MANCHESTER
1824

The University of Manchester

version 0.9 - DRAFT
27 August 2008

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;
  - vectored 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.
- Ethernet interface for host connection
- 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. In the standard model, 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

The details in this datasheet are presented in good faith but no liability can be accepted for errors or inaccuracies. The design of a complex chip multiprocessor is a research activity where there are many uncertainties to be faced, and there is no guarantee that a SpiNNaker system will perform in accordance with the specifications presented here.

The APT group in the School of Computer Science at the University of Manchester was responsible for all of the architectural and logic design of the SpiNNaker chip, with the exception of synthesizable components supplied by ARM Limited. All design verification was also carried out by the APT group. As such the industrial project partners bear no responsibility for the correct functioning of the device.

Change history

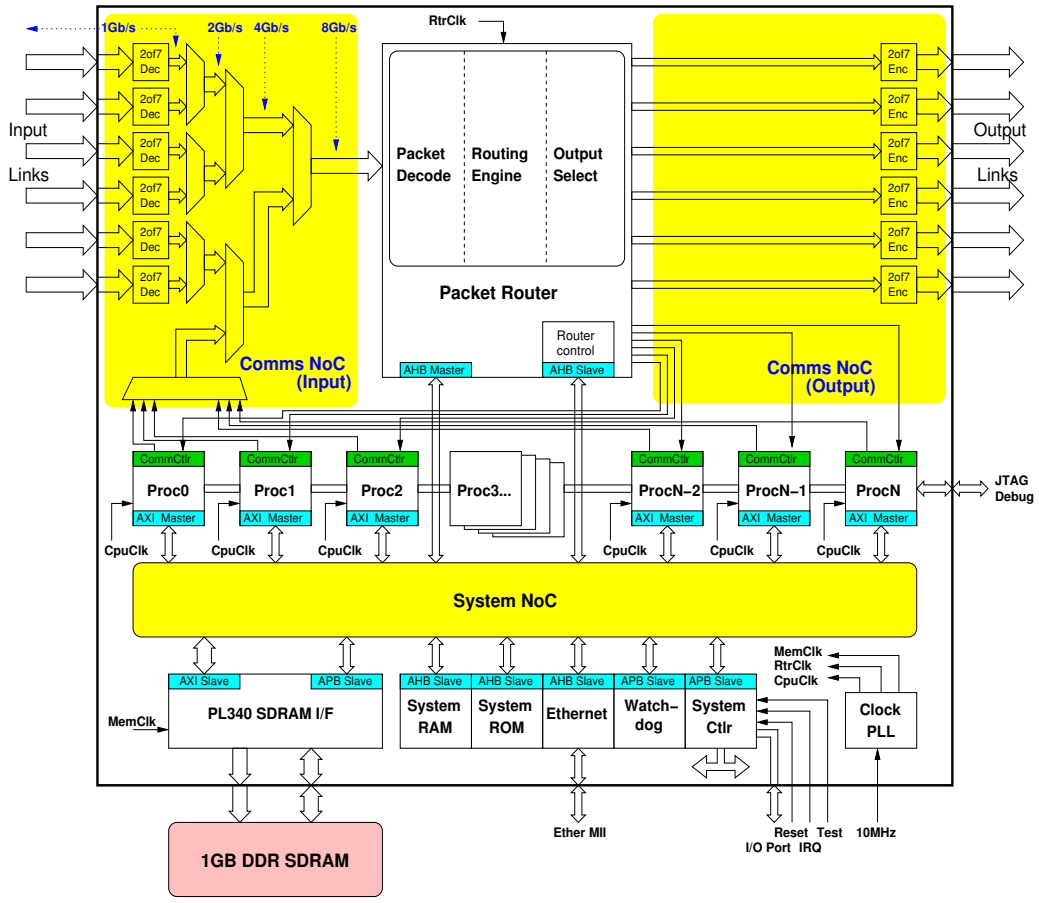
version	date	changes
0.0	27/12/05	First draft.
0.1	16/8/06	Sundry - document still developing.
0.2	18/11/06	Comms controller and NN protocol details modified.
0.3	19/02/07	Added ARM968 memory map, updated router pseudo-code, expanded system controller spec.
0.4	23/04/07	Added DMA controller, updated system controller to use ADK watchdog, updated comms controller.
0.5	05/11/07	Updated area estimates, added test chip details, added pin-out detail, added Ethernet MII interface, updated system diagrams.
0.6	04/12/07	Added details of timer/counter, vectored interrupt controller, PL340; removed Router implementation detail;
0.7	6/2/08	Sundry updates - added watchdog timer
0.8	26/4/08	Memory map changes; Router updates
0.9	27/8/08	Many detailed changes

Contents

1. Chip organization

1.1 Block diagram

The primary functional components of SpiNNaker are illustrated in the figure below.



Each chip contains *20* identical processing subsystems. Following self-test, at start-up one of the processors is nominated as the Monitor Processor and thereafter performs system management tasks. Each of the remaining processors is responsible for modelling one or more neuron fascicles - a fascicle being a group of neurons with associated inputs and outputs (although some processors may be reserved as spares for fault-tolerance purposes).

The Router is responsible for routing neural event packets both between the on-chip 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 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

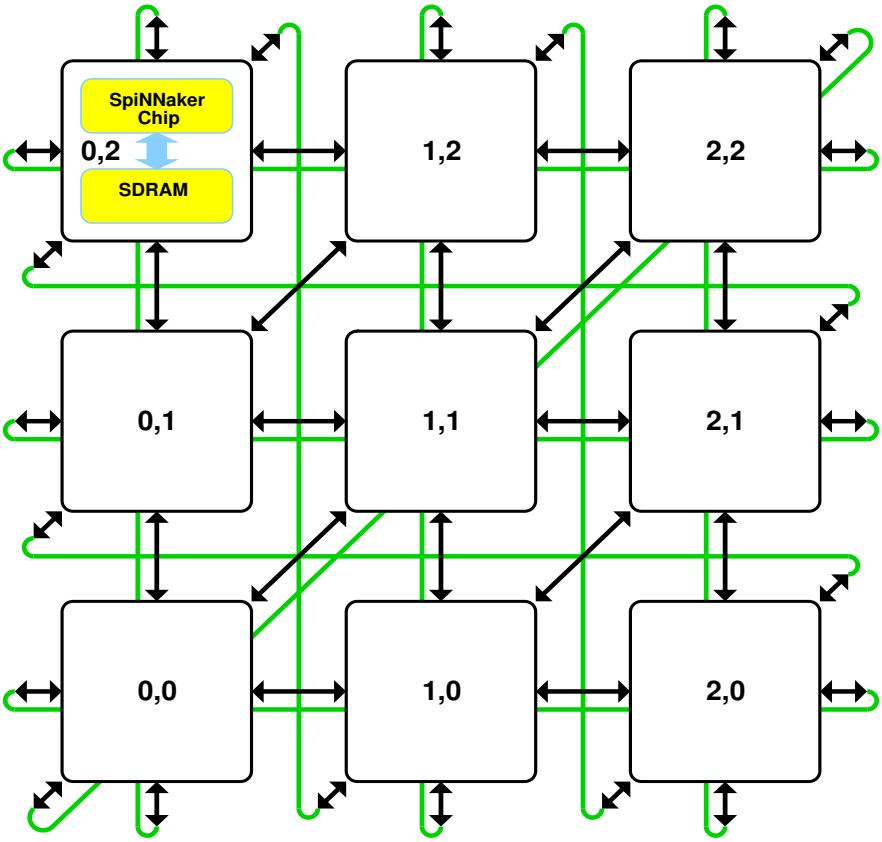
The SpiNNaker chip is viewed as having the following structural hierarchy, which is reflected

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 / System NoC interface
  - communications controller, including communications NoC interface
- Communications NoC
  - Router, including multicast, algorithmic, nearest-neighbour, 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
  - Ethernet MII interface
  - 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 a Router register, and the Monitor Processor informed;

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 and an error flagged to the local Monitor Processor. 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.

Firefly synchronization

The local time phase, used for errant packet trapping as described above, is maintained across the system by a combination of local slightly randomized timers and local phase-locking using nearest-neighbour communication.

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 a 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).

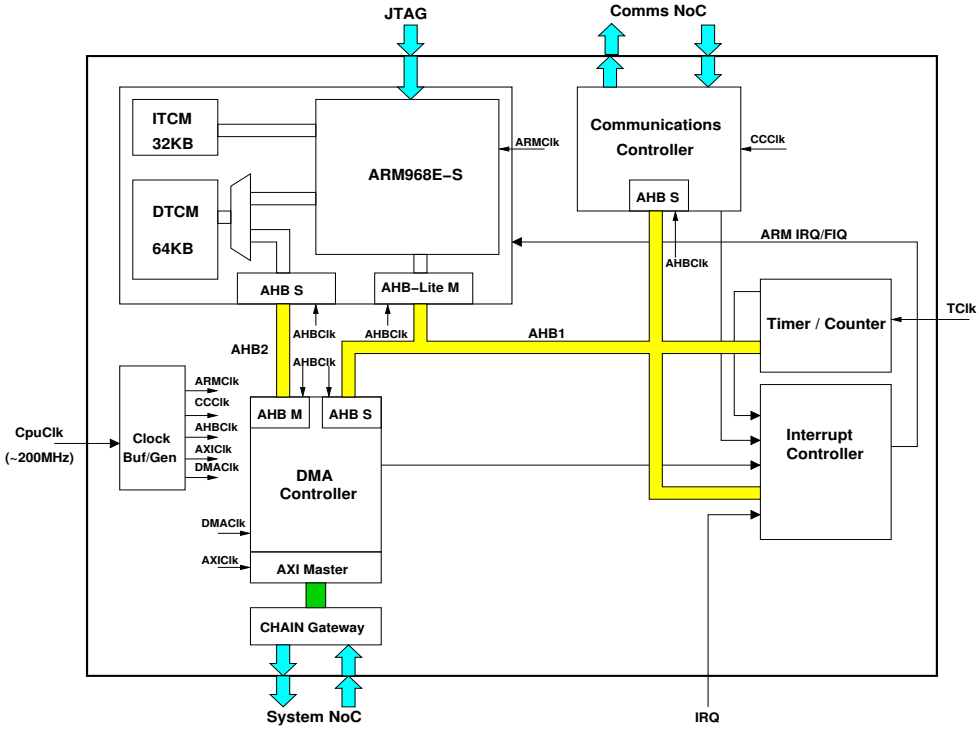
3. ARM968 processing subsystem

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
  - DMA controller & interface to the System NoC
  - timer/counter and interrupt controller

3.2 ARM968 subsystem organisation



3.3 Memory Map

The memory map of the ARM968 spans a number of devices and buses. The tightly coupled memories are directly connected to the processor and accessible at the processor clock speed. All other parts of the memory map are visible via the AHB master interface. This gives direct access to the registers of the DMA controller, communications controller and the timer/interrupt controller. In addition, a path is available through the DMA controller onto the System NoC which provides processor access to all memory resources on the System NoC. The memory map is defined as follows:

```
// ARM968 local memories
#define ITCM_START_ADDRESS          0x00000000 // instruction memory
#define DTCM_START_ADDRESS          0x00400000 // data memory

// 8MB address area (NNOP) reserved for 'operation mapping' in the NN protocol
#define NNOP_START_ADDRESS          0x0f800000
#define NNOP_END_ADDRESS            0x0fffffff

// Local peripherals - unbuffered write
#define COMM_CTL_START_ADDRESS_U    0x10000000 // Communications Controller
#define CTR_TIM_START_ADDRESS_U     0x11000000 // Counter-Timer
#define VIC_START_ADDRESS_U         0x1f000000 // vectored interrupt controller

// Local peripherals - buffered write
#define COMM_CTL_START_ADDRESS_B    0x20000000 // Communications Controller
#define CTR_TIM_START_ADDRESS_B     0x21000000 // Counter-Timer
#define VIC_START_ADDRESS_B         0x2f000000 // vectored interrupt controller

// DMA controller
#define DMA_CTL_START_ADDRESS_U     0x30000000 // DMA controller - unbuffered
#define DMA_CTL_START_ADDRESS_B     0x40000000 // DMA controller - buffered

// SDRAM
#define SDRAM_START_ADDRESS_U       0x50000000 // SDRAM - unbuffered
#define SDRAM_START_ADDRESS_B       0x60000000 // SDRAM - buffered

// System NoC peripherals - buffered write
#define PL340_APB_START_ADDRESS_B   0xe0000000 // PL340 APB port
#define RTR_CONFIG_START_ADDRESS_B  0xe1000000 // Router configuration
#define SYS_CTL_START_ADDRESS_B     0xe2000000 // System Controller
#define WATCHDOG_START_ADDRESS_B    0xe3000000 // Watchdog Timer
#define ETH_CTL_START_ADDRESS_B     0xe4000000 // Ethernet Controller
#define SYS_RAM_START_ADDRESS_B     0xe5000000 // System RAM
#define SYS_ROM_START_ADDRESS_B     0xe6000000 // System ROM

// System NoC peripherals - unbuffered write
#define PL340_APB_START_ADDRESS_U   0xf0000000 // PL340 APB port
#define RTR_CONFIG_START_ADDRESS_U  0xf1000000 // Router configuration
#define SYS_CTL_START_ADDRESS_U     0xf2000000 // System Controller
#define WATCHDOG_START_ADDRESS_U    0xf3000000 // Watchdog Timer
#define ETH_CTL_START_ADDRESS_U     0xf4000000 // Ethernet Controller
#define SYS_RAM_START_ADDRESS_U     0xf5000000 // System RAM
#define SYS_ROM_START_ADDRESS_U     0xf6000000 // System ROM

// Boot area and VIC
#define BOOT_START_ADDRESS          0xff000000 // Boot area
#define HI_VECTORS                  0xffff0000 // high vectors (for boot)
#define VIC_START_ADDRESS_H         0xfffff000 // vectored interrupt controller
```

The areas shown against a yellow background are accessible only by their local ARM968 processor, not by a DMA controller nor by Nearest Neighbour packets via the Router.

The ARM968 should be configured to use high vectors after reset (to use the vectors in the Boot area), but then switched to low vectors once the ITCM is enabled and initialised.

The vectored interrupt controller (VIC) has to be at 0xfffff000 to enable efficient access to its vector registers.

All other peripherals start at a base address that can be formed with a single MOV immediate instruction.

3.4 Fault-tolerance

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

3.5 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 could be protected by parity bits (not implemented).
- A chip-wide watchdog timer catches 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. Vectored interrupt controller

Each processor node on an SpiNNaker chip has a local vectored interrupt controller (VIC) 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 IRQ vectoring purposes.

The VIC is the ARM PL190, described in ARM DDI 0181E.

5.1 Features

- manages the various interrupt sources to each local processor
- individual interrupt enables
- routing to FIQ and/or IRQ (there will normally be only one FIQ source: CC Rx ready)
- a central interrupt status view
- a vector to the respective IRQ handler
- programmable IRQ priority
- interrupt sources:
  - Communication Controller flow-control interrupts
  - DMA complete/error/timeout
  - Timer 1 & 2 interrupts
  - interrupt from another processor on the chip (usually the Monitor processor), set via a register in the System Controller
  - packet-error interrupt from the Router
  - system fault interrupt
  - software interrupt, for downgrading FIQ to IRQ

5.2 Register summary

Base address: 0x2f000000 (buffered write), 0x1f000000 (unbuffered write), 0xfffff000 (high).

User registers

The following registers allow normal user programming of the VIC:

Name	Offset	R/W	Function
r0: VICirqStatus	0x0	R	IRQ status register
r1: VICfiqStatus	0x4	R	FIQ status register
r2: VICrawInt	0x8	R	raw interrupt status register
r3: VICintSel	0xC	R/W	interrupt select register
r4: VICintEnable	0x10	R/W	interrupt enable register
r5: VICintEnClear	0x14	W	interrupt enable clear register
r6: VICsoftInt	0x18	R/W	soft interrupt register
r7: VICsoftIntClear	0x1C	W	soft interrupt clear register

Name	Offset	R/W	Function
r8: VICprotection	0x20	R/W	protection register
r9: VICvectAddr	0x30	R/W	vector address register
r10: VICdefVectAddr	0x34	R/W	default vector address register
VICvectAddr[15:0]	0x100-13c	R/W	vector address registers
VICvectCtrl[15:0]	0x200-23c	R/W	vector control registers

ID registers

In addition, there are test ID registers that will not normally be of interest to the programmer:

Name	Offset	R/W	Function
VICPeriphID0-3	0xFE0-C	R	Timer peripheral ID byte registers
VICPCID0-3	0xFF0-C	R	Timer Prime Cell ID byte registers

See the VIC Technical Reference Manual ARM DDI 0181E, for further details of the ID registers.

