Overview of the SpiNNaker System Architecture

Steve B. Furber, *Fellow*, *IEEE*, David R. Lester, Luis A. Plana, *Senior Member*, *IEEE*, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown, *Senior Member*, *IEEE*

Abstract—SpiNNaker (a contraction of *Spiking Neural Network Architecture*) is a million-core computing engine whose flagship goal is to be able to simulate the behavior of aggregates of up to a billion neurons in real time. It consists of an array of ARM9 cores, communicating via packets carried by a custom interconnect fabric. The packets are small (40 or 72 bits), and their transmission is brokered entirely by hardware, giving the overall engine an extremely high bisection bandwidth of over 5 billion packets/s. Three of the principal axioms of parallel machine design (memory coherence, synchronicity, and determinism) have been discarded in the design without, surprisingly, compromising the ability to perform meaningful computations. A further attribute of the system is the acknowledgment, from the initial design stages, that the sheer size of the implementation will make component failures an inevitable aspect of day-to-day operation, and fault detection and recovery mechanisms have been built into the system at many levels of abstraction. This paper describes the architecture of the machine and outlines the underlying design philosophy; software and applications are to be described in detail elsewhere, and only introduced in passing here as necessary to illuminate the description.

Index Terms—Interconnection architectures, parallel processors, neurocomputers, real-time distributed

1 Introduction

The SpiNNaker engine [1] is a massively parallel multicore computing system. It will contain up to 1,036,800 ARM9 cores and 7 Tbytes of RAM distributed throughout the system in 57K *nodes*, each node being a System-in-Package (SiP) containing 18 cores plus a 128-Mbyte off-die Synchronous Dynamic Random Access Memory (SDRAM). Each *core* has associated with it 64 Kbytes of data tightly coupled memory (DTCM) and 32 Kbytes of instruction tightly coupled memory (ITCM).

The cores have a variety of ways of communicating with each other and with the memory, the dominant of which is by *packets*. These are 5 or 9-byte (40 or 72-bit) quanta of information that are transmitted around the system under the aegis of a bespoke concurrent hardware routing system.

The physical hierarchy of the system has each node containing two silicon dies—the SpiNNaker chip itself, plus the Mobile Double Data Rate (DDR) SDRAM, which is physically mounted on top of the SpiNNaker die and wire-bonded to it—see Fig. 1. The nodes are packaged and mounted in a 48-node hexagonal array on a Printed Circuit Board (PCB), the full system requiring 1,200 such boards. In operation, the engine consumes at most 90 kW of electrical power.

- *S.B. Furber, D.R. Lester, L.A. Plana, J.D. Garside, E. Painkras, and S. Temple are with the School of Computer Science, the University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom. E-mail: {steve.furber, David.Lester}@manchester.ac.uk, {plana, jdg, painkras, temples}@cs.man.ac.uk.*
- *A.D. Brown is with Electronics and Computer Science, the University of Southampton, Southampton, SO17 1BJ, United Kingdom. E-mail: adb@ecs.soton.ac.uk.*

Manuscript received 14 Jan. 2012; revised 25 May 2012; accepted 28 May 2012; published online 12 June 2012.
Recommended for acceptance by C. Bolchini.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2012-01-0038.
Digital Object Identifier no. 10.1109/TC.2012.142.

This paper will describe architectural and physical design aspects of the system. Clearly, there are many challenges associated with the design, construction, and use of a system as large and complex as this—the software and application portfolio will be described in detail elsewhere. While previous papers have presented aspects of the architecture (e.g., [2], [3]; a complete list of SpiNNaker publications is available on the project web site [1]), the contribution here is to offer a comprehensive overview focusing on the motivation and rationale for the architectural decisions taken in the design of the machine.

2 High-Level Project Goals and Background

Multicore processors are now clearly established as the way forward on the desktop, and highly parallel systems have been the norm for high-performance computing for some while. In a surprisingly short space of time, industry has abandoned the exploitation of Moore's Law through ever more complex uniprocessors, and is embracing the "new" Moore's Law: the number of *processor cores* on a chip will double roughly every 18 months. If projected over the next 25 years, this leads inevitably to the landmark of a million-core processor system.

Much work is required to understand how to optimize the scheduling of workloads on such machines, but the nature of this task is changing: in the past, a large application was distributed "evenly" over a few processors and much effort went into scheduling to keep all of the processor resources busy; today, the nature of the cost function is different: processing is effectively a free resource. Although the automatic parallelization of general-purpose codes remains a "holy grail" of computer science, biological systems achieve much higher levels of parallelism, and we turn for inspiration to connectivity patterns and computational models based on our (extremely limited) understanding of the brain.

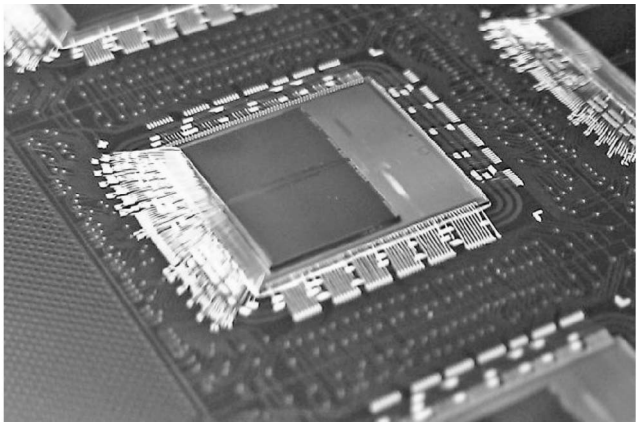


Fig. 1. SDRAM wire-bonded to the underlying SpiNNaker die. 3D packaging by UNISEM (Europe) Ltd.

This biological inspiration draws us to two parallel, synergistic directions of enquiry [4]; significant progress in *either* direction will represent a major scientific breakthrough:

- How can massively parallel computing resources accelerate our understanding of brain function?
- How can our growing understanding of brain function point the way to more efficient, parallel, fault-tolerant computation?

We start from the following question: what will happen when processors become so cheap that there is, in effect, an unlimited supply of them? The goal is now to get the job done as quickly and/or energy efficiently as possible, and as many processors can be brought into play as is useful; this may well result in a significant number of processors doing identical calculations, or indeed nothing at all—*they are a free resource*.

2.1 The Mammalian Nervous System

The mammalian nervous system—by any metric—is one of the most remarkable, effective, and efficient structures occurring in nature. The human brain exhibits massive parallelism (10^{11} neurons), and massive connectivity (10^{15} synapses). It consumes around 25W, and is composed of very low-performance components (neurons "behave" at up to around 100 Hz; the biological interconnect propagates information at speeds of a few ms^{-1}). It is massively tolerant of component-level failure—typically a human will lose neurons at a rate of about $1\ \mathrm{s}^{-1}$ throughout their adult life [5].

For a computer engineer, the similarities between the nervous system and a digital system are overwhelming. The principal component of the nervous system, the neuron [6], is a unidirectional device, connected to its peers via a single output, the *axon*. Near its terminal the axon branches and forms connections (*synapses*) with the inputs of its fellow neurons. The input structure of a neuron is termed the *dendritic tree*—see Fig. 2. Specialized neurons interface to muscles (and drive the system "actuators"), and others to various sensors.

2.2 Spiking Communication

Most biological neurons communicate predominantly via an electrochemical impulse known as an *action potential* [6]. This is a complex, propagating electrochemical pulse, supported mainly by transient sodium, potassium, chloride, and electron fluxes, and perturbations of the electrochemical impedance to these species in the axon cell walls. To a first approximation, these impulses can be viewed as spikes. The size and shape of the spike is largely invariant, (and, indeed, probably irrelevant) being determined by local instabilities in the cell membrane current balance, so a spike can be viewed as a unit impulse that conveys information solely in the time at which it occurs. It costs the axon energy to transmit an event, but this is provided by a kind of electrochemical "gain" distributed along the length of the fibre: the net effect is that—again to a first approximation—the axon can be viewed as a lossless, dispersion free transmission line, although it has to have a "rest" just after a pulse has gone by to "charge itself up" again.

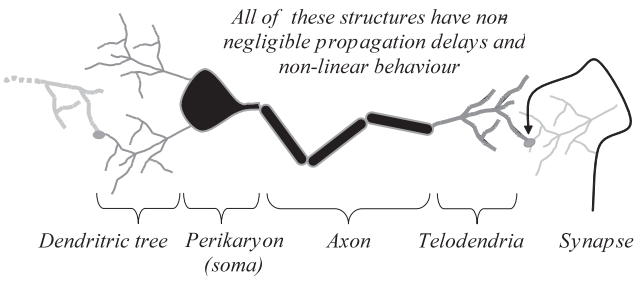


Fig. 2. A biological neuron.

