 130 nm process, and delivers 1.1 dhrystone MIPS/MHz. Thus, to a good approximation, its power-efficiency is 5,000 to 10,000 MIPS/W and it uses 100-200 pJ/instruction.

neuron dynamics

30 instructions at 1 kHz = 30 kIPS = 3-6 μW.

connection processing

1,000 inputs at 10 Hz (ave.) and 10 instructions/input = 100 kIPS = 10-20μW.

SDRAM access

assume SDRAM uses 250mW at 1 Gbyte/s; accessing 4 bytes costs 1 nJ.

1,000 inputs at 10 Hz (ave.) = 40 kByte/s = 10 μW.

communications link

2.5V I/Os, 10 pF/wire = 30 pJ/transition

3 transitions/4 bits + EOP = 33 transitions/spike = 1nJ/spike/link.

Router

assume power budget at full throughput of 200 MHz is 200 mW, so 1 nJ/route.

neuron total

at 10 Hz (ave.), with H hops, power = 3-6 + 10-20 + 10 + (1 + 2H)10^{-3} μW

= 23-36 μW (routing & inter-chip hops are negligible).

Chip

20 processors x 1,000 neurons/processor x 13-26 μW = 260-520 mW.

Node

chip + SDRAM = 460-720 mW.

System

1 billion neurons = 50,000 nodes = 23-36 kW.