5.3 Register details

register 0: IRQ status

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
IRQ status																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This read-only register yields the set of active IRQ requests (after masking).

register 1: FIQ status

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
FIQ status																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This read-only register yields the set of active FIQ requests (after masking).

register 2: raw interrupt status

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
interrupt request status																															

This read-only register yields the set of active input interrupt requests (before any masking).

register 3: interrupt select

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
interrupt select																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register selects for each of the 32 interrupt inputs whether it gets sent to IRQ (0) or FIQ (1). The reset state is not specified (though is probably '0'?); all interrupts are disabled by r4 at reset.

register 4: interrupt enable register

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
interrupt enables																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register disables (0) or enables (1) each of the 32 interrupt inputs. Writing a '1' sets the corresponding bit in r4; writing a '0' has no effect. Interrupts are all disabled at reset.

register 5: interrupt enable clear

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
interrupt enable clear																															

This write-only register selectively clears interrupt enable bits in r4. A '1' clears the corresponding bit in r4; a '0' has no effect.

register 6: soft interrupt register

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
soft interrupt register																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

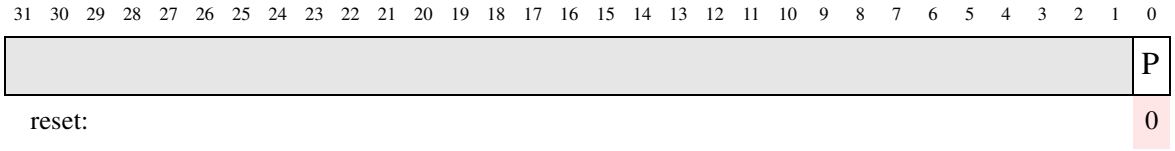
This register enables software to force interrupt inputs to appear high (before masking). A '1' written to any bit location will force the corresponding interrupt input to be active; writing a '0' has no effect. The reset state for these bits is unspecified, though probably '0'?

register 7: soft interrupt register clear

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
soft interrupt register clear																															

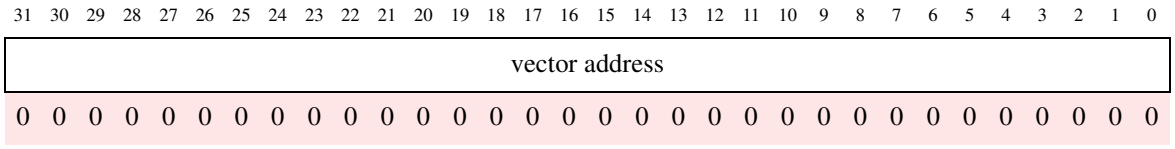
This write-only register selectively clears soft interrupt bits in r6. A '1' clears the corresponding bit in r6; a '0' has no effect.

register 8: protection



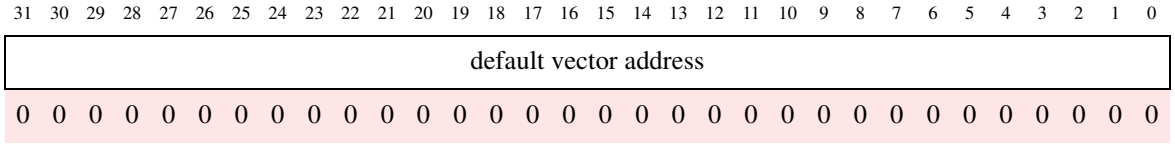
If the P bit is set VIC registers can only be accessed in a privileged mode; if it is clear then User-mode code can access the registers.

register 9: vector address



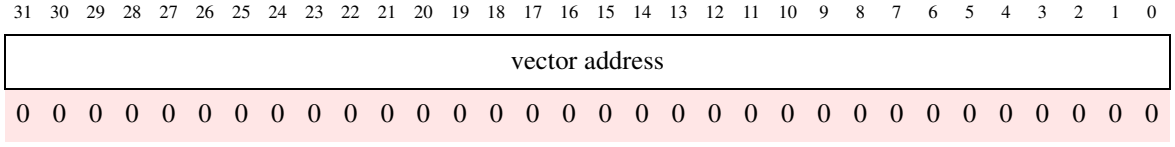
This register contains the address of the currently active interrupt service routine (ISR). It must be read at the start of the ISR, and written at the end of the ISR to signal that the priority logic should update to the next priority interrupt. Its state following reset is undefined.

register 10: default vector address



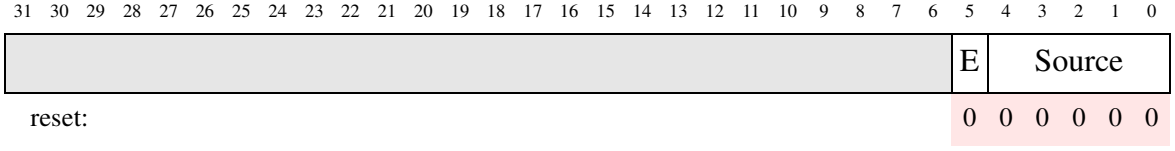
The default vector address is used by the 16 interrupts that are not vectored. Its state following reset is undefined.

vector address [15:0]



The vector address is the address of the ISR of the selected interrupt source. Their state following reset is undefined.

vector control [15:0]



The interrupt source is selected by bits[4:0], which choose one of the 32 interrupt inputs. The interrupt can be enabled (E = 1) or disabled (E = 0). It is disabled following reset.

The highest priority interrupt uses vector address [0] at offset 0x100 and vector control [0] at offset 0x200, and then successively reduced priority is given to vector addresses [1], [2], ... and vector

controls [1], [2], ... at successively higher offset addresses.

5.4 Interrupt sources

14 of the 32 interrupt sources are local to the processor, 4 are from chip-wide sources (which will normally be enabled only in the Monitor Processor), and 14 are unused.

The interrupt sources are summarised in the table below:

#	Name	Function
0	Watchdog	Watchdog timer interrupt
1	software int	used only for local software interrupt generation
2	Comms Rx	the debug communications receiver interrupt
3	Comms Tx	the debug communications transmitter interrupt
4	Timer 1	Local counter/timer interrupt 1
5	Timer 2	Local counter/timer interrupt 2
6	CC Rx ready	Local comms controller packet received
7	CC Rx parity error	Local comms controller received packet parity error
8	CC Rx framing error	Local comms controller received packet framing error
9	CC Tx full	Local comms controller transmit buffer full
10	CC Tx overflow	Local comms controller transmit buffer overflow
11	CC Tx empty	Local comms controller transmit buffer empty
12	DMA done	Local DMA controller transfer complete
13	DMA error	Local DMA controller error
14	DMA timeout	Local DMA controller transfer timed out
15	Router diagnostics	Router diagnostic counter event has occurred
16	Router dump	Router packet dumped - indicates failed delivery
17	Router error	Router error - packet parity, framing, or time stamp error
18	Sys Ctl int	System Controller interrupt bit set for this processor
19	Ethernet Tx	Ethernet transmit frame interrupt
20	Ethernet Rx	Ethernet receive frame interrupt
21	Ethernet PHY	Ethernet PHY/external interrupt
22-31	not used	

5.5 Fault-tolerance

Fault insertion

It is fairly easy to mess up vector locations, and to fake interrupt sources.

Fault detection

A failed vector location effectively causes a jump to a random location; this would be messy!

Fault isolation

Failed vector locations can be removed from service.

Reconfiguration

A failed vector location can be removed from service (provided there are enough vector locations available without it). Alternatively, the entire vector system could be shut down and interrupts run by software inspection of the IRQ and FIQ status registers.

5.6 Test

production test

start-up test

run-time test

Most registers can be tested by read/write tests at any time.

6. Counter/timer

Each processor node on an SpiNNaker chip has a local counter/timer.

The counter/timers use the standard AMBA peripheral device described on page 4-24 of the AMBA Design Kit Technical Reference Manual ARM DDI 0243A, February 2003. The peripheral has been modified only in that the APB interface of the original has been replaced by an AHB interface for direct connection to the ARM968 AHB-Lite bus.

6.1 Features

- the counter/timer unit provides two independent counters, for example for:
  - millisecond interrupts for real-time dynamics
- free-running and periodic counting modes
  - automatic reload for precise periodic timing
  - one-shot and wrapping count modes
- 100 MHz counter clock may be pre-scaled by dividing by 1, 16 or 256

6.2 Register summary

Base address: 0x21000000 (buffered write), 0x11000000 (unbuffered write).

User registers

The following registers allow normal user programming of the counter/timers:

Name	Offset	R/W	Function
r0: Timer1load	0x0	R/W	Load value for Timer 1
r1: Timer1value	0x4	R	Current value of Timer 1
r2: Timer1Ctl	0x8	R/W	Timer 1 control
r3: Timer1IntClr	0xc	W	Timer 1 interrupt clear
r4: Timer1RIS	0x10	R	Timer 1 raw interrupt status
r5: Timer1MIS	0x14	R	Timer 1 masked interrupt status
r6: Timer1BGload	0x18	R/W	Background load value for Timer 1
r8: Timer2load	0x20	R/W	Load value for Timer 2
r9: Timer2value	0x24	R	Current value of Timer 2
r10: Timer2Ctl	0x28	R/W	Timer 2control
r11: Timer2IntClr	0x2c	W	Timer 2interrupt clear
r12: Timer2RIS	0x30	R	Timer 2raw interrupt status
r13: Timer2MIS	0x34	R	Timer 2masked interrupt status
r14: Timer2BGload	0x38	R/W	Background load value for Timer 2

Test and ID registers

In addition, there are test and ID registers that will not normally be of interest to the programmer:

Name	Offset	R/W	Function
TimerITCR	0xF00	R/W	Timer integration test control register
TimerITOP	0xF04	W	Timer integration test output set register
TimerPeriphID0-3	0xFE0-C	R	Timer peripheral ID byte registers
TimerPCID0-3	0xFF0-C	R	Timer Prime Cell ID byte registers

See AMBA Design Kit Technical Reference Manual ARM DDI 0243A, February 2003, for further details of the test and ID registers.

6.3 Register details

As both timers have the same register layout they can both be described as follows (X = 1 or 2):

register 0/8: Timer X load value

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
Load value for TimerX																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When written, the 32-bit value is loaded immediately into the counter, which then counts down from the loaded value. The background load value (r6/14) is an alternative view of this register which is loaded into the counter only when the counter next reaches zero.

register 1/9: Current value of Timer X

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
TimerX current count																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

This read-only register yields the current count value for Timer X.

register 2/10: Timer X control

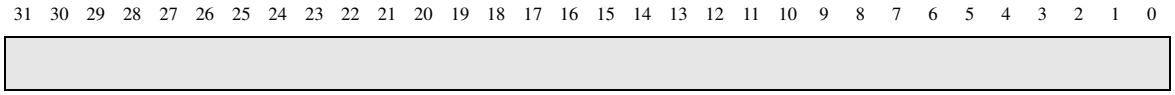
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								E	M	I		Pre		S	O
reset:																								0	0	1		0	0	0	0

The shaded fields should be written as zero and are undefined on read. The functions of the remaining fields are described in the table below:

Name	bits	R/W	Function
E: Enable	7	R/W	enable counter/timer (1 = enabled)

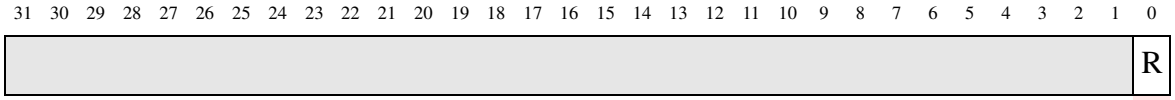
Name	bits	R/W	Function
M: Mode	6	R/W	0 = free-running; 1 = periodic
I: Int enable	5	R/W	enable interrupt (1 = enabled)
Pre: TimerPre	3:2	R/W	divide input clock by 1 (00), 16 (01), 256 (10)
S: Timer size	1	R/W	0 = 16 bit, 1 = 32 bit
O: One shot	0	R/W	0 = wrapping mode, 1 = one shot

register 3/11: Timer X interrupt clear



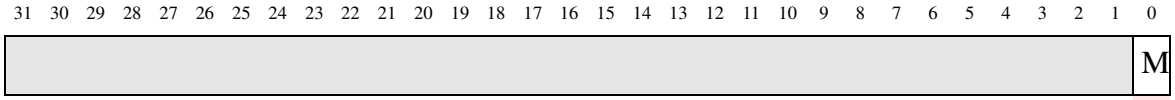
Any write to this address will clear the interrupt request.

register 4/12: Timer X raw interrupt status



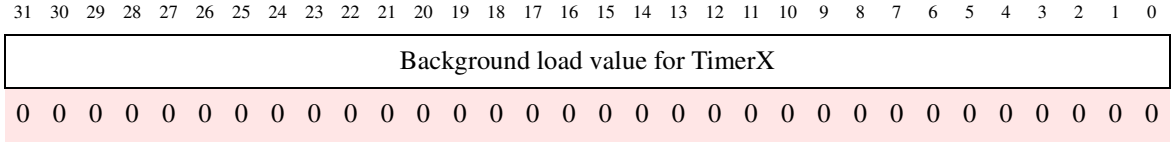
Bit zero yields the raw (unmasked) interrupt request status of this counter/timer.

register 5/13: Timer X masked interrupt status



Bit zero yields the masked interrupt status of this counter/timer.

register 6/14: Timer X background load value



The 32-bit value written to this register will be loaded into the counter when it next counts down to zero. Reading this register will yield the same value as reading register 0/8.

6.4 Fault-tolerance

Fault insertion

Disabling a counter (by clearing the E bit in its control register) will cause it to fail in its function.

Fault detection

Use the second counter/timer with a longer period to check the calibration of the first?

Fault isolation

Disable the counter/timer with the E bit in the control register; disable its interrupt output; disable the interrupt in the interut controller.

Reconfiguration

If one counter fails then a system that requires only one counter can use the other one.

6.5 Test

production test

start-up test

run-time test

See ‘fault detection’ above.

7. 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. In addition, the DMA controller provides access to other targets on the System NoC such as the System RAM and Boot ROM.

7.1 Features

- Multithreaded DMA engine supporting parallel operations
  - DMA requests
  - direct pass-through requests from the ARM968
  - dual buffers supporting simultaneous direct and DMA transfers
- Support for CRC error control in transferred blocks
- Interrupt-driven or polled DMA completion notification
  - DMA complete interrupt signal
  - DMA error interrupt signal
  - DMA time-out interrupt signal
- Parameterisable buffer sizes
- Direct and DMA request queueing

7.2 Using the DMA Controller

There are 2 types of requests for DMA controller services. DMA requests are initiated by writing to control registers in the controller, executed in background, and signal an interrupt when complete. Bridge requests occur when the ARM initiates a request directly to the needed device or service. The DMA controller fulfills these requests transparently, the host processor retaining full control of the transfer. Invisible to the user, the controller may buffer the data from write requests for more efficient bus management. If an error occurs on such a buffered write the DMA controller will signal an error interrupt. The ARM processor must control the entire data transfer throughout the process.

The controller acts as a bridge between the AHB bus on the ARM AHB slave interface and the AXI interface on the system NoC, performing the required address and control resequencing (stripping addresses from non-first beats of a burst, and retiming for AXI clock), data flow management (ensuring that data does not reach the AXI interface ahead of address) and request arbitration. The arbiter prioritises requests in the following order:

1. bridge reads and unbuffered writes,
2. buffered bridge writes,
3. DMA burst requests.

No request can gain access to the AXI interface until all active burst transactions on the interface have completed. Read requests while a DMA transfer is in progress require special handling. The read must wait until all active requests have completed, and therefore a bridge read could stall the processor and AHB slave bus for many cycles. In addition, if buffered writes exist, potential data coherency conflicts exist The recommended procedure is for the ARM processor to interrogate the buf_write_pending (B) bit in the DMA_Status register before requesting a bridge read.

To initiate a DMA transfer, the ARM must write to the following registers in the DMA controller: System Address (ADRS), TCM Address (ADRT), Length (LEN), and Control (CTRL). Ordering of the first 3 register operations is not important but the control write must be the last, to set the Start bit that initiates the DMA transfer. The processor may also optionally write the Global Control

(GCTL) register to set up additional parameters. The expected model, however, is that this register is updated infrequently, perhaps only once after power-up. The processor may read from any register at any time. Once a transfer has been committed (as the MAIN_COMT bit in the Status (STAT) register indicates), the processor may queue another request. The processor may have a maximum of 2 submitted DMA requests of which only one will be active. For the active request, the processor may only write to certain specific bits in the control register. An attempt to write any other register during an active transfer will result in an error.

7.3 Register summary

Base address: 0x40000000 (buffered write), 0x30000000 (unbuffered write).

Name	Offset	R/W	Function
r0: ADRS	0x00	R/W	DMA address on the system interface
r1: ADRT	0x04	R/W	DMA address on the TCM interface
r2: LEN	0x08	R/W	Length of the transfer in bytes
r3: CTRL	0x0C	R/W	Control DMA transfer
r4: STAT	0x10	R	Status of DMA and other transfers
r5: GCTL	0x14	R/W	Control of the DMA device
r6: CRCP	0x18	R/W	32-bit CRC checking polynomial
r7: CRCC	0x1C	R	CRC value calculated by CRC block
r8: CRCR	0x20	R	CRC value in received block
r64: AD2S	0x100	R*	Queued system address
r65: AD2T	0x104	R*	Queued TCM address
r66: LN2	0x108	R*	Queued length
r67: CTL2	0x10C	R*	Queued control
r70: CRC2	0x118	R*	Queued CRC polynomial

* These double-buffer registers are automatically written to if there is an active DMA transfer by writing to the addresses of their corresponding primary registers

7.4 Register details

r0: System Address.