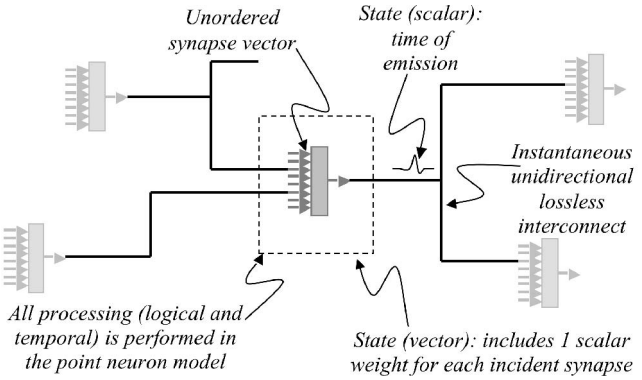


Fig. 3. The corresponding point neuron model.

2.3 Point Neuron Model

SpiNNaker is optimized for what is commonly known as the *"point neuron model"* [4], where the details of the dendritic structure of the neuron are ignored and all inputs are effectively applied direct to the soma (the "body" of the neuron). The inputs arrive in the correct temporal order—more or less—but there is no attempt to model the geometry of the dendritic tree. The abstract synaptic inputs are summed to form a net soma input that drives a system of simple differential equations that compute when an output spike should be issued.

2.4 Synapses

A synapse is the "component" whereby a spike from one neuron couples into the input to another neuron. A spike has unit impulse, but the synapse has a variable efficiency that is often represented by a numerical "weight" [4]. If the weight is positive, the synapse is excitatory. If the weight is negative, the synapse is inhibitory.

The modeling abstraction is summarized in Fig. 3. In the jargon of electronic circuits, a neural circuit is represented

by a devices-on-devices graph. Biology—as one might expect—is vastly more complex than this extreme abstraction. An unresolved issue is how much of the complexity is biological artifact, and how much is necessary for the information processing required to support a viable organism? The performance of electronic circuits is ultimately dictated by the speed and efficiency with which the flow of electrons through silicon can be choreographed by the designer—and there are physical limits. In biology, the information carriers are more diverse (ionic species) and they are controlled by an electrochemical field gradient. Ions are necessarily big, electrochemical fields necessarily small. Nature compensates by utilizing massive parallelism, but there will always be huge functional compromises. It is interesting to note that almost every creature on the planet today utilizes broadly the same structure for its controlling neural system. Comprehensive descriptions of the many types of real neurons and synapses are available elsewhere [6], [7].

2.5 Address Event Representation (AER)

The central idea of the standard SpiNNaker execution model is that of *Address Event Representation* [8], [9]. The underlying principle of AER, which is well established in the neuromorphic community, is that when a neuron fires the spike is a pure asynchronous "event." All of the information is conveyed solely in the *time* of the spike and the *identity* of the neuron that emitted the spike. In a real-time system, time models itself, so in an AER system the identity ("address") of a neuron that spikes is simply broadcast at the time that it spikes to all neurons to which the spiking neuron connects.

In SpiNNaker, AER is implemented using packet-switched communication and multicast (MC) routing. Although the communication system introduces some temporal latency, provided this is small compared with biological time constants (which in practice means provided it is well under 1 ms) then the error introduced by this latency is negligible (when modeling biological neural systems).

2.6 Topological Virtualization

Biological neural systems develop and operate in three dimensions, and both their topologies and geometries are constrained by their physical structures. SpiNNaker employs a two-dimensional physical communication structure, but this in no way limits its capacity to model three- (or higher-) dimensional networks. Because electronic communication is effectively instantaneous on biological time scales, every neuron in a SpiNNaker system can be connected to any other neuron with a time delay that equates to adjacency in the biological three-dimensional space. Thus, the mapping of neurons from the biological 3D space into the SpiNNaker 2D network of processors can be arbitrary—any neuron can be mapped to any processor. In practice, the SpiNNaker model will be more efficient if the mapping is chosen carefully, and this, in turn, means mapping physically close neurons into physically close processors, but this is only a matter of efficiency and is in no way fundamentally constrained by the SpiNNaker implementation.

2.7 Time Models Itself

Biological systems have no central synchronizing clock. Spikes are launched, spikes propagate, spikes arrive (usually), target neurons react. In a conventional electronic synchronous system, data are expected to be at the right place at the right time. If they are not, the system is broken. In an asynchronous electronic system, data arrive, are processed and passed on, and a nontrivial choreography of request and acknowledge signals ensures that the integrity of the dataflow is maintained. In biology, data are transmitted in the *hope* that most of them will get to the right place in a timely—but strictly undefined—manner. Strangely, it is clear by inspection that it is possible to create hugely complex systems—mammals—operating successfully on this principle.

In SpiNNaker, cores react to packets, process packets, and optionally emit further packets. These are transmitted to their target by the routing subsystem, to the best of its ability. If the routing fabric becomes congested—an unpredictable function of the workload—packets will, in the first instance, be rerouted (causing them to arrive late) or even dropped (if there is no space to hold them). A design axiom of SpiNNaker is that nothing can ever prevent a packet from being launched. A consequence of the effects described above is that not only is the arrival time of the packets nondeterministic, the packet ordering is nontransitive.

A single SpiNNaker core is a single ARM9 processor. This is deterministic and is expected to multiplex the behavior of around 1,000 neurons. The nodes, each containing 18 such cores, are equipped with six bidirectional fast links and embedded in a communication mesh—see the next section—which intelligently redirects and duplicates packets as necessary. The speed at which packets are transmitted over the network is about 0.2-μs/node hop, all of which means that we can reasonably expect the neuron models to react to stimuli on a wall-clock timescale of ms—just like biology.

3 The Technology Landscape

There are other approaches to brain modeling with objectives broadly similar to, though approaches rather different from, the work described here.

3.1 Blue Brain

The Blue Brain project at EPFL [10] is bringing together wet neuroscience with high-performance computing to deliver high-fidelity computer models of biological neural systems. The computing resource available to the project is an IBM Blue Gene supercomputer [11] with very sophisticated visualization facilities.

3.2 SyNAPSE

An IBM project, funded under the DARPA *SyNAPSE* programme [12] claims the successful modeling of a neural network on the scale of a cat cortex (which is around a billion neurons with 10^{13} synapses).

3.3 Izhikevich

Eugene Izhikevich, at the Neuroscience Research Institute in San Diego, developed a 100 billion neuron model based

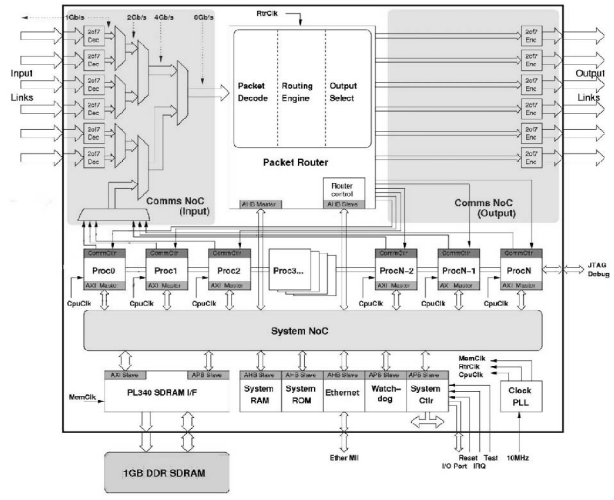


Fig. 4. A SpiNNaker node.

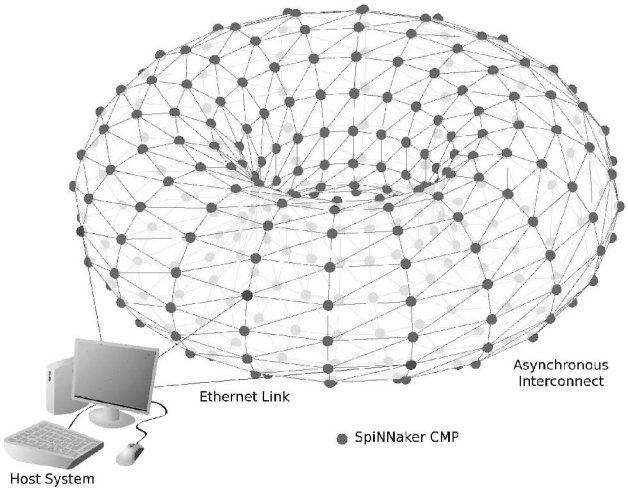


Fig. 5. The SpiNNaker machine.

on the mammalian thalamo-cortical system [13], [14]. One second of simulation took 50 days on a 27-processor Beowulf cluster.

3.4 Issues

These major projects demonstrate the debate (that is as yet unresolved within the brain modeling research community): to what extent are the finer details of biological neurons essential to the accurate modeling of the information processing capabilities of the brain, and to what extent can they be ignored as artifacts resulting from the evolutionary development of the biological neuron and its need to grow and find energy?

The SpiNNaker architecture is biased toward the simpler side of this debate—the machine is optimized for simple point neuron models and it is capable of modeling very complex networks of these simple models.

The principal differentiator of the SpiNNaker project from other large-scale neural models is our objective to run in *biological real time*. None of the above systems are close to this goal, but we believe this to be essential if the neural experiments are to benefit from "embodiment" by integration with robotic systems.

