SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation

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

***Abstract*—The modelling of large systems of spiking neurons is computationally very demanding in terms of processing power and communication. SpiNNaker—Spiking Neural Network architecture—is a massively parallel computer system designed to provide a cost-effective and flexible simulator for neuroscience experiments. It can model up to a billion neurons and a trillion synapses in biological real time. The basic building block is the SpiNNaker Chip Multiprocessor (CMP), which is a custom-designed globally asynchronous locally synchronous (GALS) system with 18 ARM968 processor nodes residing in synchronous islands, surrounded by a lightweight, packet-switched asynchronous communications infrastructure. In this paper, we review the design requirements for its very demanding target application, the SpiNNaker micro-architecture and its implementation issues. We also evaluate the SpiNNaker CMP, which contains 100 million transistors in a 102-mm^2 die, provides a peak performance of 3.96 GIPS, and has a peak power consumption of 1 W when all processor cores operate at the nominal frequency of 180 MHz. SpiNNaker chips are fully operational and meet their power and performance requirements.**

***Index Terms*—Asynchronous interconnect, chip multiprocessor, energy efficiency, globally asynchronous locally synchronous (GALS), network-on-chip, neuromorphic hardware, real-time simulation, spiking neural networks (SNNs).**

I. Introduction

SPINNAKER [1] is a biologically inspired, massively parallel computing engine designed to facilitate the modelling and simulation of large-scale spiking neural networks of up to a billion neurons and a trillion synapses (inter-neuron connections) in biological real time. It is a generic and programmable platform for neuroscientists, psychologists, and brain researchers to explore brain functions with software neuronal models.

SpiNNaker is designed as a large array of nodes, each containing a chip multiprocessor (CMP) die and a 128-MB SDRAM die stacked, stitch-bonded together, and housed in a single 300-pin BGA package (see Fig. 1). The CMP contains 18 ARM968 processing cores, each capable of simulating up to 1000 spiking neurons. The architecture scales from a single chip, in its smallest configuration, to a system of 65 536 chips, although the full-size SpiNNaker system will contain 57 600 chips with 1 036 800 processors, delivering peak processing power exceeding 228 Dhrystone TIPS.

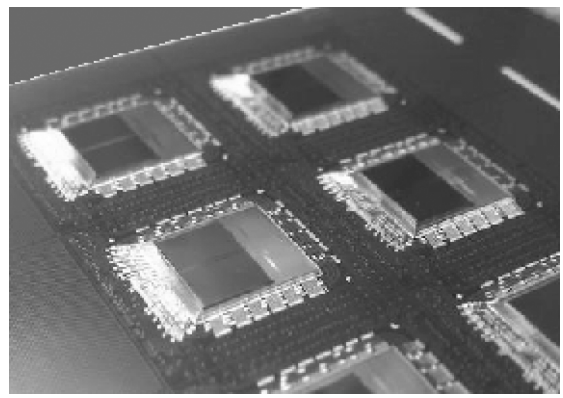


Fig. 1. SpiNNaker CMP and SDRAM dies stitch-bonded together.

The CMP micro-architecture design assumes that processors are free: the real cost of computing is energy. In a massively parallel computer, there are costs associated with keeping the devices within the correct temperature window and, more importantly, the cost of the energy used to operate the machine over its lifetime can be significantly larger than that of the machine itself. This is why we use energy-efficient ARM9 embedded processors and mobile double data rate (DDR) SDRAM, in both cases trading off some performance for greatly enhanced power efficiency. Additionally, inter-chip communication uses self-timed channels which, although costly in wires, are significantly more power-efficient than synchronous links of similar bandwidth. We estimate that a full-size million-core SpiNNaker will consume around 90 kW, which is just within the range of off-the-shelf forced-air cooling systems.