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
System Address																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The 32-bit start address on the system interface. Note that a read is considered a data movement from a source on the system bus to a destination on the TCM bus.

r1: TCM Address.

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
System Address																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The 32-bit start address on the TCM interface. Note that a read is considered a data movement from a source on the system bus to a destination on the TCM bus.

r2: Length.

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
																Length															
reset:																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The function of this field is described in the table below:

Name	bits	R/W	Function
Length	15:0	R/W	length of the DMA transfer, in bytes

The TCM as currently envisioned has a maximum size of 64k (for the data TCM), thus, the length is a 16-bit register. A DMA transfer must of necessity either take as a source or a destination the TCM, justifying this restriction.

r3: Control Register

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
																					burst				W	P	D	L	R	A	S
reset:																					0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
S: Start	0	R/W	setting this bit starts a DMA transfer
A: Abort	1	R/W	end current transfer and discard data
R: Restart	2	R/W	resume transfer
L: Lock	3	R/W	lock transfer on system interface
D: Direction	4	R/W	read from (0) or write to (1) system bus
P: Priority	5	R/W	normal (0) or high (1) priority DMA transfer
W: Width	6	R/W	transfer word width: 32-bit (0) or 64-bit (1)
B: Burst	10:7	R/W	burst length - 1

r4: Status Register.

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
processor ID											Error											Qpos				K	B	F	T	Q	M	
hardwired proc ID											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
M: Committed	0	R	main DMA registers busy - transfer in progress
Q: Queue full	1	R	queue DMA registers committed
T: Transfer	2	R	a DMA or bridge transfer is in progress
F: Bridge full	3	R	bridge buffer is full
B: Bridge write	4	R	the bridge write buffer is not empty
K: Token	5	R	DMA controller granted access to system NoC
Qpos	9:6	R	position in system NoC request queue
Error	20:10	R	error code
processor ID	31:24	R	hardwired processor ID identifies CPU on chip

r5: Global Control

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
																								BfE		ErD		D	S	B	E
reset:																								1	1	0	0	1	0	0	1

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

Name	bits	R/W	Function
E: Enable	0	R/W	enable the DMA interface
B: Bridge buffer	1	R/W	buffer bridge writes
S: Swap	2	R/W	swap request queue buffers: r0 <-> r64, etc.
D: Double	3	R/W	enable double request DMA buffering
ErD: Error detect	5:4	R/W	error detect (none, generate, examine, both)
BfE: Buff enable	7:6	R/W	enable buffers [1,0] - double buffer control

r6: CRC polynomial

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
CRC_polynomial																															
0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1

The 32-bit dynamic polynomial value to use in calculating CRCs. The default value is CRC-32: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$. The processor may change this value for flexible CRC checking, however, the value should not change during a transfer or it will corrupt all CRC checks. (Note that this feature is useful for debugging because by violating this condition the processor can inject automatic errors) Conventional notation for CRC polynomials is of the form SX_n, where the n represents a bit-position in the polynomial set to 1, and all unindicated positions are 0. For example, polynomial $X^{31} + X^{29} + X^{27} + X^{21} + X^{20} + X^{17} + X^{16} + X^{15} + X^{12} + X^{11} + X^5 + X^3 + X + 1$ would correspond to a value of 10101000001100111001100000101011. The low-order bit (the "+1") is always 1 as a fundamental property of CRCs and is therefore not in the CRC polynomial register which holds the bits for X^{32}-X^1.

r7: Calculated CRC

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
CRC_value (calculated)																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This is the 32-bit CRC value calculated by the DMA CRC unit.

r8: Received CRC

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
CRC_value (received)																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This is the 32-bit CRC value read in the block of data loaded by a DMA transfer.

r64-67, r70: Double request buffer registers

These registers are not directly written, but if the processor writes to r0-3, r6 when a DMA transfer is in progress, the value written will be transferred to the correspondig register here (rN -> r64+N). When the active DMA transfer completes these registers will automatically be copied to r0-3, r6 to initiate the next transfer.

7.5 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

7.6 Test

production test

start-up test

run-time test

8. 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.

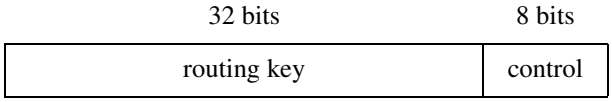
8.1 Features

- Support for 3 packet types:
  - multicast neural event packets routed by a key provided at the source;
  - point-to-point packets routed algorithmically by destination address;
  - nearest-neighbour packets routed algorithmically by arrival port.
- Packets are either 40 or 72 bits long. The longer packets carry a 32-bit payload.
- 2-bit time stamp (used by Routers to trap errant packets).
- Parity (to detect corrupt packets).

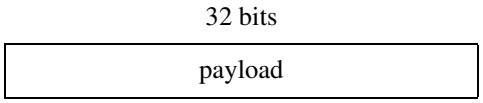
8.2 Packet formats

Neural event multicast (mc) packets (type 0)

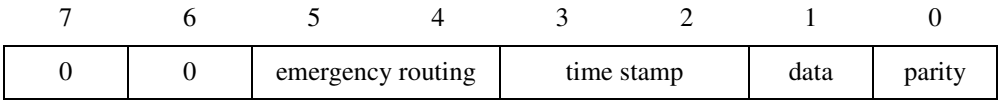
Neural event packets include a 32-bit routing key inserted by the source, and a control byte:



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

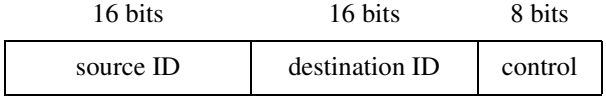


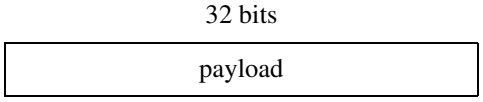
The 8-bit control field includes packet type (= 00 for multicast packets), emergency routing and time stamp information, a data payload indicator, and error detection (parity) information:



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

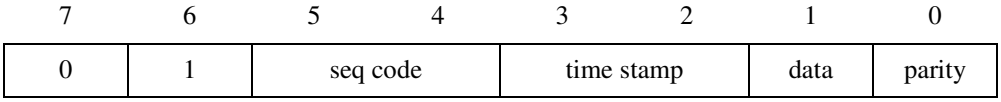
Point-to-point packets include 16-bit source and destination chip IDs, plus a control byte and an optional (normally used) 32-bit payload:





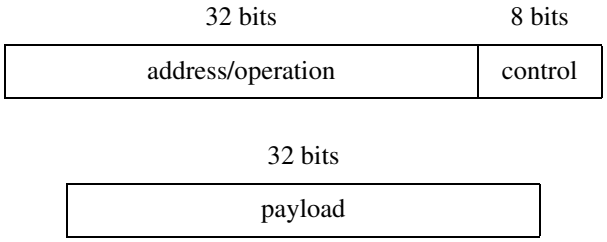
Here the 8-bit control field includes packet type (=01 for p2p packets), a sequence code, time

stamp, a data payload indicator and error detection (parity) information:

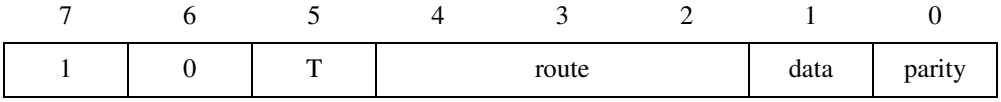


Nearest-neighbour (nn) packets (type 2)

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



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



8.3 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.

8.4 Register summary

Base address: 0x20000000 (buffered write), 0x10000000 (unbuffered write).

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

Name	Offset	R/W	Function
r4: receive data	0x10	R	32-bit received data
r5: receive key	0x14	R	Received mc key/p2p source ID & seq code
r6: reserved	0x18	-	-
r7: test	0x1C	R/W	Used for test purposes

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

8.5 Register details

r0: transmit control

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
E	F	O	U		Route			control byte								p2p source ID															
1	0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
E: empty	31	R	Tx buffer empty
F: full	30	R/W	Tx buffer full (sticky)
O: overrun	29	R/W	Tx buffer overrun (sticky)
U: unused	28:27	-	-
Route	26:24	W	Set ‘fake’ route in packet
control byte	23:16	W	control byte of next sent packet
p2p source ID	15:0	W	16-bit chip source ID for p2p packets

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.

The transmit buffer full control is expected to be used, by polling or interrupt, to prevent buffer overrun. It is sticky, and once set will remain set until 0 is written to bit 30. Transmit buffer overrun indicates packet loss and will remain set until explicitly cleared by writing 0 to bit 29.

E, F and O reflect the levels on the Tx interrupt signals sent to the interrupt controller.

The route field allows a packet to be sent by a processor to the router which appears to have come from one of the external links. Normally this field will be set to 7 (0b111) but can be set to a link number in the range 0 to 5 to achieve this.

r1: receive status

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
R	P	F	U		Route			control byte								U															
0	0	0			0	0	0	0	0	0	0	0	0	0	0																

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

Name	bits	R/W	Function
R: received	31	R	Rx packet received
P: parity	30	R/W	Rx packet parity error (sticky)
F: framing error	29	R/W	Rx packet framing error (sticky)
U: unused	28:27	-	-
Route	26:24	R	Rx route field from packet
Control byte	23:16	R	Control byte of last Rx packet
U: unused	15:0	-	-

A packet that is received without parity or framing error will set R, which will remain set until r5 has been read. A packet that is received with a parity and/or framing error sets P and/or F instead of R. These bits remain set until explicitly reset by writing 0 to bit 30 or bit 29 respectively.

R, P and F reflect the levels on the Rx interrupt signals sent to the interrupt controller.

r2: send 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 data payload for sending with next packet																															

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.

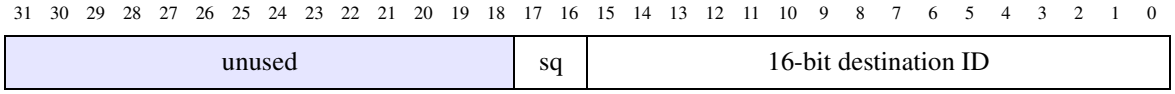
r3: send mc key/p2p dest ID & sequence code

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).

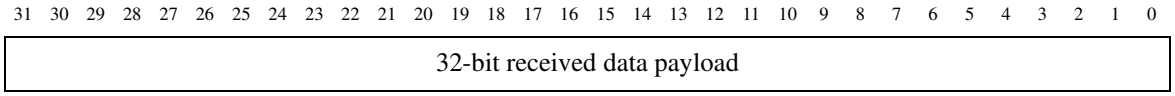
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32-bit multicast routing key																															

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 30.)



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



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.

r6: reserved

This register is reserved for future use.

r7: test

Setting bit 0 of this register makes all registers read/write for test purposes. Clearing bit 0 restricts write access to those register bits marked as read-only in this datasheet. All register bits may be read at any time.

8.6 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; received packet framing error; transmit buffer overrun.

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.

8.7 Test

production test

start-up test

run-time test

8.8 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.

9. Communications NoC

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

9.1 Features

- On- and inter-chip links
- Router which handles multicast, point-to-point and nearest neighbour packets.
- Arbiter to merge all sources into a sequential packet stream into the Router.
- Individual links can be reset to clear blockages and deadlocks.

9.2 Block diagram

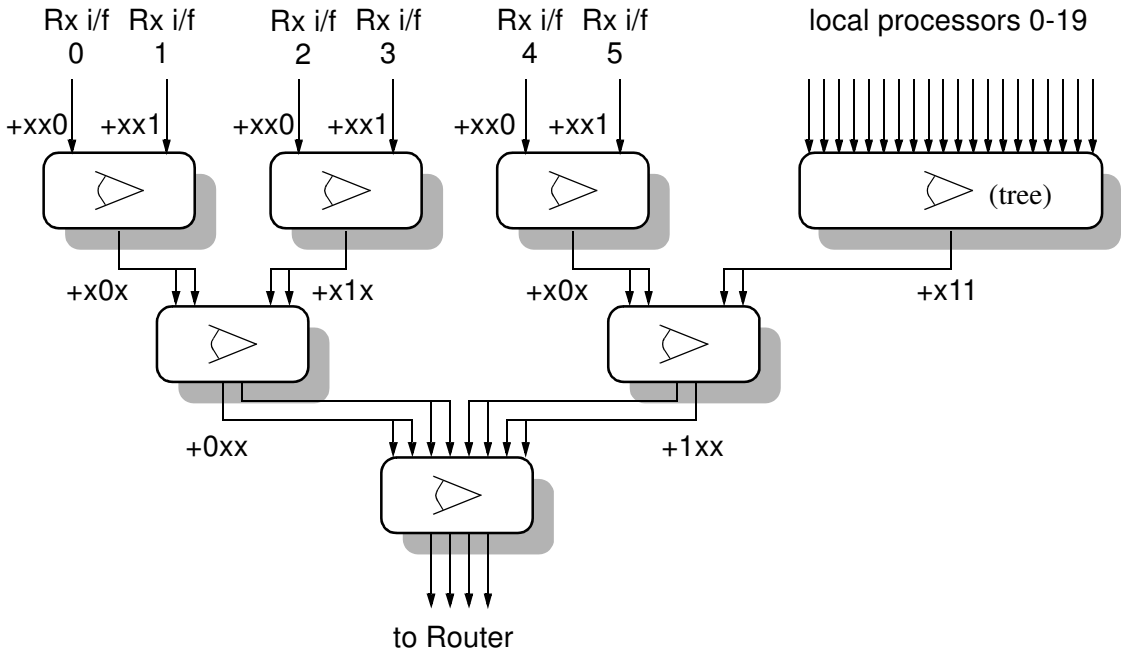
A block diagram of the Communications NoC was given in section 1.1 on page 5.

9.3 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.



9.4 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.

9.5 Test

production test

start-up test

run-time test

9.6 Notes

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

10. 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 a look-up table), 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.

10.1 Features

- *1024* programmable associative multicast (MC) routing entries.
  - associative routing based on source 'key'.
  - with flexible 'don't care' masking.
  - updatable 'on the fly'.
- look-up table routing of point-to-point (P2P) packets
- algorithmic routing of nearest-neighbour (NN) packets.
- support for 40- and 72-bit multicast, point-to-point and nearest neighbour packets.
- default routing of unmatched multicast packets.
- automatic 'emergency' re-routing around failed links.
  - programmable wait time before emergency routing and before dropping packet
- pipelined implementation to route 1 packet per cycle (peak)
  - back-pressure flow control
  - power-saving pipeline control
- failure detection and handling:
  - packet parity error
  - time-expired packet
  - output link failure
  - packet framing (wrong length) error

10.2 Description

We assume that packets 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 packets, but as each packet is handled independently the order is unimportant (though it is desirable for packets following the same route to stay in order).

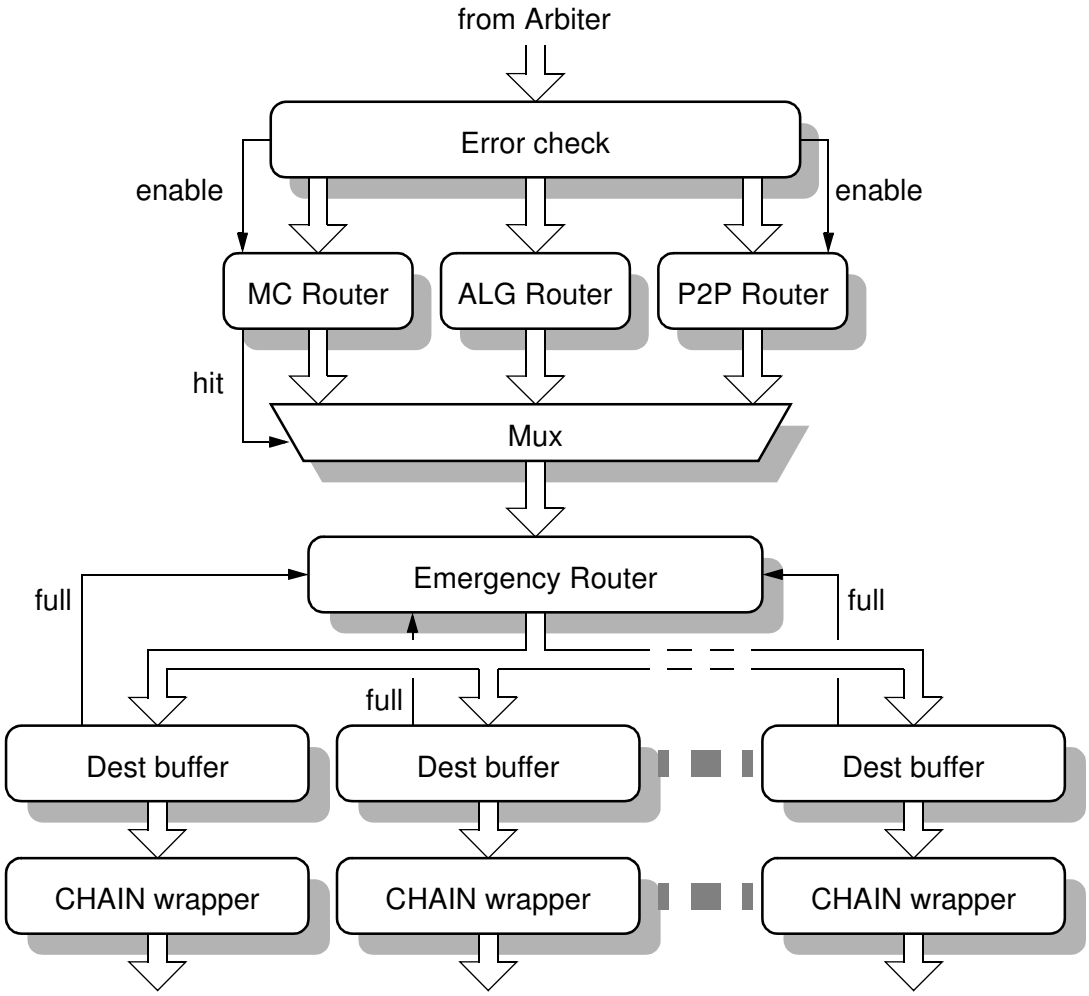
Each multicast packet contains an identifier that is used by the Communications Router to determine which of the outputs the packet is sent to. These outputs may include any subset of the output links, where the packet may be sent via the respective link transmitter interface, and/or any subset of the internal processor nodes, where the packet 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 packet – the neuron that generated the packet by firing. This is 'source address routing'. In this case the packet 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.