Other approaches to large-scale neural modeling are, of course, possible, for example, using GPGPUs or FPGAs. It is difficult with such approaches to achieve the balance of computation, memory hierarchy, and communication that SpiNNaker achieves, though of course they do avoid the high development cost of the bespoke chip approach.

4 Architecture Overview

4.1 Overview

A block diagram of a single SpiNNaker node is shown in Fig. 4. The six communications links are used to connect the nodes in a triangular lattice; this lattice is then folded onto the surface of a toroid, as in Fig. 5. Other tilings are obviously possible; this design decision was guided by the pragmatics of assembling the system onto a set of two-dimensional printed circuit boards.

Fig. 6 depicts the individual SpiNNaker die. Each chip contains 18 identical processing subsystems (ARM cores). The die is fabricated by UMC on a 130-nm CMOS process, and was designed using Synopsys, Inc., synthesis tools for the clocked subsystems and Silistix Ltd tools [15] and libraries for the self-timed on-chip and interchip networks.

At start-up, following self-test, one of the processors is elected to a special role as *Monitor Processor*, achieved by a deliberate hardware race, and thereafter performs system management tasks. The other processors are available for application processing; normally, 16 will be used to support the application and one reserved as a spare for fault-tolerance and manufacturing yield-enhancement purposes.

The router is responsible for routing neural event packets both between the on-chip processors and from and to other SpiNNaker nodes. The Tx and Rx interface components (Fig. 4) are used to extend the on-chip Communications NoC (Network-on-Chip) to other SpiNNaker chips. Inputs from the various on and off-chip sources are assembled into a single serial stream which is then passed to the router.

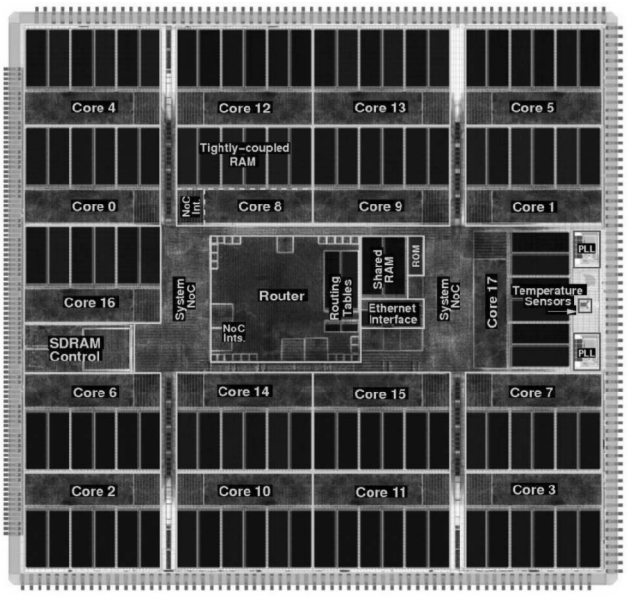


Fig. 6. The SpiNNaker die.

Various resources are accessible from the processor systems via the System NoC. Each of the processors has access to the shared off-die SDRAM, and various system components also connect through the System NoC in order that, whichever processor is the Monitor, it will have access to these components.

4.2 Quantitative Drivers

The SpiNNaker architecture is driven by the quantitative characteristics of the biological neural systems it is designed to model. The human brain comprises in the region of 10^{11} neurons; the objective of the SpiNNaker work is to model 1 percent of this scale, which amounts to a billion neurons. This corresponds approximately to 10 percent of the human cortex, or ten complete mouse brains. Each neuron in the brain connects to thousands of other neurons. The mean firing rate of neurons is below 10 Hz, with the peak rate being 100 s of Hz. These numerical points of reference can be summarized in the following deductions:

$$
\begin{aligned}
&10^9 \text{ neurons, mean fan in/out } 10^3 && \Rightarrow 10^{12} \text{ synapses.}\\
&10^{12} \text{ synapses, } \sim 4 \text{ bytes/synapse} && \Rightarrow 4 \times 10^6 \text{ Mbytes.}\\
&10^{12} \text{ synapses switching at } \sim 10 \text{ Hz} && \Rightarrow 10^{13} \text{ connections/s.}\\
&10^{13} \text{ conn/s, 20 instr/conn} && \Rightarrow 2 \times 10^8 \text{ MIPS.}\\
&2 \times 10^8 \text{ MIPS, } \sim 200 \text{ MHz ARM} && \Rightarrow 10^6 \text{ ARMs.}
\end{aligned}
$$

So 10^9 neurons need 10^6 ARMs, whence:

$$
\begin{aligned}
&1 \text{ ARM at } \sim 200 \text{ MHz} && \Rightarrow 10^3 \text{ neurons.}\\
&1 \text{ node: } 16 \text{ ARM968} + 64 \text{ MB} && \Rightarrow 1.6 \times 10^4 \text{ neurons.}\\
&6 \times 10^5 \text{ nodes, } 1.6 \times 10^4 \text{ neur/node} && \Rightarrow 10^9 \text{ neurons.}
\end{aligned}
$$

The above numbers all assume each neuron has 1,000 inputs. In biology, this number varies from 1 to of the order of 10^5, and it is probably most useful to think of each ARM being able to model about 1 M synapses, so it can model 100 neurons each with 10,000 inputs, and so on.

The system will be inefficient unless there is some commonality across the inputs to the set of neurons modeled on a processor, so that each input event typically connects to tens or hundreds of neurons modeled by a processor. In biology, connections tend to be sparse, so, for example, a processor could model 1,000 neurons each of which connects to a random 10 percent of the 10^4 inputs that are routed to the processor. The standard model assumes sparse connectivity.

4.3 Routing

With a billion neurons a 32-bit address is (more than) sufficient. The AER packets incur a small overhead for control purposes, which amounts to 1 byte in the current design. This is generally transparent to the software running on the ARM cores and exists only while the packet is in transit. Since spike events are unit impulses, all the packet need carry is the control byte and the 32-bit ID of the neuron that fired. SpiNNaker packet formats support an optional 32-bit data payload in addition, but that is not used for neural system modeling directly. The payload will be used for other applications and for debug and diagnostics. Thus, the communication traffic generated by one node is

$$1.6 \times 10^4 \text{ neurons} \times 10 \text{ Hz} \times 5 \text{ bytes} \Rightarrow 0.8 \text{ Mbyte/s.}$$

Each chip incorporates a router that implements AER-based routing of neural spike-event packets. The total traffic from neurons modeled by the processors on the same chip as the router averages 1.6×10^5 packets/s, which is undemanding, although the router also handles incoming and passing traffic:

$$
\begin{aligned}
&\text{4-bit symbols @ 60 MHz/link} && \Rightarrow 6 \times 10^6 \text{ pkts/s}\\
&\text{Six incoming links} && \Rightarrow 3.6 \times 10^7 \text{ pkts/s}
\end{aligned}
$$

So a router operating at 100 MHz processing one packet per clock cycle can easily handle all local, incoming and passing traffic.

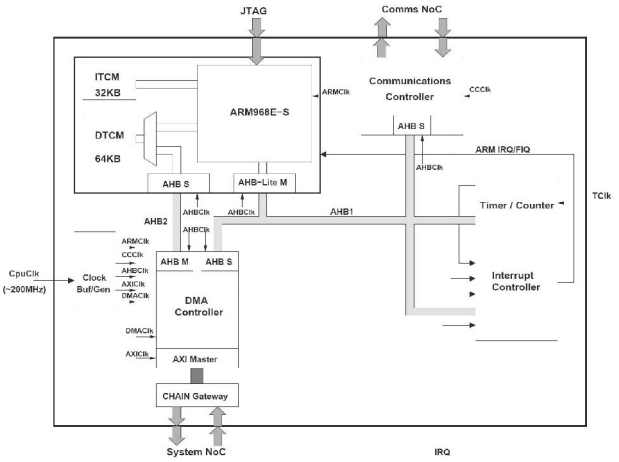


Fig. 7. ARM968 subsystem organization.

4.4 Bisection Bandwidth

If a 57K-node system is organized in such a way that all of the neurons in one-half are connected to at least one neuron in the other half, the traffic across the border from one half to the other is $29\text{K} \times 160\text{K} = 4.6\text{-G packets/s}$. The border is 480 nodes long (assuming a square layout, mapped to a toroid), so each node must carry 10 M packets/s, which is well within the capacity of the router, and 960 links connect the two halves, each carrying 5 M packets/s, which is again within a link's capability.

5 System Components

The routing subsystem, which is a crucial component of SpiNNaker, is described in Section 7.

5.1 ARM968

The ARM subsystem [16] organization is shown in Fig. 7. The system is memory-mapped (see Section 6), and the map for the ARM968 spans a number of devices and buses. The tightly-coupled core-local 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 Advanced High-Speed Bus (AHB—one of the ARM AMBA [17]—Advanced Microcontroller Bus Architecture—interface standards) master interface, which runs at the full processor clock rate. 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 that provides processor access to all memory and other resources on the System NoC.