Inter-processor communication is based on an efficient multicast infrastructure inspired by neurobiology. It uses a packet-switched network to emulate the very high connectivity of biological systems. The packets are source-routed, i.e., they carry only information about the packet issuer; the network is responsible for delivering them to their destinations. The heart of the communications infrastructure is a bespoke router able to replicate packets, where necessary, to implement the multicast function associated with sending the same packet to several different destinations.

Manuscript received November 15, 2012; revised February 09, 2013; accepted March 10, 2013. Date of publication May 13, 2013; date of current version July 19, 2013. This work is supported in part by the Engineering and Physical Sciences Research Council of the U.K. through Grant EP/D07908X/1 and Grant EP/G015740/1 and by ARM and Silistix. This paper was approved by Guest Editor Hasnain Lakdawala.

E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, and S. B. Furber are with the Advanced Processor Technologies Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: luis.plana@manchester.ac.uk).

A. D. Brown is with Electronics and Electrical Engineering, The University of Southampton, Southampton SO17 1BJ, U.K.

Digital Object Identifier 10.1109/JSSC.2013.2259038

II. Neural Network Simulators

There is a significant body of existing and proposed work on the hardware implementation of neural networks using a wide variety of underlying technology: neuromorphic hardware [2], [3], mixed-signal VLSI [4], GPGPU systems [5], FPGA-based accelerator boards [6], nanotechnology [7], and combinations thereof. Neuromorphic hardware, though energy-efficient, tends to be model-specific. This, in turn, imposes too many restrictions on neuron and synaptic model exploration compared with the flexibility offered by general-purpose computing systems. Moreover, the scale of simulation that can be accomplished with most of the above hardware is limited.

Large-scale real-time cortical simulations need enormous computational power. For a neural simulation of the scale of the human brain, composed of approximately 10^{11} neurons each with about 10^4 synapses firing at the rate of about 10 Hz [8], the processing power needed is approximately 10^{16} operations per second; only an Exascale supercomputer [9], yet to be built, would have the capability to simulate an entire human brain. Smaller scale cortical simulations have been demonstrated, so far, by the deployment of IBM BlueGene supercomputers by Markram *et al.* [10] in the *Blue Brain Project* at EPFL (which centres around *in silico* experimentation with the mammalian brain), Djurfeldt *et al.* [11] in brain-scale neocortical simulations, and by Modha *et al.* [12] in cat-scale thalamocortical simulations. Another platform under development is the *Cog Ex Machina* [13] that targets real-time simulation of brain models. The proposed hardware consists of clusters of accelerator nodes, currently composed of off-the-shelf GPUs, which communicate through a photonic network. The *Dendra* chips [14], being developed to replace the GPUs, are envisioned as massively parallel processor chips that integrate hundreds of cores with memristive memory banks. The distinguishing feature of this platform is that it is designed specifically for large-scale neural simulation.

Clearly, a range of simulation approaches and accuracy levels is available. In terms of the scale of simulations that can be performed, the supercomputers outweigh other computing platforms, there are limitations however. Supercomputers incur very high purchase and maintenance costs. In terms of communications, the Message Passing Interface (MPI) widely utilized in HPCs has large overheads, due to its large frame sizes, and is not entirely suitable for neural communications. Power consumption is another major issue. Supercomputers are built with the goal of providing a computing platform for general-purpose scientific applications and, hence, are not a natural fit for neural modelling.

III. Requirements and Design Criteria

Neural modelling and simulation, especially on a very large (cortical size) scale, are characterized by their high computation, communication, and memory requirements. It is beyond the capabilities of the most powerful existing HPCs to model, faithfully, even a subset of such structures, let alone an entire human brain. It would be possible only with massively parallel systems much more powerful than the current state-of-the-art [15] and the power consumption of those systems would be of the order of MegaWatts [9]. Achieving the SpiNNaker goal of simulating 1% of the human brain (10^9 neurons and 10^{12} synapses with each neuron spiking at an average rate of 10 Hz or less) in biological real time will require a large number of processing cores, a very efficient communications infrastructure, and a large memory capacity; energy consumption also must be kept under control. The aim in SpiNNaker is therefore to meet these requirements in a balanced manner in terms of performance, silicon area, power consumption, and cost.