10.3 Internal organization

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



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, emergency 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, the point-to-point (P2P) handles point-to-point packets while the algorithmic (ALG) router handles nearest-neighbour packets and also deals with default 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.

10.4 Multicast (MC) router

The MC router uses the routing key in the MC packet to determine how to route the packet. The router has *1024* look-up entries, each of which has a mask, a key value, and an output vector.

The packet's routing key is compared with each entry in the MC router. For each entry it is first

ANDed with the mask, then compared with the entry's key. If it matches, the entry's output vector is used to determine where the packet is sent; it can be sent to any subset (including all) of the local processors and the output links.

The matching is perfomed in a parallel ternary associative memory, with a RAM used to store the output vectors. The associative memory should be set up to ensure that at most one entry matches any incoming routing key. Behaviour is undefined if two entries match a routing key (though multiple simultaneous matches *can* be used to improve test efficiency).

If no entry matches an MC packet's routing key then default routing is employed - the packet will be sent to the output link opposite the input link through which it arrived. There is no default routing for packets from local processors - the router table must have a valid entry for every locally-sourced packet.

10.5 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 3-bit SRAM lookup table directs the P2P packet to:

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

Each 3-bit entry is decoded to determine whether the packet is delivered to the local Monitor Processor, one of the six output links, or broadcast to all of the six output links.

10.6 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 poke '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 peek '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.

As the direct NN packets convey only 32-bit data payloads the bottom 2 bits of the address should always be zero. All direct NN packets return a response to the sender, with bit 0 of the address set to 1. Bit 1 will also be set to 1 if there was a bus error at the target. Peeks return a 32-bit data payload; pokes return without a payload.

default and error routing

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

10.7 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, and copied to the Error registers).

10.8 Packet error handler

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 dropped to the Error registers and the local Monitor Processor may be interrupted for error-handling purposes.

The packet error handler is a routing engine that simply flags the packet for dropping to the Error registers 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.

The Monitor Processor can be interrupted to deal with packets dropped with errors.

10.9 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 can track the problem using an appropriate diagnostic counter:
  - 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 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;
  - 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.

10.10 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) %SF - back to 0b11
```

```
            AND (p.type == 0b0x);           % time phase error
LNerr = (p.lastSymbol != EOP);              % packet length error

error = PPerr OR TPerr OR LNerr;

% 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 != 0b10);
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);
else       hit          = 0;
if (enP2P) {P2Pvect}    = P2Prouter(p.destID);

% default emergency routing vector

erVect = 0;

% dump all errors to Router error registers...

if (error)   discardPacket(p);          % ...and abandon packet; else...

% routing depends on packet type

case (p.type) {

MC: if (hit) vect = MCvect;
    else if (src == 7)
             vect = 2^(mpID+6)          % local: miss => error
    else if (p.emergencyRouting == 0b0x)
             vect = 2^[(src+3)mod6];    % normal default
    else if (p.emergencyRouting == 0b11)
             vect = 2^[(src+2)mod6];    % ER 2nd stage default
    else     vect = 0;                  % ER only
    if (p.emergencyRouting == 0b01 or 0b10)
             erVect = 2^[(src-1)mod6]; % ER 1st stage
    else     erVect = 0;

P2P:         vect = P2Pvect;

NN: if (src == 7) {                     % local source
      if (p.route == 7)                 % local MP
             vect = 2^(mpID+6);
      else if (p.route == 6)            % all neighbours
             vect = 0b00000000000000000000111111;
      else   vect = 2^(p.route);       % one neighbour
    } else {                            % external source
      if (p.T) {                        % direct NN
        if (p.data) write.SystemNoC(p.address,p.data);
        else        p.payload = read.SystemNoC(p.address);
        p.data       = ¬p.data;
```

```
        p.address[0] = 1;              % return status
        p.address[1] = SystemNoC_bus_error;
        p.T          = 0;              % change to normal
        ParityFix(p.parity);
        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<6) {
    if (bFull[i] AND (vect.bit[i] OR erVect.bit[i])) blocked = TRUE;
  }
  for (i = 6; i++; i<26) {
    if (bFull[i] AND vect.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<6) {
    if (bFull[i] AND ((vect.bit[i]       % normal route blocked
                AND (bFull[(i-1)mod6]    % ER route also blocked
                OR (p.type == NN)))      % NN are not ER'd...
                OR erVect.bit[i]))       % ... nor are ER packets
       blocked = TRUE;
  }
  for (i = 6; i++; i<26) {
     if (bFull[i] AND vect.bit[i])        % local targets can't be ER'd
       blocked = TRUE;
  }
  clockCycles++;
} while (blocked AND (clockCycles < MaxWaitForER));

% if Emergency Routing has failed...

if (blocked) discardPacket(p);             % drop packet (to error regs)

% can now proceed

for (i = 0; i++; i<6) {
  if (NOT bFull[i]) {                % send only if link open
    p2 = p;                          % copy packet

    case (p.type) {

    MC:  if (vect.bit[i] OR erVect.bit[i]
              OR (bFull[(i+1)mod6] AND vect.bit[(i+1)mod6])) {
           if (vect.bit[i]) {
             if (bFull[(i+1)mod6] AND vect.bit[(i+1)mod6])
```

```
                    p2.emergencyRouting = 0b01;     % normal + ER 1st
                 else p2.emergencyRouting = 0b00;     % normal
               } elseif (bFull[(i+1)mod6] AND vect.bit[(i+1)mod6]) {
                    p2.emergencyRouting = 0b10;     % ER 1st stage
               } elseif (erVect.bit[i]) {
                    p2.emergencyRouting = 0b11;     % ER 2nd stage
               }
               ParityFix(p2.parity);
               sendPacketTo(p2, buff[i]);
          }

     P2P: if (vect.bit[i] OR (bFull[(i+1)mod6] AND vect.bit[(i+1)mod6]))
               sendPacketTo(p2, buff[i]);

     NN:  if (vect.bit[i])
               sendPacketTo(p2, buff[i]);
     } % end case
   } % end if
} % end for

for (i = 6; i++; i<26) {
  if (vect.bit[i]) {
    p2 = p;                                          % copy packet
    if (p2.type == NN) p2.route = src;
    sendPacketTo(p2, buff[i]);
  }
}