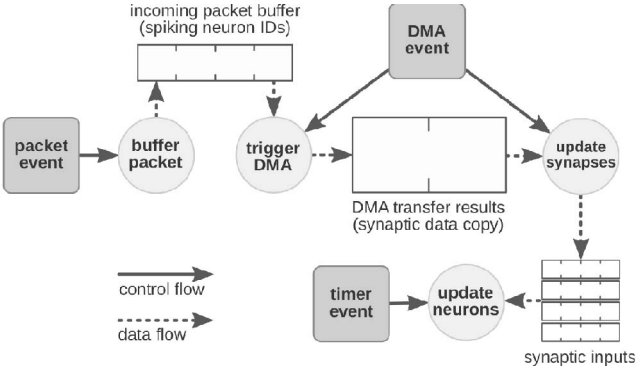


Fig. 8. SpiNNaker event-driven operating mode.

5.2 Vectored Interrupt Controller (VIC)

In standard use, it is envisaged that SpiNNaker will be entirely interrupt-driven [18]. There is no conventional operating system running on the cores, simply a low-level "service provision" system. An interrupt arrives (usually in the form of a message packet); the core wakes to handle it, possibly emitting more packets of its own as a consequence; and then returns to sleep—see Fig. 8.

Each processor node on a SpiNNaker chip has a vectored interrupt controller [19] 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 centralized management of IRQ (standard interrupt) and FIQ (fast interrupt) sources, and offers an efficient indication of the active sources for IRQ vectoring purposes.

The sources of interrupts on SpiNNaker are:

- Communication controller flow-control,
- DMA complete/error/time-out,
- Timers (& watchdog timer),
- Interrupt from another processor on the chip,
- Packet-error from the router,
- System fault,
- Ethernet controller,
- Off-chip signals,
- 32-kHz slow system clock, and
- Software interrupt, for downgrading FIQ to IRQ.

5.3 Counter/Timers

Each node has a counter/timer. This uses the standard AMBA [17] peripheral device, modified in that the Advanced Peripheral Bus interface of the original has been replaced by an AHB interface for direct connection to the ARM968 AHB bus.

The unit provides two independent counters, providing:

- Millisecond interrupts for real-time dynamics,
- Free-running and periodic counting modes,
- Automatic reload for precise periodic timing,
- One-shot and wrapping count modes, and
- The counter clock (which runs at the processor clock frequency) may be prescaled by dividing by 1, 16 or 256.

5.4 DMA Control

Each node includes a DMA controller. The primary application of the DMA subsystem is manually controlled paging. This is used for transferring interneural connection data from the SDRAM in large blocks in response to an input event arriving at an application processor, and for returning updated state information. In addition, the DMA controller can transfer data to/from other targets on the System NoC such as the System RAM and Boot ROM.

As a secondary function the DMA controller incorporates a "Bridge" across which its host ARM968 has direct read and write access to System NoC devices, including the SDRAM. The ARM968 can use the Bridge whether or not DMA transfers are active.

The DMA controller:

- Allows direct pass-through requests from the ARM968,
- Possesses dual buffers supporting simultaneous direct and DMA transfers,
- Provides support for CRC error control in transferred blocks, and
- Provides DMA completion notification interrupts.

DMA transfers are initiated by writing to control registers in the controller. They are executed in the background, and the relevant interrupt caught when the work is complete. Bridge transfers 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. Invisibly 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 signals an error interrupt.

5.5 Ethernet

The SpiNNaker system connects to a host machine via Ethernet links. Every node includes an Ethernet MII (Media Independent Interface), although only a few of the chips are expected to use this. These chips require an external physical connection to a transceiver PHY (PHYsical layer) chip. The interface hardware operates at the frame level: all higher-level protocols are implemented in software running on the local Monitor Processor.

The Ethernet subsystem provides:

- support for full-duplex 10 and 100-Mbit/s Ethernet;
- an outgoing 1.5-Kbyte frame buffer, for one maximum-size frame;
- outgoing frame control, CRC generation, and inter-frame gap insertion;
- incoming 3-Kbyte 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 and reset; and
- interrupt sources for frame-received, frame-transmitted, and PHY (external) interrupt.

TABLE 1
Memory Types

Memory area	Size	Speed/CPU	Visibility
ITCM	32kB	800MBps	Core-local
DTCM	64kB	800MBps	Core-local
SRAM	32kB	25MBps	Node-local
SDRAM	128MB	64MBps	Node-local

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

6 Memory Spaces

Each processor has, directly available to it, four memory spaces, which are mapped onto a single processor-local 32-bit memory space. Processor-local memory is visible only to the core to which it is bound, and that core can use the full available bandwidth. Node-local memory and off-die SDRAM are directly visible to all the cores in a given node, and the available bandwidth is shared between all processors with active accesses.

The memory space access characteristics are summarized in Table 1. Note that although the off-chip SDRAM has higher bandwidth than the on-chip SRAM, it also has higher access latency.

No node has direct visibility of any memory on any other node, except for limited access via the communications fabric to node-local memory on neighboring nodes, and there is no mechanism for maintaining memory coherence of any type across nodes.

The memory map for a single node is shown in Table 2.

The shaded areas in Table 2 represent core-local resources, the others are node-local. The buffered column indicates whether or not the resource is accessed directly (and, therefore, capable of exact recovery if a fault occurs) or via a FIFO (which is faster but not exactly recoverable).

TABLE 2
Single Node Memory Map

Address	Area	Buffered
0x00000000	ITCM	n/a
0x00008000	Not used	n/a
0x00400000	DTCM	n/a
0x00410000	Not used	n/a
0x10000000	Local peripherals - comms, counter, VIC, DMA	mixture
0x50000000	Bus error	n/a
0x60000000	SDRAM	yes
0x70000000	SDRAM	no
0x80000000	Bus error	n/a
0xe0000000	NoC peripherals - router, controller, watchdog, ethernet	yes
0xe5000000	System RAM	yes
0xe6000000	System ROM	yes
0xe7000000	Bus error	n/a
0xf0000000	NoC peripherals	no
0xf5000000	System RAM	no
0xf6000000	System ROM	no
0xf7000000	Bus error	n/a
0xff000000	Boot area	no

7 The Routing Subsystem

The only direct internode communication mechanism in the SpiNNaker engine is via packets. These are launched by cores, and transmitted by hardware to the local node router [20]. There they are redirected as necessary to their target core(s). If these are in the same node as the source, the onward transmission is direct; if a target is a core in another node, the packet is handed out to a physically adjacent node to begin its journey. Each node is only physically directly connected to a handful of neighbor nodes (Fig. 5); a variety of routing techniques ensure that a packet is delivered to the target node.

Packets consist of 40 or 72 bits of data, conveniently broken up into a control byte (8 bits) and one or two data words (1× or 2× 32 bits). The second data word is optional; its presence or absence is signified by a bit in the control byte.

The router is responsible for routing all packets that arrive at its input to one or more of its outputs. It is responsible for routing multicast neural event packets (which it does through an associative multicast router subsystem); point-to-point (P2P) packets (for which it uses a look-up table); nearest neighbor packets (using a simple algorithmic process); fixed-route (FR) packet routing (where the route is defined in a register); default routing (when a multicast packet does not match any entry in the multicast router), and emergency routing (when an output link is blocked due to congestion or hardware failure). Various error conditions are identified and handled by the router, for example, packet parity errors, time-out, and output link failure.

The sheer physical size of the system makes global clock synchronization a virtual impossibility. This is sidestepped by simply doing away with the requirement altogether [22]. The asynchronous interconnect infrastructure allows arbitrary clock skew (although not clock drift) across the system.

7.1 Packet Taxonomy

SpiNNaker packets are of four types: nearest neighbor (NN), point-to-point, multicast, and fixed route. The passage of each through the router subsystem is brokered by hardware; nominally, the latency of a packet transfer through a router is 0.1 μs, irrespective of its type, source or destination.

7.1.1 Nearest Neighbor Packets

The layout of an NN packet is shown in Fig. 9a. The control byte contains a packet-type bit-pair (bits[7:6] = 10 for NN packets), a "peek/poke" or "normal" type indicator (T)—see below, routing information (where the packet should be sent), a payload indicator (the presence or absence of the second data word), and error detection (parity) information.