End stage 3=======================================
```

10.11 Register summary

Base address: 0xe4000000 (buffered write), 0xf4000000 (unbuffered write).

Name	Offset	R/W	Function
r0: control	0x0	R/W	Router control register
r1: status	0x4	R	Router status
r2: error header	0x8	R	error packet control byte and flags
r3: error routing	0xC	R	error packet routing word
r4: error payload	0x10	R	error packet data payload
r5: error status	0x14	R	error packet status
r6: dump header	0x18	R	dumped packet control byte and flags
r7: dump routing	0x1C	R	dumped packet routing word
r8: dump payload	0x20	R	dumped packet data payload
r9: dump outputs	0x24	R	dumped packet intended destinations
r10: dump status	0x28	R	dumped packet status
r11: diag enables	0x2C	R/W	diagnostic counter enables
r2N: diag filter	0x200-21F	R/W	diagnostic count filters (N = 0-7)
r3N: diag count	0x300-31F	R/W	diagnostic counters (N = 0-7)

Name	Offset	R/W	Function
rT1: test register	0xF00	R/W	hardware test register 1
rT2: test key	0xF04	R/W	hardware test register 2 - CAM input test key
route[1023:0]	0x4000	R/W	MC Router routing word values
key[1023:0]	0x8000	W	MC Router key values
mask[1023:0]	0xC000	W	MC Router mask values
P2P[65535:0]	0x10000	R/W	P2P Router routing entries

10.12 Register details

register 0: Router control register

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
wait2[7:0]								wait1[7:0]								W			MP[4:0]					TP					D	E	R
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				0	0	0	0	0	0	0				0	0	1

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

Name	bits	R/W	Function
R	0	R/W	enable packet routing
E	1	R/W	enable error packet interrupt
D	2	R/W	enable dump packet interrupt
TP	7:6	R/W	time phase (c.f. packet time stamps)
MP[4:0]	12:8	R/W	Monitor Processor ID number
W	15	W	re-initialise wait counters
wait1[7:0]	23:16	R/W	wait time before emergency routing
wait2[7:0]	31:24	R/W	wait time before dropping packet

The wait times (defined by wait1[] and wait2[]) are stored in a floating point format to give a wide range of values with high accuracy at low values combined with simple implementation using a binary pre-scaler and a loadable counter. Each 8-bit field is divided into a 4-bit mantissa M[3:0] = wait[3:0] and a 4-bit exponent E[3:0] = wait[7:4]. The wait time in clock cycles is then given by:

$$wait = (M + 16 - 2^{4-E}).2^{E} \quad \text{for } E \leq 4;$$

$$wait = (M + 16).2^{E} \quad \text{for } E > 4;$$

Note that wait[7:0] = 0x00 gives a wait time of zero, and the wait time increases monotonically with wait[7:0]; wait[7:0] = 0xFF is a special case and gives an infinite wait time - wait forever.

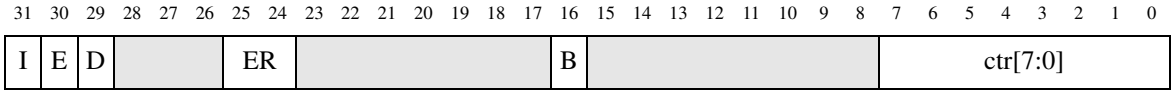
If r0 is written when one of the wait counters is running, writing a 1 to W (bit[15]) will cause the active counter to restart from the new value written to it. This enables the Monitor Processor to clear a deadlocked ‘wait forever’ condition. If 0 is written to W the active counter will not restart

but will use the new wait time value the next time it is invoked.

Note that the Router is enabled after reset. This is so that a neighbouring chip can peek and poke a chip that fails after reset using NN packets, to diagnose and possibly fix the cause of failure.

register 1: Router status

All Router interrupt request sources are visible here, as is the current status of the emergency routing system.



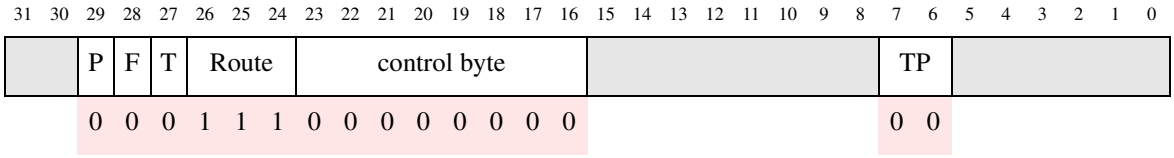
The functions of these fields are described in the table below:

Name	bits	R/W	Function
ctr[7:0]	7:0	R	diagnostic counter interrupt active
B	16	R	busy - active packet(s) in Router pipeline
ER[1:0]	25:24	R	Emergency Routing status (clear, wait1, wait2)
D: dump int	29	R	dump packet interrupt active
E: error int	30	R	error packet interrupt active
I: interrupt active	31	R	combined Router interrupt request

The Router generates three interrupt request outputs that are handled by the VIC on each processor: diagnostic counter event interrupt, dump interrupt and error interrupt. These correspond to the OR of ctr[7:0], D and E respectively.

register 2: error header

A packet which contains an error is copied to r2-5. Once a packet has been copied (indicated by bit[31] of r5 being set) any further error packet is ignored, except that it can update the sticky bits in r5 (and can be counted by a suitably-configured diagnostic counter).



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

Name	bits	R/W	Function
TP: time phase	7:6	R	time phase when packet received
Control byte	23:16	R	control byte of error packet
Route	26:24	R	Rx route field of error packet
T: TP error	27	R	packet time stamp error
F: framing error	28	R	packet framing error

Name	bits	R/W	Function
P: parity	29	R	packet parity error

register 3: error routing word

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 routing word																															

register 4: error data payload

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 data payload																															

register 5: error status

The Monitor Processor resets r5 by reading its contents.

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
E	V	P	F	T	U										TS error count																
0	0	0	0	0	0										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
TS error count	15:0	R	time stamp error count
U: undefined	26	R	undefined packet type [=11] (sticky)
T: TP error	27	R	packet time stamp error (sticky)
F: framing error	28	R	packet framing error (sticky)
P: parity	29	R	packet parity error (sticky)
V: overflow	30	R	more than one error packet detected
E: error	31	R	error packet detected

register 6: dump header

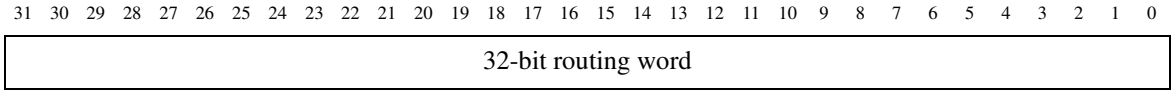
A packet which is dumped because it cannot be routed to its destinations is copied to r6-10. Once a packet has been dumped (indicated by bit[31] of r10 being set) any further packet that is dumped is ignored, except that it can update the sticky bits in r10 (and can be counted by a suitably-configured diagnostic counter).

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
					Route			control byte																TP							
reset:					1	1	1	0	0	0	0	0	0	0	0									0	0						

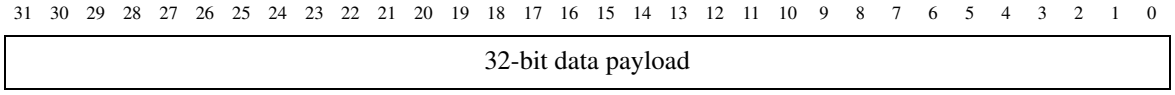
The functions of these fields are described in the table below:

Name	bits	R/W	Function
TP: time phase	7:6	R	time phase when packet dumped
Control byte	23:16	R	control byte of dumped packet
Route	26:24	R	Rx route field of dumped packet

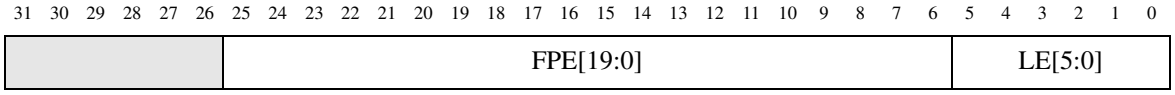
register 7: dump routing word



register 8: dump data payload



register 9: dump outputs

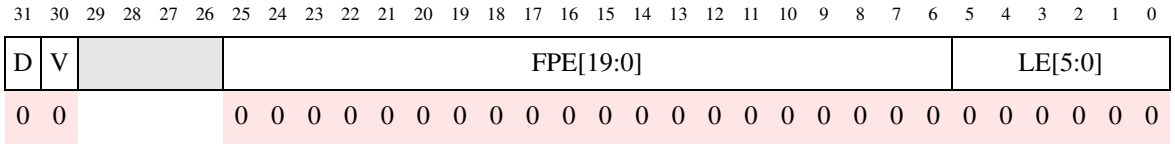


The functions of these fields are described in the table below:

Name	bits	R/W	Function
LE[5:0]	5:0	R	Tx link transmit error caused packet dump
FPE[19:0]	25:6	R	Fascicle Processor link error caused dump

register 10: dump status

The Monitor Processor resets r10 by reading its contents.

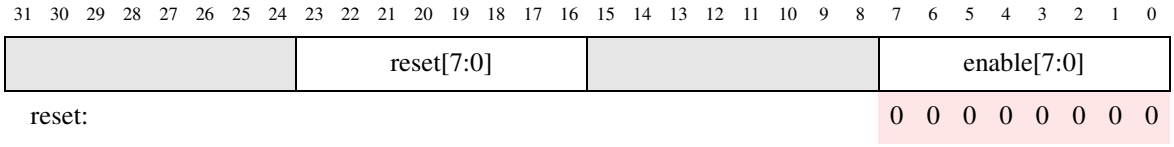


The functions of these fields are described in the table below:

Name	bits	R/W	Function
LE[5:0]	5:0	R	Tx link error caused dump (sticky)
FPE[19:0]	25:6	R	Fascicle Proc link error caused dump (sticky)
V: overflow	30	R	more than one packet dumped
D: dumped	31	R	packet dumped

register 11: diagnostic counter enable/reset

This register provides a single control point for the eight diagnostic counters, enabling them to count events over a precisely controlled time period.



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

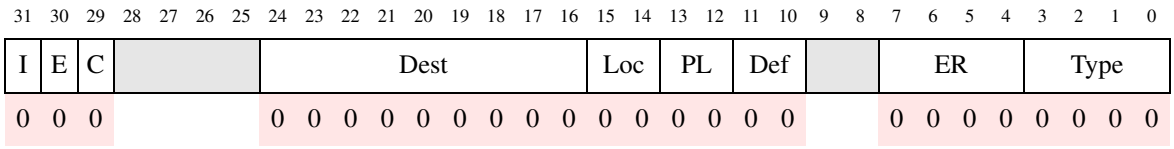
Name	bits	R/W	Function
enable[7:0]	7:0	R/W	enable diagnostic counter 7..0
reset[7:0]	23:16	W	write a 1 to reset diagnostic counter 7..0

Writing a 0 to reset[7:0] has no effect. Writing a 1 clears the respective counter.

register 2N: diagnostic control

The Router has 8 diagnostic counters (N = 0..7) each of which counts packets passing through the Router filtered on packet characteristics defined here. A packet is counted if it has characteristics that match with a '1' in each of the 7 fields. Setting all bits [23:0] to '1' will count all packets.

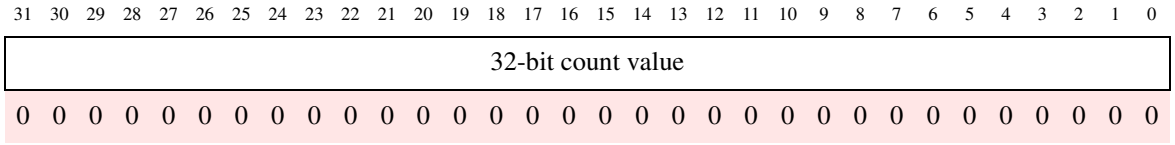
A diagnostic counter may (optionally) generate an interrupt on each count.The C bit[29] is a sticky bit set when a counter event occurs and is cleared whenever this register is read.



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

Name	bits	R/W	Function
Type	3:0	R/W	packet type: mc, p2p, nn, undefined
ER	7:4	R/W	Emergency Routing field = 0, 1, 2 or 3
Def	11:10	R/W	default [x1]/non-default [1x] routed packets
PL	13:12	R/W	packets with [x1]/without [1x] payload
Loc	15:14	R/W	local [x1]/non-local[1x] packet source
Dest	24:16	R/W	packet dest (Tx link[5:0], MP, local, dump)
C	29	R	counter event has occurred (sticky)
E	30	R/W	enable interrupt on counter event
I	31	R	counter interrupt active: I = E AND C

register 3N: diagnostic counters

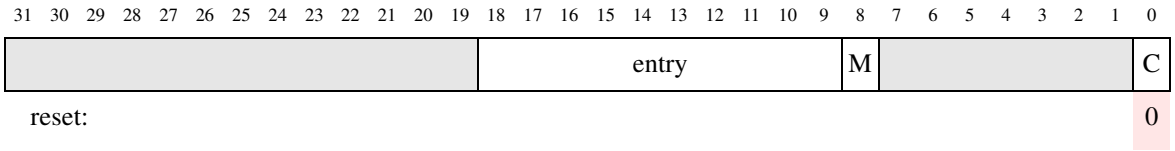


Each of these counters can be used to count selected types of packets under the control of the corresponding r1N. The counter can have any value written to it, and will increment from that value when respective events occur.

If an event occurs as the counter is being written it will not be counted. To avoid missing an event it is better to avoid writing the counter; instead, read it at the start of a time period and subtract this value from the value read at the end of the period to get a count of the number of events during the period.

register T1: hardware test register 1

This register is used only for hardware test purposes, and has no useful functions for the application programmer.

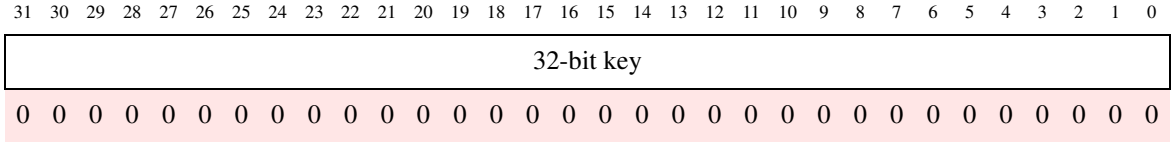


The functions of these fields are described in the table below:

Name	bits	R/W	Function
C	0	R/W	pipeline clock gating control
M	8	R	MC router associative look-up ‘miss’ output
entry	18:9	R	MC router associative look-up entry address

The input key used for the associative look-up whenever this register is read is in register T2.

register T2: hardware test register 2



This register holds the key presented to the association input to the multicast router when register T1 is read.

Multicast (MC) Router

The effect of the various combinations of bit values in the mask[] and key[] regions is decribed in

the table below:

key[]	mask[]	Function
0	0	don't care - bit matches
1	0	bit misses - entry invalidated
0	1	match 0
1	1	match 1

Thus a particular entry [i] will match only if:

- wherever a bit in the mask[i] word is 1, the corresponding bit in the MC packet routing word is the same as the corresponding bit in the key[i] word, AND
- wherever a bit in the mask[i] word is 0, the corresponding bit in the key[i] word is also 0.

Note that the MC Router CAM is not initialised at reset. Before the Router is enabled all CAM entries must be initialised by software. Unused mask[] entries should be initialised to 0x0000000, and unused key[] entries should be initialised to 0xffffffff. This invalidates every bit in the word, ensuring that the word will miss even in the presence of minor component failures.

10.13 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 multicast router entry can be disabled if it fails - see initialisation guidance above.

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)

10.14 Test

production test

The ternary CAM used in the multicast router has access for parallel testing purposes, so that a processor can write the same value to all locations and see if an input with 1 bit flipped results in a

hit or a miss.

All RAMs should have read-write access for test purposes.

start-up test

run-time test

10.15 Notes

- time stamp: writeable, updated on newer incoming packet and by internal counter?

11. 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 3-of-6 return-to-zero to 2-of-7 non-return-to-zero. Each conversion maps one CHAIN symbol to one 2-of-7 symbol. The interfaces include logic to minimise the risk of a protocol deadlock caused by glitches on the inter-chip wires.

11.1 Features

- transmit (Tx) interface:
  - converts on-chip 3-of-6 RTZ symbol into off-chip 2-of-7 NRZ symbol;
  - control input to induce a fault;
  - failure detection output?
  - fault reset input?
- receive (Rx) interface:
  - converts off-chip 2-of-7 NRZ symbol into on-chip 3-of-6 RTZ symbol;
  - control input to induce a fault;
  - failure detection output?
  - fault reset input?

11.2 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.

11.3 Fault-tolerance

The fault inducing, detecting and resetting functions are controlled from the System Controller (see ‘System Controller’ on page 65). The interfaces are ‘glitch hardened’ to greatly reduce the probability of a link deadlock arising as a result of a glitch on one of the inter-chip wires. Such a glitch may introduce packet errors, which will be detected and handled elsewhere, but it is very unlikely to cause deadlock. As a result it is expected that the link reset function will not be required at all often.

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

11.4 Test

production test

start-up test

run-time test

12. System NoC

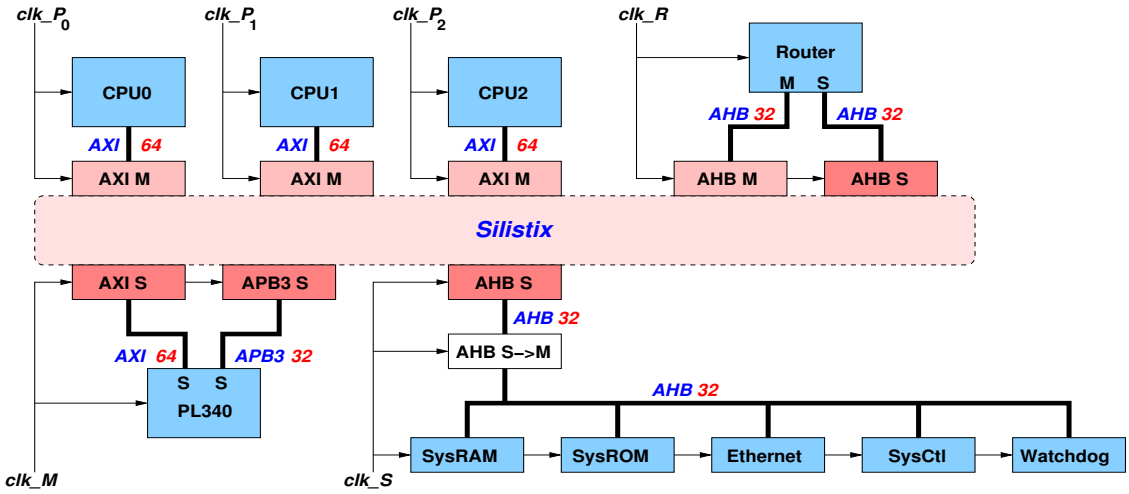
The System NoC has a primary function of connecting the ARM968 processors to the SDRAM interface. It is also used to connect the Monitor Processor to system control and test functions, and for a variety of other purposes.

The System NoC is generated by the Silistix CHAINworks tool.

12.1 Features

- supports full bandwidth block transfers between the SDRAM and the ARM968 processors.
- the Router is an additional initiator for system debug purposes.
- can be reset (in subsections?) to clear deadlocks.
- multiple targets:
  - SDRAM interface - ARM PL340
  - System RAM
  - System ROM
  - Ethernet interface
  - System Controller
  - Router configuration register

12.2 Organisation



12.3 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

12.4 Test

production test

start-up test

run-time test

13. SDRAM interface

The SDRAM interface connects the System NoC to an off-chip SDRAM device. It is the ARM PL340, described in ARM document DDI 0331D.

13.1 Features

- control for external Mobile DDR SDRAM memory device
- memory request queue with one entry per ARM968 processor
- out of order request sequencing to maximise memory throughput
- AXI interface to System NoC

13.2 Register summary

Base address: 0xe0000000 (buffered write), 0xf0000000 (unbuffered write).

User registers

The following registers allow normal user programming of the PL340 SDRAM interface:

Name	Offset	R/W	Function
r0: status	0x0	R	memory controller status
r1: command	0x4	W	PL340 command
r2: direct	0x8	W	direct command
r3: mem_cfg	0xC	R/W	memory configuration
r4: refresh_prd	0x10	R/W	refresh period
r5: CAS_latency	0x14	R/W	CAS latency
r6: t_dqss	0x18	R/W	write to DQS time
r7: t_mrd	0x1C	R/W	mode register command time
r8: t_ras	0x20	R/W	RAS to precharge delay
r9: t_rc	0x24	R/W	active bank x to active bank x delay
r10: t_rcd	0x28	R/W	RAS to CAS minimum delay
r11: t_rfc	0x2C	R/W	auto-refresh command time
r12: t_rp	0x30	R/W	precharge to RAS delay
r13: t_rrd	0x34	R/W	active bank x to active bank y delay