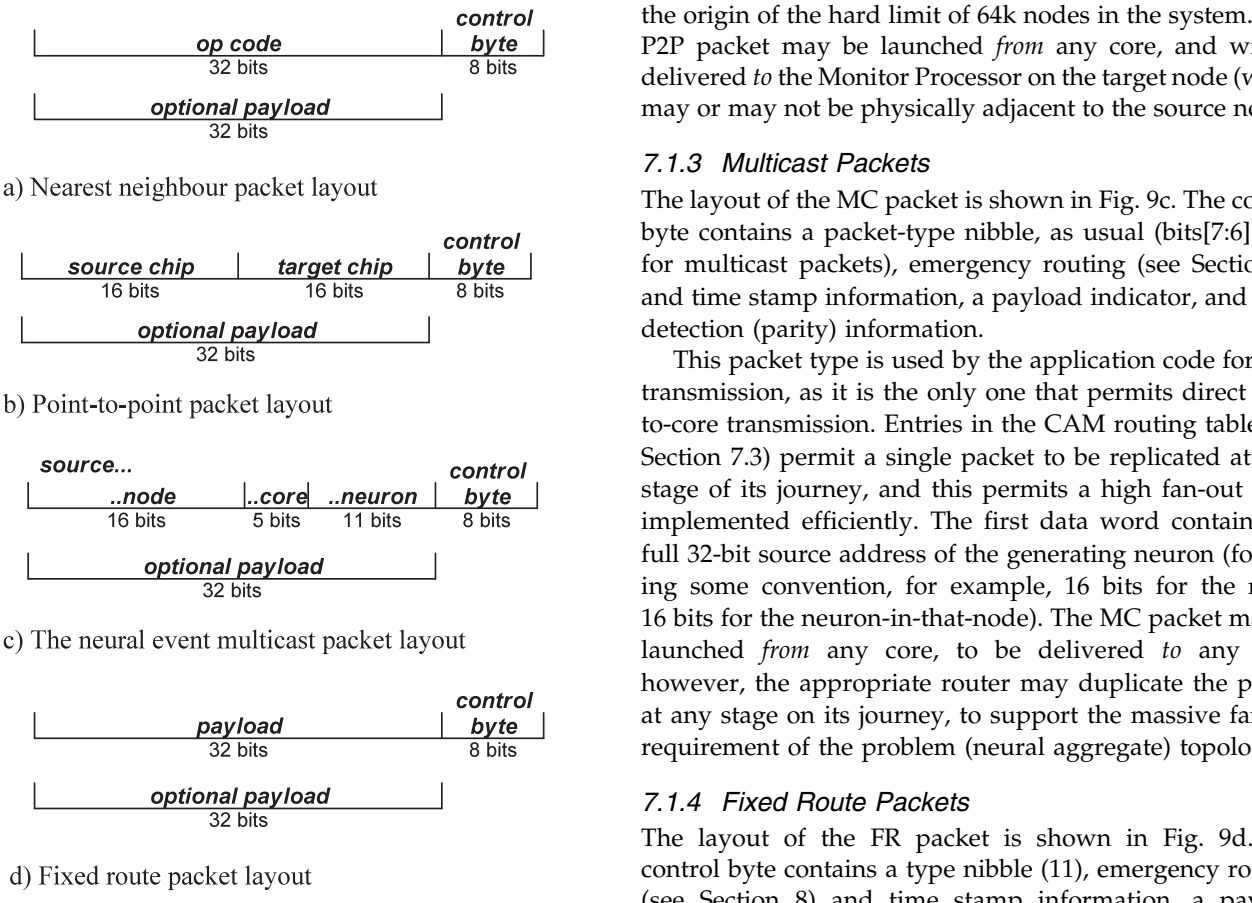


Fig. 9. Packet layouts.

The routing nibble—3 bits—decodes to eight choices: which of the six physical ports is to be used, OR the packet is to be duplicated and sent from all or a register-defined subset of all six simultaneously, OR the packet is to be directed to the local Monitor core.

The T bit indicates whether the packet is a normal packet or a special type known as *peek-poke*. The usage of this facility is discussed in Section 8.

The NN packet may be launched *from* any core; it will only be delivered *to* the Monitor core on a physically adjacent node (or its own Monitor). Monitors may talk to themselves.

7.1.2 Point-to-Point Packets

The layout of a P2P packet is shown in Fig. 9b. The control byte contains a packet-type nibble (bits[7:6] = 01 for P2P packets), a sequence code (used for multipart messages), time stamp, a payload indicator (as above), and error detection (parity) information.

The system has a coarse global time phase that cycles through the sequence $00, 01, 11, 10, 00, \ldots$ Global synchronization must be accurate to within 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 $\text{launched} = 11$) it will drop it to the local Monitor Processor. This provides a rudimentary garbage collection mechanism.

The first data payload word (Fig. 9b) is used to carry two 16-bit values—the source and target node addresses. (This is the origin of the hard limit of 64k nodes in the system.) The P2P packet may be launched *from* any core, and will be delivered *to* the Monitor Processor on the target node (which may or may not be physically adjacent to the source node).

7.1.3 Multicast Packets

The layout of the MC packet is shown in Fig. 9c. The control byte contains a packet-type nibble, as usual (bits[7:6] = 00 for multicast packets), emergency routing (see Section 8), and time stamp information, a payload indicator, and error detection (parity) information.

This packet type is used by the application code for data transmission, as it is the only one that permits direct core-to-core transmission. Entries in the CAM routing table (see Section 7.3) permit a single packet to be replicated at each stage of its journey, and this permits a high fan-out to be implemented efficiently. The first data word contains the full 32-bit source address of the generating neuron (following some convention, for example, 16 bits for the node, 16 bits for the neuron-in-that-node). The MC packet may be launched *from* any core, to be delivered *to* any core; however, the appropriate router may duplicate the packet at any stage on its journey, to support the massive fan-out requirement of the problem (neural aggregate) topology.

7.1.4 Fixed Route Packets

The layout of the FR packet is shown in Fig. 9d. The control byte contains a type nibble (11), emergency routing (see Section 8) and time stamp information, a payload indicator, and error detection (parity) information. The packet provides a programmable "fast track" from wherever it is launched, usually to the nearest Ethernet-enabled node; all 64 bits of payload are at the disposal of the application programmer.

7.2 Initializing the Router

Each node has a router; there are up to 65,536 of them. Setting up the data tables necessary to drive the routing is a nontrivial task, in terms of both complexity and data size. Configuring the system to simulate a large neural aggregate takes place as a sequence of steps:

Step 0. On power-up the NN routing capability is immediately available. Each node is physically connected to six adjacent neighbors, and the internal port addresses of each link are known. For the sake of clarity, we ignore the issue of hardware failures here.

Step 1. The P2P routing tables need to be defined. Fig. 10 shows a very simple SpiNNaker mesh, consisting of a flat 3×3 grid of nodes, *not* connected as a torus. The nodes are labeled 0-8, the ports on each node a-f. The figure shows the P2P tables for nodes 0, 1, 2, and 5. To send a P2P packet from node 5 to node 0, say, requires the following:

- The P2P packet is launched from a core on node 5, and sent to the node 5 router. The packet will be directed out of port e on node 5.
- The packet will arrive via port b on node 1, and the node 1 router will redirect it to port f.
- The packet will arrive via port c on node 0, and be directed to the Monitor Processor.

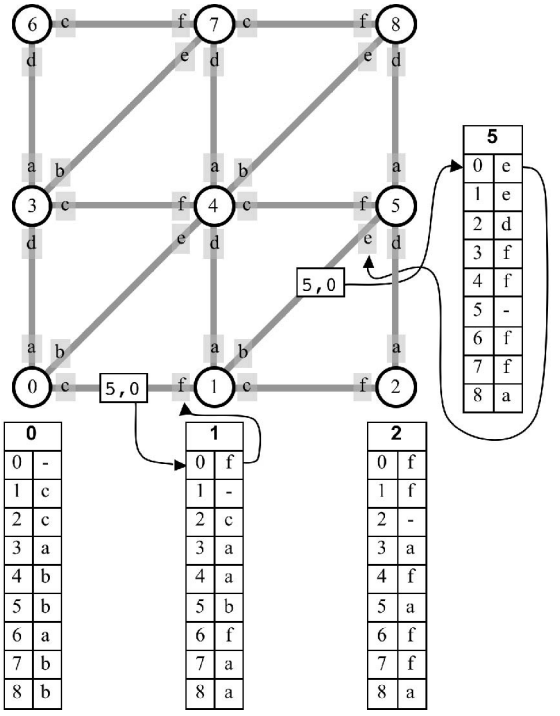


Fig. 10. Propagation of P2P packets.

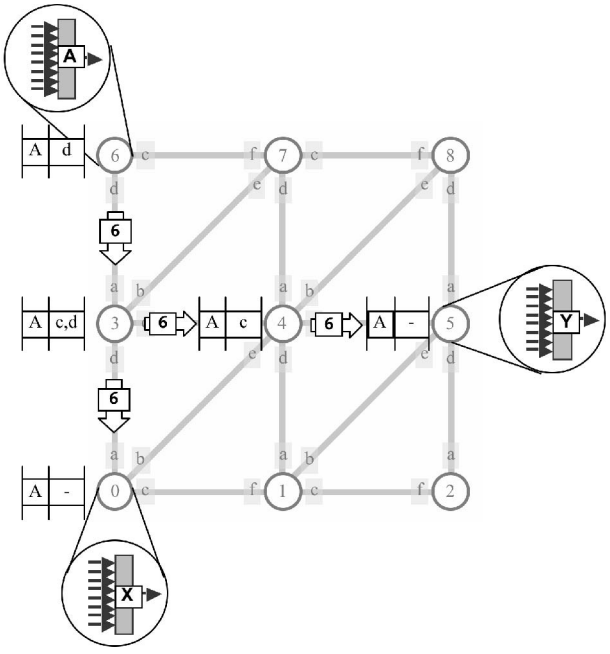


Fig. 11. Propagation of MC packets.

It follows that in a full 65,536-node SpiNNaker engine, with each node containing a 65,536-entry table, some 4×10^9 table entries have to be derived.

These table entries are derived internally by boot code, and define a distributed definition of the working node mesh topology. Note there is no requirement for the P2P route from node X to node Y to be the inverse of that from Y to X.

Step 2. It is now necessary to map the problem graph—the neuron topology—onto the working node mesh. This is done by a combination of techniques taken from the world of design automation; *prima facie* a simple mapping problem is made extremely hard by the sheer size of the data sets we are forced to confront: 10^9 neurons, with an average fan-out of 10^3. Even the most brutally simplistic of data structures requires over 4-Tbytes simply to store the definition.

Step 3. Once the mapping has been achieved (1,000 neurons to each core), it is possible to define the MC routing tables. This again is a nontrivial task, but a (much simplified) illustration of the table structure necessary is shown in Fig. 11.

The neural circuit consist of three neurons; A excites X and Y. A has been mapped onto node 6, X to node 0, and Y to node 5. Recall that the MC packets are labeled using address event representation—they contain only the originating neuron identification. The relevant MC table entries are shown. In node 6, an event from source neuron A will be routed out of port d. Thus, it will be received via port a on node 3, whence it will be duplicated and copies sent out of ports c and d. The packet out of port d will arrive via port a on node 0, and hence to its target neuron. The copy sent to node 4 will be forwarded to node 5.

The table entries are derived partly internally and partly externally, and define a distributed version of the problem graph definition.

7.3 Router Internals

The P2P routing hardware is fairly conventional, and will not be detailed here.

The organization of the MC packet router is shown in Fig. 12. The 32-bit source key (source neuron) is input to a 1,024 × 32-bit tristate (0, 1, X) CAM—in general, multiple hits will be both possible and common. These hits are written to a 1,024 × 1-bit hit register. All but the most significant single bit in this register are discarded, and this single remaining bit treated as a 1,024 bit 1-hot and passed into an address encoder. This generates a 10-bit binary

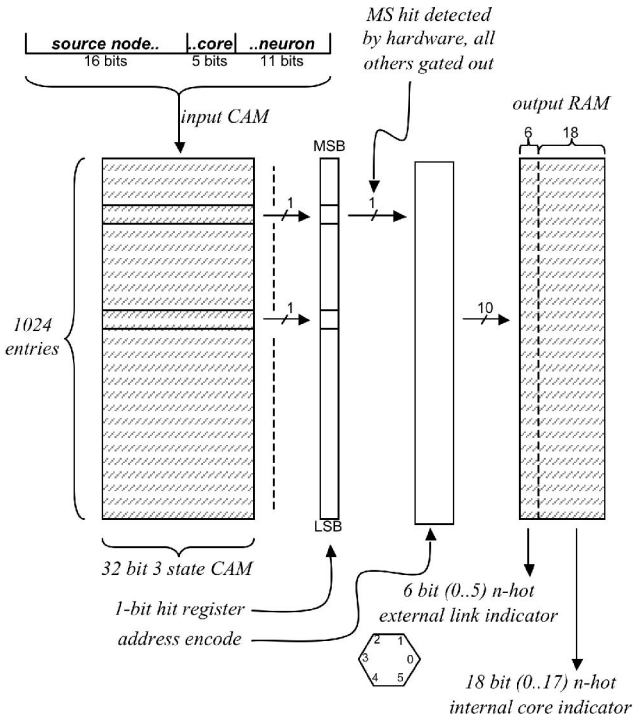


Fig. 12. Internal structure of the MC table router.

equivalent, which drives a 1,024 × 24 bit lookup RAM. (This is a RAM, not a ROM, because it needs at least to be programmed at load time, and the ultimate intention is to be able to change it on the fly.) The 24-bit word so generated consists of a 6-bit nibble and an 18-bit nibble. The 6-bit nibble represents a 6-hot external link indicator (0-5) to which the packet is forwarded (for example, 010110 would cause the packet to be routed to external links 1, 2, and 4). The 18-bit nibble represents an 18-hot internal core address (0-17) to which the packet will be forwarded, triggering an interrupt as it arrives. (For example, 001000100100000000 will cause packets to go to cores 8, 11, and 15 on the current chip.) It is easy to see how packets may be duplicated by this mechanism.

(The 1,024 × 32-bit tristate CAM may be thought of as a 1,024 × 32-bit binary CAM and a 1,024 × 32-bit binary RAM. In the actual implementation, the RAM simply holds a bit mask indicating the position of the "do not cares" in the CAM.)

7.4 Networks-on-Chip

SpiNNaker contains two NoCs: A Communications NoC (used to handle on and off-chip interprocessor communication) and a System NoC (used to handle on-chip processor-to-memory and processor-to-peripheral communication). Both of these use delay-insensitive (DI) asynchronous logic. DI communication makes no assumptions about gate and interconnect delays (except that they be finite). This makes it extremely robust in the face of timing issues that would probably defeat a synchronous system. Further, once designed, it is not necessary to perform any timing validation on the layout, because the physical geometry has no effect. The price paid for this is that the protocols require extra information to be embedded in the signaling to signify data validity, requests, and acknowledge, so more physical wires are required, though data transmission energy costs can be lower due to the absence of a high-speed clock and careful choice of data encoding to minimize the number of transitions used by the communication protocol.

7.4.1 System NoC

The System NoC was developed using tools from Silistix Ltd [15] that generate the self-timed fabric to meet the bandwidth requirements for communication between each client-pair. Data are transmitted through multiple parallel channels, each of which carries 4-bit data encoded in 3-of-6 RTZ (Return To Zero) form.

The major requirement on the System NoC is to allow up to 16 active application processors to share the 1 Gbyte per second available SDRAM bandwidth equitably, while providing independent access (via a cross-bar organization) for the Monitor Processor to other system resources.

7.4.2 Communication NoC

The communications NoC carries packets between the processors on the same or different chips. It plays a central role in the system architecture—Fig. 4. The NoC can be cleanly divided into two sections—input and output.

The input half of the communication NoC is shown in Fig. 13. The structure is a tree arbiter which merges the various sources of packets (internode links and in-node cores) into a single stream, for input to the router. The flow rate of data at each stage is maintained by doubling the available bandwidth each time two streams merge.

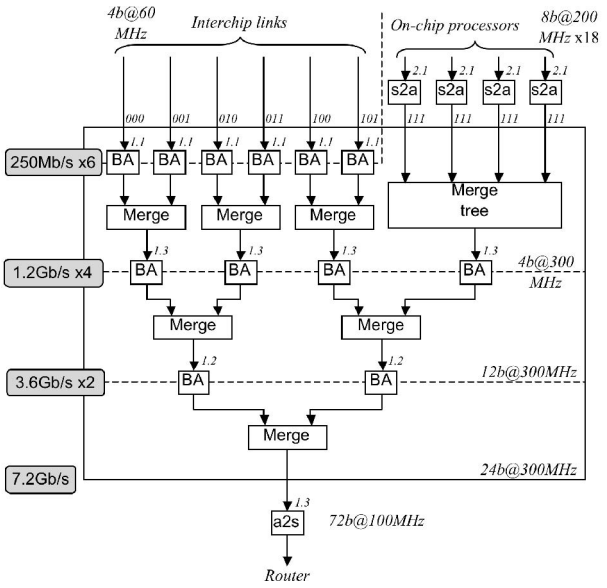


Fig. 13. Input section of the communications NoC.

The output half of the NoC simply transforms the data protocol of the router output into the DI NRZ (Non Return to Zero) protocol needed to drive the output ports, and/or sends packets back to the communications controllers of the node-local cores.

The on-chip sections of the Communication NoC use the same 3-of-6 RTZ protocol as the System NoC, but the interchip links use a 2-of-7 NRZ protocol. This protocol change was chosen to minimize the power consumed in the large PCB track capacitances. The 2-of-7 NRZ protocol sends 4 bits with three signal transitions (including the acknowledge), whereas the 3-of-6 RTZ protocol uses eight signal transitions to send 4 bits.

8 Fault Tolerance

As computing architectures move inexorably toward the ExaFLOP regime and beyond, failure rates will become an important—if not dominant—design concern [21]. For example, IBM cite a failure rate of 0.02 faults/month/TF on a BlueGene machine, which scales to around 1 fault/minute on an ExaFLOP system [23].

The complete SpiNNaker engine contains around 57,000 nodes, 350,000 internode communication links, and around 7 TBytes of memory. The ability of the system to degrade gracefully in the face of point failures in the underlying hardware has been considered at all levels of both the software and hardware design—the discussion here is limited to the hardware aspects, and divided into three sections: Fault insertion (made easy by the interrupt-driven nature of the system), fault detection, and fault isolation.

8.1 Processors

Insertion. Any core can be disabled by the Monitor Processor, and software can be used to corrupt the RAM to model soft errors.

Detection. The node watchdog will catch rogue software and periodic self-test interrupt handlers can be run.

Isolation. Individual ARMs can be locked out (but not individual parts of the memory subsystems).