r14: t_wr	0x38	R/W	write to precharge delay
r15: t_wtr	0x3C	R/W	write to read delay
r16: t_xp	0x40	R/W	exit power-down command time
r17: t_xsr	0x44	R/W	exit self-refresh command time

Name	Offset	R/W	Function
r18: t_esr	0x48	R/W	self-refresh command time
id_n_cfg	0x100	R/W	QoS settings
chip_n_cfg	0x200	R/W	external memory device configuration
user_status	0x300	R	state of user_status[7:0] primary inputs
user_config	0x304	W	sets the user_config[7:0] primary outputs

Test and ID registers

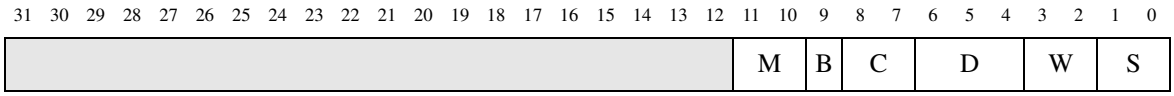
In addition, there are test and ID registers that will not normally be of interest to the programmer:

Name	Offset	R/W	Function
int_cfg	0xE00	R/W	integration configuration register
int_inputs	0xE04	R	integration inputs register
int_outputs	0xE08	W	integration outputs register
periph_id_n	0xFE0-C	R	PL340 peripheral ID byte registers
pcell_id_n	0xFF0-C	R	PL340 Prime Cell ID byte registers

See ARM document DDI 0331D for further details of the test registers.

13.3 Register details

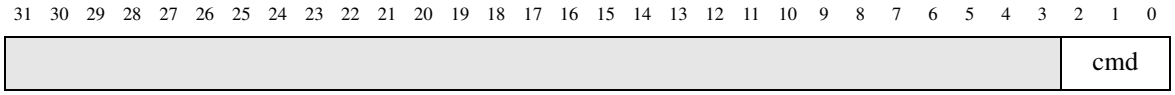
register 0: memory controller status



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

Name	bits	R/W	Function
S: status	1:0	R	Config, ready, paused, low-power
W: width	3:2	R	Width of external memory: 2'b01 = 32 bits
D: DDR	6:4	R	DDR type: 3b'011 = Mobile DDR
C: chips	8:7	R	Number of different chip selects (1, 2, 3, 4)
B: banks	9	R	Fixed at 1'b01 = 4 banks on a chip
M: monitors	11:10	R	Number of exclusive access monitors (0, 1, 2, 4)

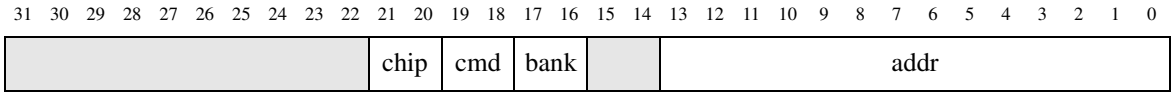
register 1: memory controller command



The function of this field is described in the table below:

Name	bits	R/W	Function
cmd: command	2:0	W	Go, sleep, wake-up, pause, config, active_pause

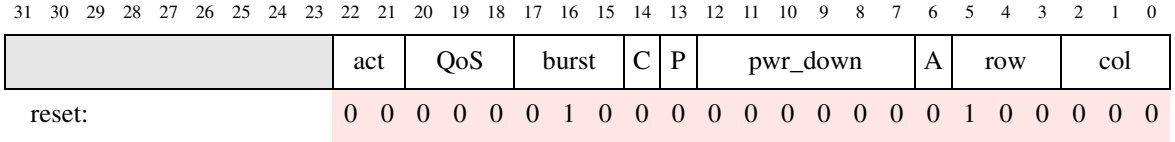
register 2: direct command



This register is used to pass a command directly to a memory device attached to the PL340. The functions of these fields are described in the table below:

Name	bits	R/W	Function
addr[13:0]	13:0	W	address passed to memory device
bank	17:16	W	bank passed to memory device
cmd	19:18	W	command passed to memory device
chip	21:20	W	chip number

register 3: memory configuration



This register is used to pass a command directly to a memory device attached to the PL340.The functions of these fields are described in the table below:

Name	bits	R/W	Function
col	2:0	R/W	number of column address bits (8-12)
row	5:3	R/W	number of row address bits (11-16)
A	6	R/W	position of auto-pre-charge bit (10/8)
pwr_down	12:7	R/W	# memory cycles before auto-power-down
P	13	R/W	auto-power-down memory when inactive
C	14	R/W	stop memory clock when no access
burst	17:15	R/W	burst length (1, 2, 4, 8, 16)

Name	bits	R/W	Function
QoS	20:18	R/W	selects the 4-bit QoS field from the AXI ARID
act	22:21	R/W	active chips: number for refresh generation

register 4: refresh period

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
																	refresh period														
reset:																	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0

The function of this field is described in the table below:

Name	bits	R/W	Function
refresh period	14:0	R/W	memory refresh period in memory clock cycles

register 5: CAS latency

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
																												cas_lat			H
reset:																												0	1	1	0

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

Name	bits	R/W	Function
H	0	R/W	CAS half cycle - must be set to 1'b0
cas_lat	3:1	R/W	CAS latency in memory clock cycles

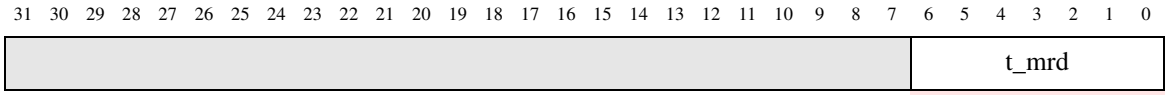
register 6: t_dqss

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
																															tdqss
reset:																														0	1

The function of this field is described in the table below:

Name	bits	R/W	Function
tdqss	1:0	R/W	write to DQS in memory clock cycles

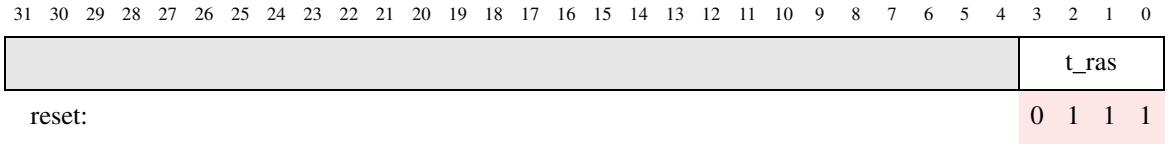
register 7: t_mrd



The function of this field is described in the table below:

Name	bits	R/W	Function
t_mrd	6:0	R/W	mode reg cmnd time in memory clock cycles

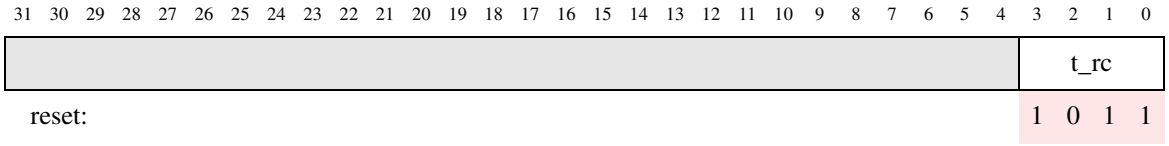
register 8: t_ras



The function of this field is described in the table below:

Name	bits	R/W	Function
t_ras	3:0	R/W	RAS to precharge time in memory clock cycles

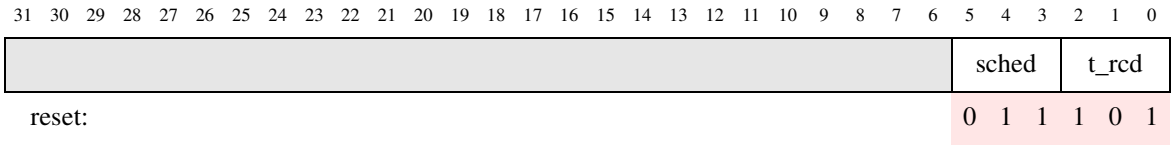
register 9: t_rc



The function of this field is described in the table below:

Name	bits	R/W	Function
t_rc	3:0	R/W	Bank x to bank x delay in memory clock cycles

register 10: t_rcd



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

Name	bits	R/W	Function
t_rcd	2:0	R/W	RAS to CAS min delay in memory clock cycles

Name	bits	R/W	Function
sched	5:3	R/W	RAS to CAS min delay in **aclk** cycles -3

register 11: t_rfc

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
																						sched					t_rfc				
reset:																						1	0	0	0	0	1	0	0	1	0

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

Name	bits	R/W	Function
t_rfc	4:0	R/W	Auto-refresh cmnd time in memory clock cycles
sched	9:5	R/W	Auto-refresh cmnd time in **aclk** cycles -3

register 12: t_rp

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
																										sched			t_rp		
reset:																										0	1	1	1	0	1

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

Name	bits	R/W	Function
t_rp	2:0	R/W	Precharge to RAS delay in memory clock cycles
sched	5:3	R/W	Precharge to RAS delay in **aclk** cycles -3

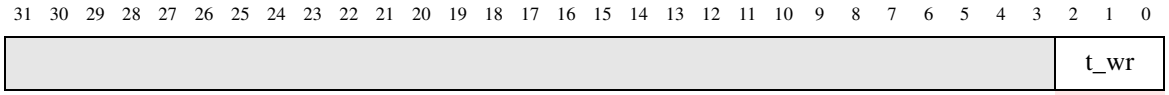
register 13: t_rrd

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
																												t_rrd			
reset:																												0	0	1	0

The function of this field is described in the table below:

Name	bits	R/W	Function
t_rrd	3:0	R/W	Bank x to bank y delay in memory clock cycles

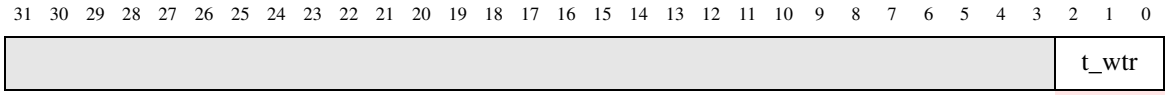
register 14: t_wr



The function of this field is described in the table below:

Name	bits	R/W	Function
t_wr	2:0	R/W	Write to precharge dly in memory clock cycles

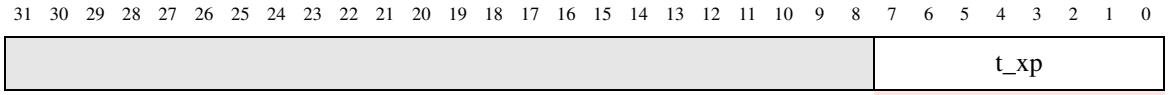
register 15: t_wtr



The function of this field is described in the table below:

Name	bits	R/W	Function
t_wtr	2:0	R/W	Write to read delay in memory clock cycles

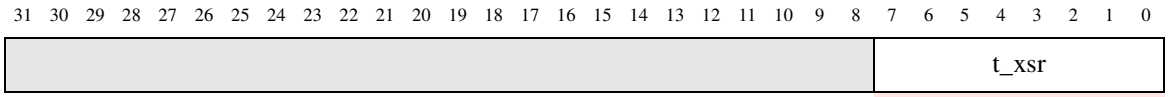
register 16: t_xp



The function of this field is described in the table below:

Name	bits	R/W	Function
t_xp	7:0	R/W	Exit pwr-dn cmnd time in memory clock cycles

register 17: t_xsr



The function of this field is described in the table below:

Name	bits	R/W	Function
t_xsr	7:0	R/W	Exit self-rfsh cmnd time in mem clock cycles

register 18: t_esr

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
																								t_esr							
reset:																								0	0	0	1	0	1	0	0

The function of this field is described in the table below:

Name	bits	R/W	Function
t_esr	7:0	R/W	Self-refresh cmnd time in memory clock cycles

id_n_cfg

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
																						QoS_max								N	E
reset:																						0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
E	0	R/W	QoS enable
N	1	R/W	minimum QoS
QoS_max	9:2	R/W	maximum QoS

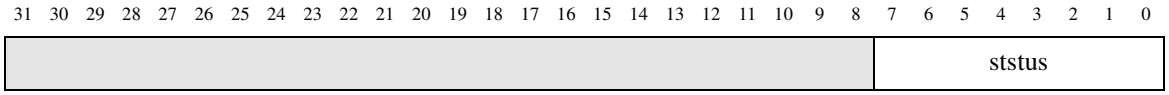
chip_n_cfg

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
															B	match								mask							
reset:															0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

There is one of these registers for each external chip that is supported. The functions of these fields are described in the table below:

Name	bits	R/W	Function
mask	7:0	R/W	address mask
match	15:8	R/W	address match
B	16	R/W	bank-rol-column/row-bank-column

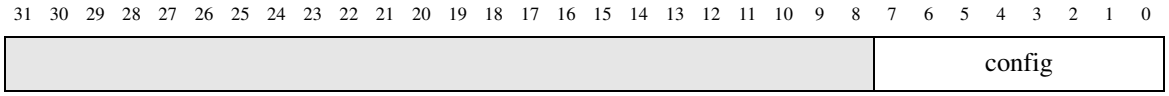
user_status



The function of this field is described in the table below:

Name	bits	R/W	Function
status	7:0	R/W	value of user_status[7:0] primary input pins

user_config



The function of this field is described in the table below:

Name	bits	R/W	Function
config	7:0	R/W	sets user_config[7:0] primary output pins

13.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

13.5 Test

production test

start-up test

run-time test

14. 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.

14.1 Features

- 'Arbiter' read-sensitive register bit to determine Monitor Processor ID at start-up
- individual interrupt, reset and clock-enable controls for each processor
- 'processor OK' bits for each processor
- sundry parallel IO and test control registers

14.2 Register summary

Base address: 0xe1000000 (buffered write), 0xf1000000 (unbuffered write).

These registers may only be accessed by a processor executing in a privileged mode; any attempt to access the System Controller from user-mode code will return a bus error.

Name	Offset	R/W	Function
r0: Chip ID	0x00	R	Chip ID register (hardwired)
r1: CPU disable	0x04	R/W	Each bit disables the clock of a processor
r2: Set CPU IRQ	0x08	R/W	Writing a 1 sets a processor's interrupt line
r3: Clr CPU IRQ	0x0C	R/W	Writing a 1 clears a processor's interrupt line
r4: Set CPU OK	0x10	R/W	Writing a 1 sets a CPU OK bit
r5: Clr CPU OK	0x14	R/W	Writing a 1 clears a CPU OK bit
r6: CPU Rst Lv	0x18	R/W	Level control of CPU resets
r7: Node Rst Lv	0x1C	R/W	Level control of CPU node resets
r8: Sbsys Rst Lv	0x20	R/W	Level control of subsystem resets
r9: CPU Rst Pu	0x24	R/W	Pulse control of CPU resets
r10: Node Rst Pu	0x28	R/W	Pulse control of CPU node resets
r11: Sbsys Rst Pu	0x2C	R/W	Pulse control of subsystem resets
r12: Reset Code	0x30	R	Indicates cause of last chip reset
r13: Monitor ID	0x34	R/W	ID of Monitor Processor
r14: Misc control	0x38	R/W	Miscellaneous control bits
r15: Misc status	0x3C	R	Miscellaneous status bits
r16: I/O port	0x40	R/W	I/O pin output register
r17: I/O direction	0x44	R/W	External I/O pin is input (1) or output (0)

Name	Offset	R/W	Function
r18: Set IO	0x48	R/W	Writing a 1 sets IO register bit
r19: Clear IO	0x4C	R/W	Writing a 1 clears IO register bit
r20: PLL1	0x50	R/W	PLL1 frequency control
r21: PLL2	0x54	R/W	PLL2 frequency control
r22: Toric0	0x58	R/W	Toric frequency synthesis control 0
r23: Toric1	0x5C	R/W	Toric frequency synthesis control 1
r24: Clk Mux Ctl	0x60	R/W	Clock multiplexer controls
r25: CPU sleep	0x64	R	CPU sleep (awaiting interrupt) status
r32-63: Arbiter	0x80-FC	R	Read sensitive semaphores to determine MP
rT1: Misc test	0x100	R/W	Miscellaneous chip test control bits
rT2: Link disable	0x104	R/W	Disables for Tx and Rx link interfaces

14.3 Register details

register 0: Chip ID

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
device												version				Year								#CPUs							
0	1	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0

This register is configured at chip design time to hold a unique ID for the chip type. The device code is 591 (which looks a bit like 'SPI'!) in BCD. The version will increment with each design variant. Year holds the last two digits of the year of first fabrication, in BCD. The bottom byte holds the number of CPUs on the chip.

The test chip ID will therefore be 0x59100802, and the first full chip ID will be 0x59100914 (assuming that is has 20 CPUs).

register 1: CPU clock disable

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
												ClkDis[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 to bit[n] (n = 0..19) will disable the clock input to processor[n], thereby halting the processor indefinitely. Writing a 0 will enable the clock. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of all of the clock disable lines.

register 2: Set CPU interrupt request

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
												SetInt[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 to bit[n] (n = 0..19) will set an interrupt request to processor[n], which can be enabled/ disabled and routed to IRQ or FIQ by that processor's local Vectored Interrupt Controller (VIC - see page 12). Writing a 0 has no effect. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of all of the processor interrupt lines.

register 3: Clear CPU interrupt request

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
												ClrInt[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 to bit[n] (n = 0..19) will clear an interrupt request to processor[n]. Writing a 0 has no effect. Reading from this register returns the current status of all of the processor interrupt lines.

register 4: Set CPU OK

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
SetOK[31:0]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 to bit[n] (n = 0..31) will set that bit, indicating that processor[n] is believed to be functional. Writing a 0 has no effect. Reading from this register returns the current status of all of the processor OK bits. Any bits that do not correspond to a processor number can be used for any purpose - the functions of this register are entirely defined by software.

In normal use a processor will set its own bit after performing some functional self-testing. The Monitor Processor will read the register after the start-up phase to establish which processors are functional, and assign them tasks accordingly. The MP may attempt to restart faulty processors by resetting them via r6-8, or it may take them off-line by disabling their clocks via r1.

register 5: Clear CPU OK

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
ClrOK[31:0]																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 to bit[n] (n = 0..31) will clear that bit, indicating that processor[n] is not confirmed as functional or has detected a fault. Writing a 0 has no effect. Reading from this register returns the current status of all of the processor OK bits.

register 6: CPU soft reset - level

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
												LSreset[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing to bit[n] (n = 0..19) will set a level on the reset input of processor[n] which is ORed with the corresponding output of the pulse reset generator, r9. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of this register, that is the level before the OR with the pulse reset output.

This is a soft reset which resets the ARM9 processor core, thereby restarting its execution at the reset vector, but does not reset the remaining hardware components in the processor node.

register 7: CPU node hard reset - level

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
												LHreset[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing to bit[n] (n = 0..19) will set a level on the reset input of processor node[n] which is ORed with the corresponding output of the pulse reset generator, r10. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of this register, that is the level before the OR with the pulse reset output.

This is a hard reset which resets the entire ARM968 processor node, including the peripheral hardware components in that node.

register 8: Subsystem reset - level

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
												LSSreset[19:0]																			
reset:												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

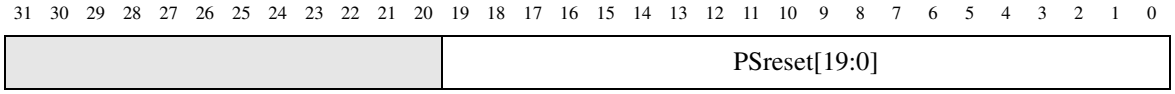