8.2 Interrupt Controller

The sensitivity of the node to errors in this functional block is high; it is hard to see how it can continue to function in the face of most errors here, so the entire node will have to be locked out.

Insertion. The vector locations can be trivially corrupted.

Detection. It is—philosophically—almost impossible to tell when an interrupt handler has been invoked erroneously, though the node watchdog will trap runaway behavior.

Isolation. Failed vector locations can be mapped to unused interrupt sources.

8.3 Counter/Timers

Insertion. A counter can be disabled (stopped) from outside.

Detection. The two counter/timer systems can periodically check the calibration of each other as part of the maintenance/self-test cycle.

Isolation. Outside control allows the counter to be disabled, its output interrupt signal inhibited and the handler disconnected.

8.4 DMA

As with the interrupt controller, it is difficult to see how the node can continue with faults here. The off-die SDRAM is memory mapped, so *in principle*, operation could continue although the memory access to the SDRAM would be some two orders of magnitude slower.

Insertion. Software can introduce bit patterns in the SDRAM that will cause DMA CRC errors.

Detection. CRC error detection is built into the hardware, and the transfer can be timed out.

Isolation. Not, in general, a viable option. The node must be closed down.

8.5 Packet Communications

The packet communication infrastructure has error detection and recovery built in at several levels.

8.5.1 Nearest Neighbor Peek-Poke

Nearest-neighbor packets are used to initialize the system and to perform runtime flood-fill and debug functions. In addition, the "peek/poke" form of NN packet can be used by neighboring systems to access System NoC resources. Here, an NN poke "write" packet (which is a peek/poke 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 peek/poke 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 neighbor that issued the original packet, using the Rx link ID to identify that source. This "peek/poke" access to the principal resources of a neighboring node can be used to investigate a nonfunctional chip, to reassign the Monitor Processor from outside, and obtain visibility into a chip for test and debug purposes.

8.5.2 Low Level Error Control

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

Hardware. The blocked link will be detected (in hardware) and subsequent packets rerouted via the other two sides of a triangle of which the suspect link was an edge, being initially rerouted via the link which is rotated one link clockwise from the blocked link (so if link Tx0 fails, link Tx5 is used).

Software: The Monitor Processor will be informed. It can track the problem using a 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; and
- if the problem appears permanent, it will 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; and
- means of inducing the various types of fault for testing purposes.

Emergency rerouting around the triangle requires additional emergency packet types for MC and FR packets. (P2P packets find their own way to their destination following emergency routing.)

These packet types use the "emergency routing" nibbles within the control byte to control emergency routing around a failed or congested link:

- 00—normal 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.

One consequence of this is that—in theory—it is possible for packets to be delivered out of order. Note that there is no software overhead for any of this; the entire operation is brokered by hardware and is transparent to the sending and receiving nodes.

8.5.3 Communication Router

The communications router has some internal fault-tolerance capacity; in particular, it is possible to map out a failed multicast router entry. This is a useful mechanism as the multicast router dominates the silicon area of the communication system.

There is also capacity to cope with external failures. Emergency routing will attempt to bypass a faulty or blocked link, however, in the event of a node (or larger) failure this will not be sufficient. To tolerate a chip failure, several expedients can be employed on a local basis:

- P2P packets can be routed around the obstruction;
- MC packets with a router entry can be redirected appropriately. In most cases, default MC packets cannot sensibly be trapped by adding table entries due to their (almost) infinite variety. To allow rerouting, these packets can be dropped to the Monitor on a link-by-link basis using a "diversion register." In principle, they can then be routed around the obstruction as P2P payloads before being resurrected at the opposite side. Should the Monitor Processor become overwhelmed, it is also possible to use the diversion register to eliminate these packets in the router; this prevents them blocking the router pipeline while waiting for a time-out and, thus, delaying viable traffic.

Detection. Packet parity errors, packet time-phase errors, wrong packet length: these are all detected by the hardware and cause—usually—the errant packet to be dropped to the Monitor Processor.

Isolation. A multicast router entry can be disabled if it fails.

Since all multicast router entries are identical, the function of any entry can be relocated to a spare entry. If a router becomes full, a global reallocation of resources can move functionality to a different router, although this is a nontrivial exercise.

8.6 Interchip Communication

The fault inducing, detecting, and resetting functions are controlled from the System Controller. The on and off-chip interfaces are "glitch hardened" to greatly reduce the probability of a link deadlock arising as a result of a glitch on one of the interchip wires. Such a glitch may introduce packet errors, which will be detected and handled elsewhere, but it is very unlikely to cause deadlock. It is expected that the link reset function will not be required often.

Insertion. An input controlled by the System Controller causes the interface to deadlock.

Detection. Monitor Processors should regularly test link functionality.

Isolation. The interface can be disabled to isolate the chip-to-chip link. This input from the system controller is also used to create a fault.

9 Physical Assembly

9.1 Topology and Geometry

With six physical links available, a variety of topologies are possible. The dominant goal in selecting a topology is the desire to keep the routing as short and isotropic as possible. A 3-cube is clearly the most desirable from this perspective, but mapping this to a two-dimensional PCB-based physical geometry would have introduced significant disparities in the length of the individual links. The hexagonal tiling wrapped onto the torus of Fig. 5 is as good a configuration as any.

9.2 Power Distribution

The chips are mounted in a 48-node hexagonal array on a double height Eurocard, with 24 cards per rack. The rack is supplied with mains AC, a rack-local supply generates 12-V DC, which is distributed to each board; finally, local down-regulators are mounted on each board to provide the 1.8 and 1.2-V DC needed for each chip and 3.3 V for other board services. The boards within a rack are interconnected via high-speed serial cables, allowing an individual rack to be configured as a 1,152-node torus. Virtually any number of racks may be interconnected to form a system of arbitrary scalability.

9.3 Power Dissipation

We budget for the nodes dissipating up to 1 W, and with other components a board will dissipate up to 75 W. Scaling this up to the full million-core machine indicates a total power budget of around 90 kW—this is just within the range of off-the-shelf forced air-cooling systems.

Each node is provided with three temperature sensors that can be used to moderate the local clock speed if the die temperature rises too high.

10 Software

Programming the SpiNNaker engine is unlike most other computing machinery. The small core-local memory—and the mode of operation—make the idea of an operating system (in the general sense of the word) inapplicable. A low level of "service-providing" software runs on each core; thereafter, the entire system executes as a sequence of choreographed interrupts. A model that is probably helpful to envisage the behavior of the system is not that of a massively parallel computing engine, rather as a hardware accelerator attached to a conventional host (or hosts), the internals of which are programmable.

Users who wish to model systems of spiking neurons on SpiNNaker can define their network in a high-level neural network description language such as PyNN [24]. Software tools have been developed that map PyNN descriptions onto SpiNNaker for real-time execution.

Although at the time of writing only small SpiNNaker systems are operational, results so far are promising, and real-time spiking neural network controllers have been demonstrated, for example, in simple robotics tasks.

11 Final Comments

From one perspective, SpiNNaker is "just another" supercomputer (cluster). However, it is significantly different, in many ways:

- It is constructed from medium-performance components (200 MHz ARM9 cores).
- The total development and construction budget to date is UK£5M (~US$8M).
- At $2 \times 10^8/90 \times 10^3 = 2{,}200$ MIPS/W it deserves an honourable place in the Green500 supercomputer rankings.[1] (Compare this with the human brain:

1. As of November 2011, the top five computers in the Green500 table [http://www.green500.org] are all IBM BlueGene/Q machines with efficiencies close to 2,000 MFLOPS/W.

10^{15} synapses at 10 Hz dissipating 25 W => 400M MOPS/W, though a synapse "operation" is much more than a 32-bit instruction.)

- SpiNNaker has no hardware floating-point support. In retrospect, although the use of fixed-point arithmetic is more energy efficient, it leads to programming difficulties, and on balance this may be the wrong approach.
- The design principles explicitly disregard three of the most significant axioms of conventional supercomputer engineering: memory coherence, synchronization, and determinism.

Designing software to run on a large system with no conventional operating system, nondeterministic communications, and almost no internal debug or visibility capability requires new techniques and thinking to be developed at numerous levels; these will be described in future publications.

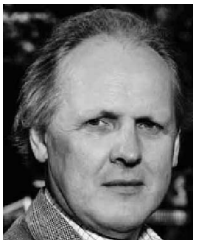
Finally, we note that, although the SpiNNaker design trajectory was originally inspired by biology, and neurological simulation remains the flagship objective, the architecture is elegantly suited to a wide variety of nonbiological applications. Suitable problems are those that can be transformed into a mesh-based representation, where the dominant node interaction is (or can be transformed into) one defined by each processor having one or more arbitrary graphs that connect its outputs to many logical neighbors (physical nearest-neighbor connectivity being a very simple example), and the problem can be recast as a global relaxation, where the solution trajectory is unimportant and only the steady-state solution corresponds to physical reality. Examples include finite elements, molecular modeling (protein folding), and discrete system simulation. These issues will also be discussed in future publications.