Writing a 1 to bit[n] (n = 0..19) will generate a pulse (255 clock cycles long) on the reset input of a subsystem. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of this register, that is the level before the OR with the pulse reset output.

The assignment of these bits to subsystems is given in the following table:

LSSreset	Reset target
0	Router
1	PL340 SDRAM controller
2	System NoC
3	Communications NoC
4-9	Tx link 0-5

LSSreset	Reset target
10-15	Rx link 0-5
16-19	unassigned

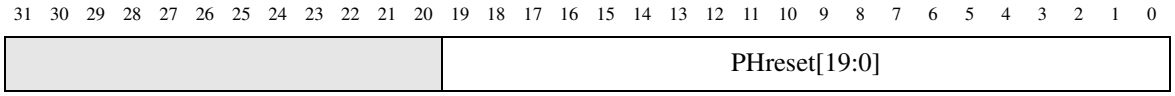
register 9: CPU soft reset - pulse



Writing a 1 to bit[n] (n = 0..19) will generate a pulse (255 clock cycles long) on the reset input of processor[n], which is ORed with the corresponding output of the reset level register r6. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of the reset lines after the OR with the level reset output.

This is a soft reset which resets the ARM9 processor core, thereby restarting its execution at the reset vector, but does not reset the remaining hardware components in the processor node.

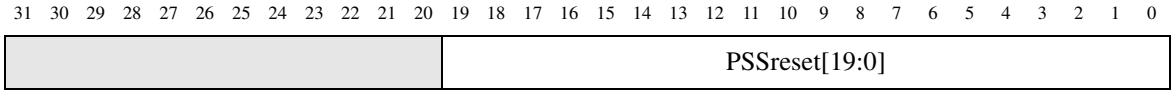
register 10: CPU node hard reset - pulse



Writing a 1 to bit[n] (n = 0..19) will generate a pulse (255 clock cycles long) on the reset input of processor node[n], which is ORed with the corresponding output of the reset level register r7. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of the reset lines after the OR with the level reset output.

This is a hard reset which resets the entire ARM968 processor node, including the peripheral hardware components in that node.

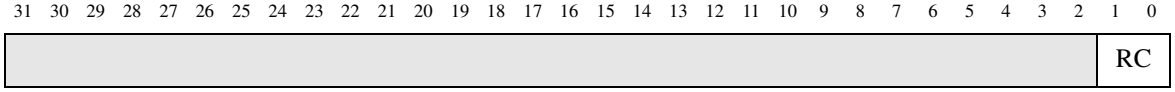
register 11: Subsystem reset - pulse



Writing a 1 to bit[n] (n = 0..19) will generate a pulse (255 clock cycles long) on the reset input of a subsystem. For a write to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX. Reading from this register returns the current status of the reset lines after the OR with the level reset output.

The assignment of these bits to subsystems is the same as that described for r8.

register 12: Reset code



These bits return a code indicating the last active reset source. The reset sources are given in the

following table:

RC[1:0]	Reset source
00	Power-on reset
01	Watchdog reset
10	??? - spare - ???
11	Watchdog interrupt

The Watchdog interrupt RC[1:0] = 11 only soft resets the Monitor Processor, and then only if this is enabled in r13.

The Power-on reset RC[1:0] = 00 hard resets everything, including setting MPID[4:0] = 11111 in r13 and B = 0 in r14.

The two intermediate resets, RC[1:0] = 01 or 10, hard reset everything apart from MPID[4:0] in r13 and B in r14, which will retain their values through the reset, thereby preventing the old Monitor Processor from competing to be Monitor Processor after the reset and allowing booting from RAM.

register 13: Monitor ID

This register holds the ID of the processor which has been chosen as the Monitor Processor, together with associated control bits.

MPID[4:0] must match the MPID value in the Router Control Register, which the Router uses to route P2P and NN packets to the Monitor Processor.

MPID[4:0] is initialised by power-on reset to an invalid value which does not refer to any processor. Other forms of reset do not change this register. It is set to the ID of the processor that wins the competition at start-up by reading its respective register r32 to r63.

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
															R								A				MPID				
reset:															0												1	1	1	1	1

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

Name	bits	R/W	Function
MPID[4:0]	4:0	R/W	Monitor Processor ID
A	8	W	Write 1 to set MP arbitration bit (see r32-63)
R	16	R/W	Reset Monitor Processor on Watchdog interrupt

The ‘R’ bit allows the Watchdog interrupt signal to cause a soft reset of processor[MPID], which will override any interrupt masking by the Monitor Processor. In any case, this interrupt is required at all processor VICs and can therefore be enabled locally as an IRQ or FIQ source.

For a write to r13 to be effective it must include a security code in bits [31:20]: 0x5ECXXXXX.

register 14: Misc control

This register contains a collection of bits which provide general chip control. The following functions

may be provided - bits to allow User mode access to various parts of the system, etc, etc.

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
misc. control																															B
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
B	0	R/W	Map System ROM (0) or RAM (1) to Boot area

register 15: Misc status

This register provides a collection of status bits. For example, state of the off-chip links, etc, etc.

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
misc. status																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

register 16: IO port

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
IO port data																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register holds a 32-bit value, some bits of which may be driven out through pins when the corresponding bit in r17 is 0. When read, the values in this register are returned. The number of physical IO pins that will be used is to be determined.

register 17: IO direction

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
IO port direction																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

This register determines whether each IO port bit is an input (1) or an output (0). Setting a bit to an input does not invalidate the corresponding bit in r16 - that value will be held in r16 until explicitly changed by a write to r16.

register 18: Set IO

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
SetIO																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 sets the corresponding bit in r16. Writing a 0 has no effect.

Reading this register returns the values on the IO pins (if present).

register 19: Clear IO

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
ClearIO																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing a 1 clears the corresponding bit in r16. Writing a 0 has no effect.

Reading this register returns the values on the IO pins (if present).

register 20: PLL1 control, and register 22: PLL2 control

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
							T						P	FR				MS[5:0]								NS[5:0]					
reset:							0						0	1	0			0	0	0	0	0	1			0	0	0	0	0	1	

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

Name	bits	R/W	Function
NS[5:0]	5:0	R/W	input clock multiplier
MS[5:0]	13:8	R/W	output clock divider
FR[1:0]	17:16	R/W	frequency range (10 = 100 to 200MHz)
P	18	R/W	NOT power down
T	24	R/W	test

The PLL output clock frequency, with a 10 MHz input clock, is given by 10*NS/MS. Thus setting NS[5:0] = 010100 [=20] and MS[5:0] = 000001 [=1] will give 200 MHz.

register 22: Toric0

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
Toric0																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

register 23: Toric1

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
Toric1																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

register 24: Clock multiplexer control

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
						Toric						Sdiv		Sys			R	Rtr		M	Mem		V	D	Po		E	Pe	
reset:						0	0					0	0	0	0		0	0	0		0	0	0	0	0	0	0		0	0	0

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

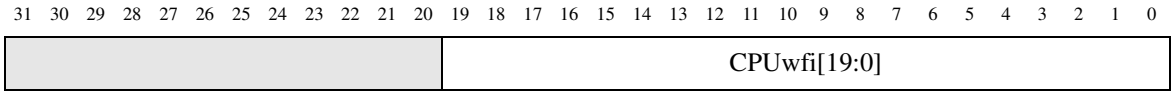
Name	bits	R/W	Function
Pe[1:0]	1:0	R/W	clock selector for even numbered CPUs
E	2	R/W	divide even CPU clock by E+1 (= 1-2)
Po[1:0]	5:4	R/W	clock selector for odd numbered CPUs
D	6	R/W	divide odd CPU clock by D+1 (= 1-2)
V	7	R/W	invert odd CPU clock
Mem[1:0]	9:8	R/W	clock selector for SDRAM
M	10	R/W	divide SDRAM clock by M+1 (= 1-2)
Rtr[1:0]	13:12	R/W	clock selector for Router
R	14	R/W	divide Router clock by R+1 (= 1-2)
Sys[1:0]	17:16	R/W	clock selector for System AHB components
Sdiv[1:0]	19:18	R/W	divide System AHB clock by Sdiv+1 (= 1-4)
Toric[1:0]	25:24	R/W	clock selector for Toric clock synthesizer

All clock selectors choose from the same clock sources:

Sel[1:0]	Clock source
00	external 10MHz clock input
01	PLL1

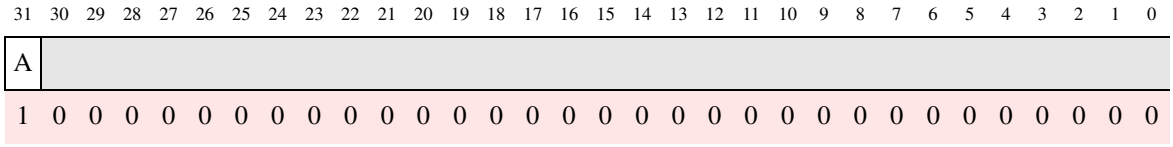
Sel[1:0]	Clock source
10	PLL2
11	Toric clock synthesizer (GND for Toric[1:0])

register 25: CPU sleep status



Each bit in this register indicates the state of the respective ARM968 STANDBYWFI (stand-by wait for interrupt) signal, which is active when the CPU is in its low-power sleep mode.

registers 32-63: Monitor Processor arbitration



The same single-bit value ‘A’ appears in all registers r32 to r63.

‘A’ is set by reset event (with RC[1:0] = 00, 01 or 10 in r12). A processor which has passed its self-test may read this register at address offset 0x80 + 4*N, where N is the processor’s number. If A is set when the read takes place and N is not equal to the current value in r13 (the Monitor Processor ID register), 0x80000000 is returned, N is placed in r13, and A is cleared.

If A is clear when the read takes place, or N equals the current value in r13, then the value 0x00000000 is returned and A and r13 are unchanged.

A read from r63 will return 0x80000000 if A is set or 0x00000000 if A is clear, but will not affect the value in A or the value in r13.

register T1: Miscellaneous test control

The test control register provides control for on-chip testing. For example, bits to simulate error condi-

tions in various parts of the chip, bits to reconfigure pins to allow testing to proceed, etc, etc.

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
misc. test																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

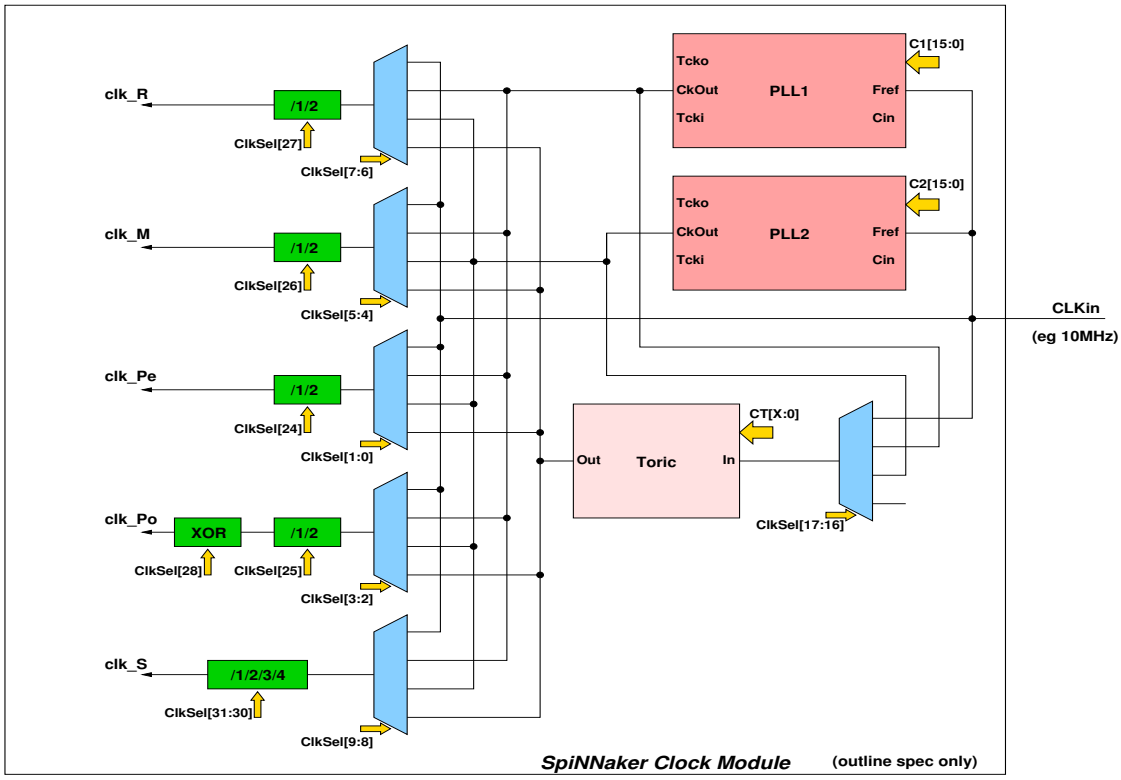
register T2: Tx and Rx link disable

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
										TxDisable																RxDisable					
reset:										0	0	0	0	0	0											0	0	0	0	0	0

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

Name	bits	R/W	Function
RxDisable[5:0]	5:0	R/W	disables the corresponding link receiver
TxDisable[5:0]	21:16	R/W	disables the corresponding link transmitter

14.4 Clock control



14.5 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

14.6 Test

production test

start-up test

run-time test

15. Ethernet MII interface

The SpiNNaker systems connects to a host machine via Ethernet links. Each SpiNNaker chip includes an Ethernet MII interface, although only a few of the chips are expected to use this interface. These chips will require an external PHY.

The interface hardware operates at the frame level. All higher-level protocols will be implemented in software running on the local monitor processor.

15.1 Features

- support for full-duplex 10 and 100 Mbit/s Ethernet via off-chip PHY
- outgoing frame buffer, for one maximum-size frame
  - outgoing frame control, CRC generation and inter-frame gap insertion
- incoming frame buffer, for two maximum-size frames
  - incoming frame descriptor buffer, for up to 48 frame descriptors
  - incoming frame control with length and CRC check
  - support for unicast (with programmable MAC address), multicast, broadcast and promiscuous frame capture
  - receive error filter
- internal loop-back for test purposes
- general-purpose IO for PHY management (SMI) and PHY reset
- interrupt sources for frame-received, frame-transmitted and PHY (external) interrupt

[The interface does not provide support for half-duplex operation (as required by a CSMA/CD MAC algorithm), jumbo or VLAN frames.]

15.2 Using the Ethernet MII interface

The Ethernet driver software must observe a number of sequence dependencies in initialising the PHY and setting-up the MAC address beore the Ethernet interface is ready for use.

Details of these issues are documented in “SpiNNaker AHB-MII module” by Brendan Lynskey. The latest version of this (currently version 003, February 2008) will be held in the SpiNNaker document repository.

15.3 Register summary

Base address: 0xe3000000 (buffered write), 0xf3000000 (unbuffered write).

User registers

The following registers allow normal user programming of the Ethernet interface:

Name	Offset	R/W	Function
Tx frame buffer	0x0000	W	Transmit frame RAM area
Rx frame buffer	0x4000	R	Receive frame RAM area
Rx desc RAM	0x8000	R	Receive descriptor RAM area
r0: gen command	0xC000	R/W	General command

Name	Offset	R/W	Function
r1: gen status	0xC004	R	General status
r2: Tx length	0xC008	R/W	Transmit frame length
r3: Tx command	0xC00C	W	Transmit command
r4: Rx command	0xC010	W	Receive command
r5: MAC addr ls	0xC014	R/W	MAC address low bytes
r6: MAC addr hs	0xC018	R/W	MAC address high bytes
r7: PHY control	0xC01C	R/W	PHY control
r8: IRQ status	0xC020	W	Interrupt clear
r9: Rx buf rd ptr	0xC024	R	Receive frame buffer read pointer
r10: Rx buf wr ptr	0xC028	R	Receive frame buffer write pointer
r11: Rx dsc rd ptr	0xC02C	R	Receive descriptor read pointer
r12: Rx dsc wr ptr	0xC030	R	Receive descriptor write pointer

Test registers

In addition, there are test registers that will not normally be of interest to the programmer:

Name	Offset	R/W	Function
r13: Rx Sys state	0xC034	R	Receive system FSM state (debug & test use)
r14: Tx MII state	0xC038	R	Transmit MII FSM state (debug & test use)
r15: PeriphID	0xC03C	R	Peripheral ID (debug & test use)

See “SpiNNaker AHB-MII module” by Brendan Lynskey version 003, January 2008 for further details of the test registers.

15.4 Register details

register 0: General command register

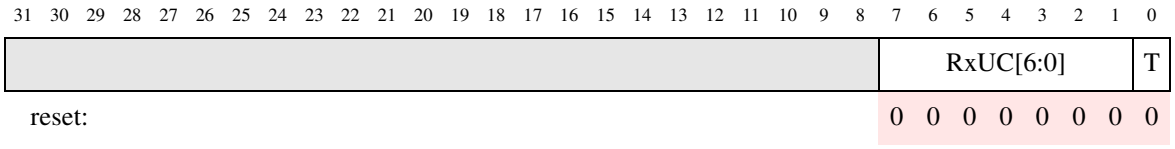
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								P	B	M	U	F	L	R	T
reset:																								0	1	1	1	1	0	0	0

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

Name	bits	R/W	Function
T	0	R/W	Transmit system enable
R	1	R/W	Receive system enable

Name	bits	R/W	Function
L	2	R/W	Loopback enable
F	3	R/W	Receive error filter enable
U	4	R/W	Receive unicast packets enable
M	5	R/W	Receive multicast packets enable
B	6	R/W	Receive broadcast packets enable
P	7	R/W	Receive promiscuous packets enable

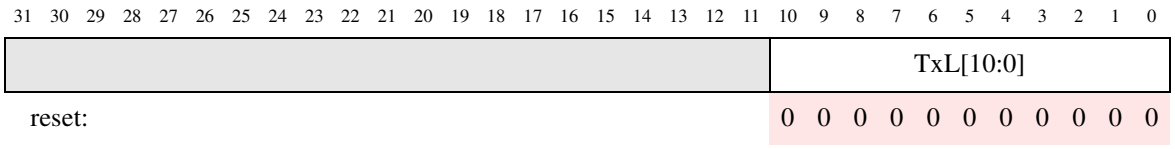
register 1: General status register



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

Name	bits	R/W	Function
T	0	R	Transmit MII interface active
RxUC[6:0]	7:1	R	Received unread frame count

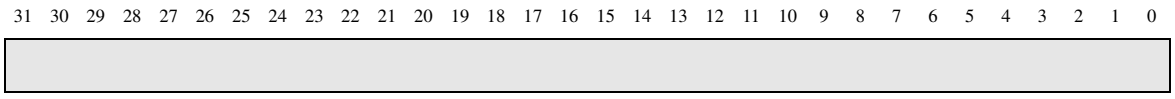
register 2: Transmit frame length



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

Name	bits	R/W	Function
TxL[10:0]	10:0	W	Length of transmit frame {60 - 1514 bytes}

register 3: Transmit command register



Any write to register 3 causes the transmission of a frame.

register 4: Receive command register

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

Any write to register 4 indicates that the current receive frame has been processed and decrements the received unread frame count in register 1.

register 5: MAC address low bytes

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
MAC3[7:0]								MAC2[7:0]								MAC1[7:0]								MAC0[7:0]							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
MAC0[7:0]	7:0	W	MAC address byte 0
MAC1[7:0]	15:8	W	MAC address byte 1
MAC2[7:0]	23:16	W	MAC address byte 2
MAC3[7:0]	31:24	W	MAC address byte 3

register 6: MAC address high bytes

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
																MAC5[7:0]								MAC4[7:0]							
reset:																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
MAC4[7:0]	7:0	W	MAC address byte 4
MAC5[7:0]	15:8	W	MAC address byte 5

register 7: PHY control

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
																											C	E	O	I	R
reset:																											0	0	0		0

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

Name	bits	R/W	Function
R	0	W	PHY reset (active low)
I	1	R	SMI data input
O	2	W	SMI data output
E	3	W	SMI data output enable
C	4	W	SMI clock (active rising)

register 8: Interrupt clear

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
																											R				T

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

Name	bits	R/W	Function
T	0	W	Clear transmit interrupt request
R	4	W	Clear receive interrupt request

Writing a 1 to bit [0] if this register clears a pending transmit frame interrupts. Writing a 1 to bit [4] clears a pending receive frame interrupt. There is no requirement to write a 0 to these bits other than in order to prevent unintentional interrupt clearance.

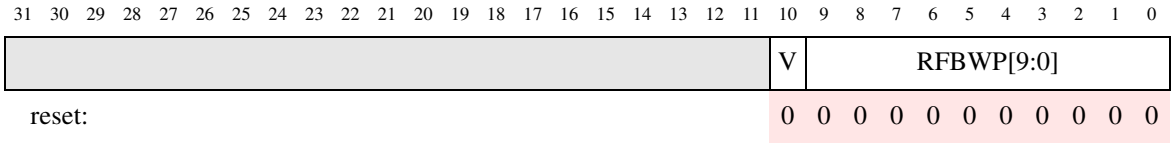
register 9: Receive frame buffer read pointer

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
																					V	RFBRP[9:0]									
reset:																					0	0	0	0	0	0	0	0	0	0	0

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

Name	bits	R/W	Function
RFBRP[9:0]	9:0	R	Receive frame buffer read pointer
V	10	R	Rollover bit - toggles on address wrap-around

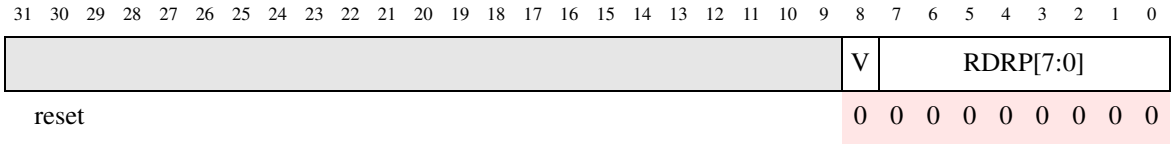
register 9: Receive frame buffer write pointer



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

Name	bits	R/W	Function
RFBWP[9:0]	9:0	R	Receive frame buffer write pointer
V	10	R	Rollover bit - toggles on address wrap-around

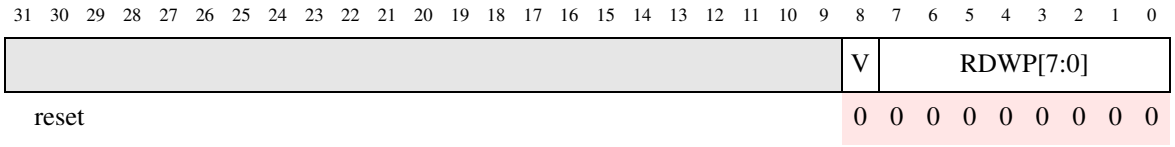
register 12: Receive descriptor read pointer



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

Name	bits	R/W	Function
RDRP[7:0]	7:0	R	Receive descriptor read pointer
V	8	R	Rollover bit - toggles on address wrap-around

register 12: Receive descriptor write pointer



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

Name	bits	R/W	Function
RDWP[7:0]	7:0	R	Receive descriptor write pointer
V	8	R	Rollover bit - toggles on address wrap-around

15.5 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

15.6 Test

production test

start-up test

Loop-back test is available.

run-time test

Loop-back test is available.

16. Watchdog timer

The watchdog timer is an ARM PrimeCell component that is responsible for applying a system reset when a failure condition is detected. Normally, the Monitor Processor will be responsible for resetting the watchdog periodically to indicate that all is well. If the Monitor Processor should crash, or fail to reset the watchdog during a pre-determined period of time, the watchdog will trigger.

16.1 Features

- generates an interrupt request after a programmable time period;
- causes a chip-level reset if the Monitor Processor does not respond to an interrupt request within a subsequent time period of the same length.

16.2 Register summary

Base address: 0xe2000000 (buffered write), 0xf2000000 (unbuffered write).

User registers

The following registers allow normal user programming of the Watchdog timer:

Name	Offset	R/W	Function
r0: WdogLoad	0x0	R/W	Count load register
r1: WdogValue	0x4	R	Current count value
r2: WdogControl	0x8	R/W	Control register
r3: WdogIntClr	0xC	W	Interrupt clear register
r4: WdogRIS	0x10	R	Raw interrupt status register
r5: WdogMIS	0x14	R	Masked interrupt status register
r6: WdogLock	0xC00	R/W	Lock register

Test and ID registers

In addition, there are test and ID registers that will not normally be of interest to the programmer:

Name	Offset	R/W	Function
WdogITCR	0xF00	R/W	Watchdog integration test control register
WdogITOP	0xF04	W	Watchdog integration test output set register
WdogPeriphID0-3	0xFE0-C	R	Watchdog peripheral ID byte registers
WdogPCID0-3	0xFF0-C	R	Watchdog Prime Cell ID byte registers

See AMBA Design Kit Technical Reference Manual ARM DDI 0243A, February 2003, for further details of the test and ID registers.

16.3 Register details

register 0: Load

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
Wdog load																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

This read-write register contains the value the from which the counter is to decrement. When this register is written to, the count immediately restarts from the new value. The minimum value is 1.

register 1: Count

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
Wdog count																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

This read-only register contains the current value of the decrementing counter. The first time the counter decrements to zero the Watchdog raises an interrupt. If the interrupt is still active the second time the counter decrements to zero the reset output is activated.

register 2: Control

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
																														E	I
reset:																														0	0

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

Name	bits	R/W	Function
I	0	R/W	Enable Watchdog counter and interrupt (1)
E	1	R/W	Enable the Watchdog reset output (1)

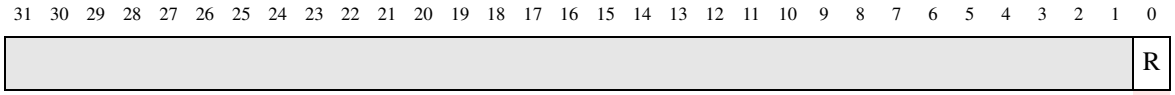
Once the Watchdog has been initialised both enables should be set to ‘1’ for normal watchdog operation.

register 3: Interrupt clear

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

A write of any value to this register clears the watchdog interrupt and reloads the counter from r1.

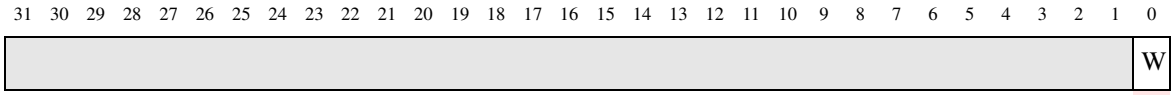
register 4: Raw interrupt status



The function of this field is described in the table below:

Name	bits	R/W	Function
R	0	R	Raw (unmasked) watchdog interrupt

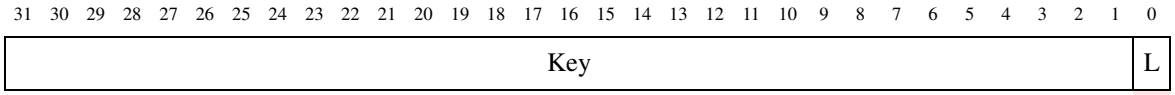
register 5: Masked interrupt status



The function of this field is described in the table below:

Name	bits	R/W	Function
W	0	R	Watchdog interrupt output

register 6: Lock



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

Name	bits	R/W	Function
L	0	R	Write access enabled (0) or disabled (1)
Key	31:0	W	Write 0x1ACCE551 to enable writes

A read from this register returns only the bottom bit, indicating whether writes to other registers are enabled (0) or disabled (1). A write of 0x1ACCE551 enables write access to the other registers; a write of any other value disables write access to the other registers. Note that the 'Key' field is 32 bits and includes bit 0.

The lock function is available to ensure that the watchdog will not be reset by errant programs.

16.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

16.5 Test

production test

start-up test

run-time test

16.6 Notes:

NOTE: What is the timer period, & do we want to try to force a different monitor processor when the watchdog expires?

17. System RAM

The System RAM is an additional *16* 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.

17.1 Features

- *16* 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?

17.2 Address location

Base address: 0xe5000000 (buffered write), 0xf5000000 (unbuffered write). Can also appear at the Boot area 0xff000000 if the 'Boot area switch' is set in the System Controller.

17.3 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.

17.4 Test

production test

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

start-up test

run-time test

18. Boot ROM

18.1 Features

- a small (16Kbyte) on-chip ROM to provide minimal support for:
  - initial self-test, and Monitor Processor selection
  - Router initialisation for bootstrapping
  - system boot.

  The Test chip will have a minimal Boot ROM suficient to enable the loading of code from an external I^2C ROM using the GPIO[1:0] pins as an I^2C interface.

18.2 Address location

Base address: 0xf6000000 and, after a hard reset and unless the 'Boot area switch' is set in the Sytem Controller, in the Boot area at 0xff000000.

18.3 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

18.4 Test

production test

start-up test

run-time test

19. Boot, test and debug support

19.1 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.

19.2 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.

19.3 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.

19.4 Fault-tolerance

Fault insertion

Fault detection

Fault isolation

Reconfiguration

19.5 Test

production test

start-up test

run-time test

20. Input and Output signals

The SpiNNaker chip has the following IO, power and ground pins. All IO is assumed to operate at 1.8v with CMOS logic levels; the SDRAM interface is tightly-specified 1.8v LVCMOS. All other IOs are non-critical, though output delay affects link throughput.

The IOs are listed and enumerated in the tables below for the full chip [and test chip].

20.1 External SDRAM interface

Signal	Type	Drive	Function	#	[#]
DQ[31:0]	I/O	8mA B	Data	1-32	1-32
A[12:0]	O	4mA B	Address	33-45	33-45
CK, CK#	O	8mA B	True and inverse clock	46, 47	46, 47
CKE	O	4mA B	Clock enable	48	48
CS#	O	4mA B	Chip select - tie to Gnd	-	-
RAS#	O	4mA B	Row address strobe	49	49
CAS#	O	4mA B	Column address strobe	50	50
WE#	O	4mA B	Write enable	51	51
DM[3:0]	O	8mA B	Data mask	52-55	52-55
BA[1:0]	O	4mA B	Bank address	56, 57	56, 57
DQS[3:0]	I/O	8mA B	Data strobe	58-61	58-61
VddQ[14:0]	1.8v		Power for SDRAM pins	62-76	62-76
VssQ[14:0]	Gnd		Ground for SDRAM pins	77-91	77-91

Noise: 10nH * (42 * 18mV/nH + 20 * 11mV/nH) / 15 = 650 mV ground/power bounce.

20.2 JTAG

Signal	Type	Drive	Function	#	[#]
TRST	I	D	Test reset	92	92
TCK	I	D	Test clock	93	93
TMS	I	D	Test mode select	94	94
TDI	I	D	Test data in	95	95
TDO	O	4mA A	Test data out	96	96

20.3 Communication links

Signal	Type	Drive	Function	#	[#]
L0in[6:0]	I	D	link 0 2-of-7 input code	97-103	97-103
L0inA	O	4mA B	link 0 input acknowledge	104	104
L0out[6:0]	O	4mA B	link 0 2-of-7 output code	105-111	105-111
L0outA	I	D	link 0 output acknowledge	112	112
L1in[6:0]	I	D	link 1 2-of-7 input code	113-119	113-119
L1inA	O	4mA B	link 1 input acknowledge	120	120
L1out[6:0]	O	4mA B	link 1 2-of-7 output code	121-127	121-127
L1outA	I	D	link 1 output acknowledge	128	128
L2in[6:0]	I	D	link 2 2-of-7 input code	129-135	129-135
L2inA	O	4mA B	link 2 input acknowledge	136	136
L2out[6:0]	O	4mA B	link 2 2-of-7 output code	137-143	137-143
L2outA	I	D	link 2 output acknowledge	144	144
L3in[6:0]	I	D	link 3 2-of-7 input code	145-151	145-151
L3inA	O	4mA B	link 3 input acknowledge	152	152
L3out[6:0]	O	4mA B	link 3 2-of-7 output code	153-159	153-159
L3outA	I	D	link 3 output acknowledge	160	160
L4in[6:0]	I	D	link 4 2-of-7 input code	161-167	-
L4inA	O	4mA B	link 4 input acknowledge	168	-
L4out[6:0]	O	4mA B	link 4 2-of-7 output code	169-175	-
L4outA	I	D	link 4 output acknowledge	176	-
L5in[6:0]	I	D	link 5 2-of-7 input code	177-183	-
L5inA	O	4mA B	link 5 input acknowledge	184	-
L5out[6:0]	O	4mA B	link 5 2-of-7 output code	185-191	-
L5outA	I	D	link 5 output acknowledge	192	-
VddL[5:0]	1.8v		Power for link pins	193-198	161-164
VssL[5:0]	Gnd		Ground for link pins	199-204	165-168

Noise: 10nH * (18 * 11mV/nH) / 6 = 330 mV ground/power bounce.

20.4 Ethernet MII

Signal	Type	Drive	Function	#	[#]
RX_CLK	I	D	Receive clock	205	145
RX_D[3:0]	I	D	Receive data	206-209	146-149
RX_DV	I	D	Receive data valid	210	150
RX_ERR	I	D	Receive data error	211	151
TX_CLK	O	2mA A	Transmit clock	212	152
TX_D[3:0]	O	2mA A	Transmit data	213-216	153-156
TX_EN	O	2mA A	Transmit data valid	217	157
TX_ERR	O	2mA A	Force transmit data error	218	158
MDC	O	2mA A	Management interface clock	219	159
MDIO	I/O	2mA A	Management interface data	220	169
PHY_RST	O	2mA A	PHY reset (optional)	221	170
PHY_IRQ	I	D	PHY interrupt (optional)	222	171
VddE	1.8v		Power for Ethernet MII pins	223	-
VssE	Gnd		Ground for Ethernet MII pins	224	-

Noise: 10nH * (9 * 3mV/nH) = 270mV ground/power bounce.

20.5 Miscellaneous

Signal	Type	Drive	Function	#	[#]
GPIO[3:0]	IO	4mA A	General-purpose IO	225-228	172-175
ResetIn	I	CSD	Chip reset	229	176
ResetOut	O	4mA A	Daisy-chain reset out	230	-
Test	I	D	Chip test mode	231	177
EtherMux	I	D	select Ethernet MII or Link 3	-	178
Clk10MIn	I	D	Main input clock - 10MHz	232	179
Clk10MOut	O	4mA A	Daisy-chain 10MHz clock out	233	-
Clk32kIn	I	CS	Slow (global) 32kHz clock	234	180
Clk32kOut	O	4mA A	Daisy-chain 32kHz clock out	235	-
VddM	1.8v		Power for miscellaneous pins	236	-
VssM	Gnd		Ground for miscellaneous pins	237	-
Vdd[7:0]	1.2v		Power for core logic	238-245	181-184
Vss[7:0]	Gnd		Ground for core logic	246-253	185-188
VddP[3:0]	1.2v		Power for PLLs	254-257	189-192
VssP[3:0]	Gnd		Ground for PLLs	258-261	193-196

Noise: 10nH * (5 * 5mV/nH) = 250mV ground/power bounce.

21. 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.)

The Artisan in-line pads for this process are 60 µm x 293 µm, but EuroPractice recommends a minimum bond pad pitch of 90 µm, so we can get 49 IOs on each side of the test chip.

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 core area (excluding pads) is 9.414 x 9.414 [4.414 x 4.414] = 88.6 [19.48] mm^2.

Estimates

Using these assumptions we total up the core logic area for the full [test] chip as follows:

- The processor nodes = 20 [2] x 3.6 = 72 [7.2] mm^2.
  - An ARM968 with *32* kByte I-RAM and *64* kByte D-RAM is 3.3 mm^2.
  - DMA, interrupt, counter/timer, communications controllers: 20 k gates = 0.2 mm^2.
  - Communications and System NoC interfaces = 0.1 mm^2.
- The Communications NoC = 7.2 [3.7] mm^2.
  - The associative router with *1024* [512] associative entries is ~ 7 [3.5] mm^2.
  - Communications network fabric ~ 0.2 mm^2.
- The System NoC = 9.85 [4.85] mm^2.
  - The *16* [16] kByte System RAM is 0.6 mm^2.
  - The 16 kByte Boot ROM is small 0.1 mm^2.
  - The System Controller with 20k gates is 0.2 mm^2.
  - The PL340 SDRAM controller is 0.35 mm^2.
  - The Ethernet controller is 0.6 mm^2.
  - The network fabric is 8 [3] mm^2.
- Boot, test and debug, PLLs = 0.5 mm^2.

Total area

The total core logic area is thus 72 [7.2] + 7.2 [3.7] + 9.85 [4.85] + 0.5 = 89.55 [16.25] mm^2.

22. 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 = 10-20 µW.

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

1.8 V I/Os, 10 pF/wire = 15 pJ/transition

3 transitions/4 bits + EOP = 33 transitions/spike = 0.5 nJ/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 + 1.5H)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.

23. To Do...

Comms controller

- Rx buffer size?

- NN packet type should be ‘1x’?

Interchip interfaces

- include Yebin's fault tolerance aspects?

Router

- details to be checked/updated following Router Review

System Controller

- several details to be completed

- reset strategy (inc. watchdog reset)

Fault-tolerance and test features

- still quite a lot to be detailed.

Does the ARM968 JTAG require the Comms Rx & Tx interrupts?

Processor ID - hard-wire onto DMA register?

Commentary

- DMA bridge operation

Fit in Toric pins & test IOs.

...and quite a lot more!