Acknowledgments

This work was supported by the UK Engineering and Physical Sciences Research Council (under EPSRC grant EP/G015740/1), with industry partners ARM Ltd, Silistix Ltd and Thales.

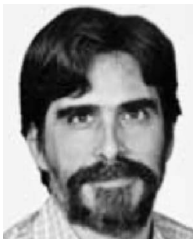
References

[1] SpiNNaker Project Website : http://apt.cs.man.ac.uk/projects/SpiNNaker/, 2013.
[2] L.A. Plana, D. Clark, S. Davidson, S. Furber, J. Garside, E. Painkras, J. Pepper, S. Temple, and J. Bainbridge, "SpiNNaker: Design and Implementation of a GALS Multi-Core System-on-Chip," *ACM J. Emerging Technologies in Computing Systems,* vol. 7, no. 4, article 17, pp. 17:1-17:18, Dec. 2011.
[3] X. Jin, M. Lujan, L.A. Plana, S. Davies, S. Temple, and S. Furber, "Modeling Spiking Neural Networks on SpiNNaker," *Computing in Science & Eng.,* vol. 12, no. 5, pp. 91-97, Sept./Oct. 2010.
[4] S.B. Furber and S. Temple, "Neural Systems Engineering," *J. The Royal Soc. Interface,* vol. 4, no. 13, pp. 193-206, Apr. 2007, doi:10.1098/rsif.2006.0177.
[5] B. Pakkenberg, D. Pelvig, L. Marner, M.J. Bundgaard, H.J.G. Gundersen, J.R. Nyengaard, and L. Regeur, "Aging and the Human Neocortex," *Experimental Gerontology,* vol. 38, nos. 1/2, pp. 95-99, Jan./Feb. 2003.
[6] D. Purves, G.J. Augustine, D. Fitzpatrick, W.C. Hall, A.-S. LaMantia, J.O. McNamara, and L.E. White, eds., *Neuroscience,* fourth ed. Sinauer Assoc. 2008.
[7] C.U.M. Smith, *Elements of Molecular Neurobiology,* third ed. Wiley, 2002.
[8] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision.* Kluwer Academic Publishers, 1994.
[9] M. Sivilotti, "Wiring Considerations in Analog VLSI Systems, with Application to Field-Programmable Networks," PhD dissertation, California Inst. of Technology, Pasadena, CA, 1991.
[10] H. Markram, "The Blue Brain Project," *Nature Rev. Neuroscience,* vol. 7, pp. 153-160, Feb. 2006, doi:10.1038/nrn1848.
[11] IBM Blue Gene Team, "Overview of the IBM Blue Gene/P Project," *IBM J. Research and Development,* vol. 52, nos. 1/2, pp. 199-220, Jan. 2008.
[12] R. Ananthanarayanan, S.K. Esser, H.D. Simon, and D.S. Modha, "The Cat Is Out of the Bag: Cortical Simulations with 10^9 Neurons and 10^{13} Synapses," *Proc. ACM/IEEE Conf. Supercomputing,* pp. 1-12, 2009.
[13] E.M. Izhikevich, "Simulation of Large-Scale Brain Models," www.nsi.edu/users/izhikevich/interest/index.htm, 2005.
[14] E.M. Izhikevich and G.M. Edelman, "Large-Scale Model of Mammalian Thalamocortical Systems," *Proc. Nat'l Academy of Sciences of USA,* vol. 105, no. 9, pp. 3593-3598, Feb. 2008, doi: 10.1073/pnas.0712231105.
[15] J. Bainbridge, "The CHAIN Works Tool Suite: A Complete Industrial Design Flow for Networks-on-Chips," *Networks-on-Chips: Theory and Practice,* F. Gebali, H. Elmilgi, and M.W. El-Kharashi eds., pp. 281-306, Taylor & Francis, Inc., 2009.
[16] ARM968E-S Tech. Ref. Manual, ARM DDI 0311C, 2004.
[17] AMBA Design Kit Tech. Ref. Manual, ARM DDI 0243A, 2003.
[18] T. Sharp, L.A. Plana, F. Galluppi, and S.B. Furber, "Event-Driven SpiNNaker Simulation," *Proc. Int'l Conf. Neural Information Processing (ICONIP '11),* pp. 424-430, 2011.
[19] ARM PL190 Tech. Ref. Manual, ARM DDI 0181E, 2004.
[20] J. Wu and S.B. Furber, "A Multicast Routing Scheme for a Universal Spiking Neural Network Architecture," *The Computer J.,* vol. 53, no. 3, pp. 280-288, 2010, doi:10.1093/comjnl/bxp024.
[21] I. Koren and C. Krishna, *Fault-Tolerant Systems.* Morgan Kaufmann, 2007.
[22] L.A. Plana, S.B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS Infrastructure for a Massively Parallel Multiprocessor," *IEEE Design & Test of Computers,* vol. 24, no. 5, pp. 454-463, Sept. 2007, doi:10.1109/MDT.2007.149.
[23] L.D. Solano-Quinde and B.M. Bode, "Module Prototype for Online Failure Prediction for the IBM Blue Gene/L," *Proc. IEEE Electro/Information Technology Conf. (EIT '08),* pp. 470-474, May 2008, doi:10.1109/EIT.2008.4554349.
[24] A.P. Davison, D. Brüderle, J.M. Eppler, J. Kremkow, E. Muller, D.A. Pecevski, L. Perrinet, and P. Yger P, "PyNN: A Common Interface for Neuronal Network Simulators," *Frontiers Neuroinformatics,* vol. 2, pp. 1-10, 2008, doi:10.3389/neuro.11.011.2008.



Steve B. Furber (M'98-SM'02-F'05) received the BA degree in mathematics in 1974 and the PhD degree in aerodynamics in 1980 from the University of Cambridge, United Kingdom. He was in the R&D Department at Acorn Computer Ltd. from 1981 to 1990, and was a principal designer of the BBC Micro and the ARM 32-bit RISC microprocessor, and moved to his current position as a ICL professor of computer engineering at the University of Manchester, United Kingdom, in 1990. He is a fellow of the Royal Society, the Royal Academy of Engineering, the BCS, the IET, and the IEEE.

David R. Lester received the MA degree in mathematics from the University of Oxford, United Kingdom, in 1983, the MSc degree in computer science from University College, London, in 1984, and the DPhil in computer science from the University of Oxford, United Kingdom, in 1988. From 1988 to 1990, he worked with GEC-Marconi on functional language compilers for parallel machines, validating communication protocols on transputer hardware. He moved to the University of Manchester as a lecturer in 1990. He is a member of the IET.

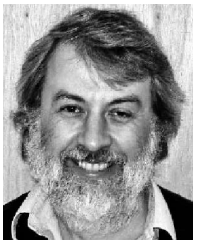


Luis A. Plana (M'97-SM'07) received the Ingeniero Electrónico degree from Universidad Simón Bolívar (Venezuela) in 1978, the MSc degree in electrical engineering from Stanford University in 1984, and the PhD degree in computer science from Columbia University in 1998. He was at Universidad Politécnica, Venezuela, for over 20 years, where he was a professor and the head of the Department of Electronic Engineering. Currently, he is a research fellow in the School of Computer Science at the University of Manchester. He is a senior member of the IEEE.

Jim D. Garside received the BSc degree in physics in 1983 and the PhD degree in computer science in 1987 from the University of Manchester, United Kingdom. His doctoral work looked at digital signal-processing architectures. He spent some time working on parallel architectures with Transputers. He joined the University of Manchester as a lecturer in 1991; his research work has primarily been concerned with VLSI technology, particularly with the Amulet asynchronous ARM processors and, more recently, with SpiNNaker.

Eustace Painkras received the BTech degree from Kerala University and the MS degree from State University of New York at Stony Brook. She is a research fellow in the School of Computer Science at the University of Manchester. Her research interests include VLSI design, wireless communications systems, reliable and power-aware computer architectures.

Steve Temple received the BA degree in computer science in 1980 and the PhD degree for research into local area networks in 1984 from the University of Cambridge, United Kingdom. He was subsequently employed as a research fellow at the University of Cambridge Computer Laboratory. He was a self-employed computer consultant from 1986 to 1993 when he took up his current post of research fellow in the School of Computer Science at the University of Manchester.



Andrew D. Brown (M'90-SM'96) received the first degree in physical electronics in 1976, and the PhD degree in microelectronics in 1981 from Southampton University, United Kingdom. He has been a member of academic staff at Southampton since 1980. He spent time at IBM Hursley Park United Kingdom as an IBM visiting scientist in 1983, at Siemens NeuPerlach (Munich) as a visiting professor in 1989, and at Multiple Access Communications Ltd as part of the Senior Academics in Industry scheme in 1994. He was appointed to an Established chair of electronics at Southampton University in 1999. In 2000, he spent a sabbatical at LME Design Automation working on cryptographic synthesis; in 2002, he was awarded a Royal Society industrial fellowship for two years; in 2004, he spent six months in Trondheim (Norway), and in 2008 in Cambridge (United Kingdom), as a visiting professor. He is a fellow of the IET and the BCS, a senior member of the IEEE, a chartered engineer, and a registered European engineer.