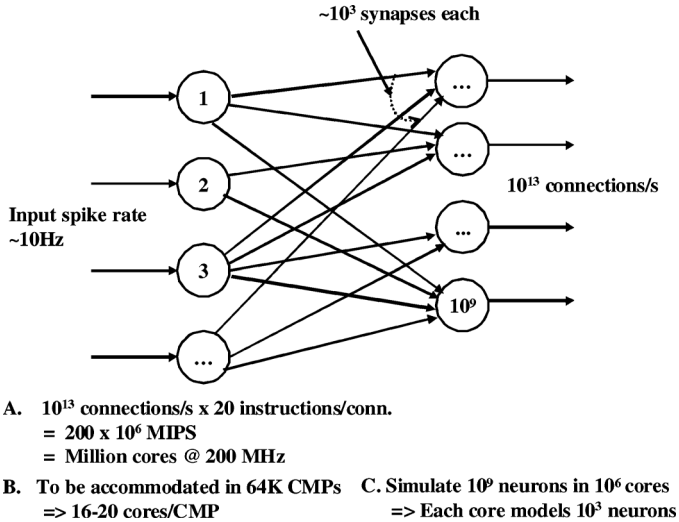


Fig. 2. Processing requirements in SpiNNaker.

A. Processing

Fig. 2 depicts the computational demands on SpiNNaker. To simulate 10^9 neurons and 10^{12} synapses, 10^6 processing cores functioning at a nominal frequency of 200 MHz are needed; each simulating the neuronal dynamics of about 10^3 neurons. These estimates are based on the use of a medium-complexity neuronal model such as that of Izhikevich [16]. If a computationally complex and biologically accurate model, such as the Hodgkin–Huxley model [17], is used, then the size of the network that can be simulated is reduced. This tradeoff highlights the advantage of software modelling: the user can choose the models.

B. Communications

A major challenge in achieving high-performance neural simulation is to emulate the very high interconnectivity of biological neural networks in hardware. Moore *et al.* [18] demonstrate that a large-scale neural simulation is communication- rather than compute-bound, and, in the design of the SpiNNaker CMP, particular emphasis is placed on the communication mechanisms employed [19].

The massive interconnectivity of spiking neural networks (SNNs) can be modelled efficiently as packet-based spike communications using address event representation (AER) proposed by Mahowald [20]. Spikes are stereotypical all-or-none events that carry information only in the timing of their occurrence. Therefore, they are efficiently formatted as small packets containing the source address of the spike-emitting neuron which are sent to a centralized location in each chip where it can be *multicast* (replicated, if needed, and sent to multiple

locations). This scheme decouples the SNN topology from the physical wiring, thereby facilitating programmable connectivity and easily scaling to larger systems [1]. The packets travel across the system at electronic speeds (in nanoseconds), which are much faster than biological spikes (of the order of milliseconds). This allows the emulation of neural networks composed of whole populations of neurons to match biological real time. A spike is processed as soon as it arrives at the destination core, which requests the associated synaptic data from memory and schedules the processing of any associated synaptic events at a time in the future when the biological delay has elapsed [21]. The communications NoC is designed to cope with very high peak packet traffic but is expected to be lightly loaded on average.

C. Memory

Neural simulations require memory to store neuron models, state information, and the results of neural computation. As the neurons are updated periodically, neuron state is kept in a fast-access memory at the processor cores. Synaptic data, on the other hand, is associated with spike sources and is required only when a spike arrives. Given that synaptic events are not processed immediately, the synaptic information can be retrieved from a larger, slower memory using DMA [21]. This larger memory is common to all cores in a chip, however, neural simulations do not need memory coherence, thus avoiding the need for shared memory and the corresponding memory-access mechanisms. In addition to neural processing memory, other forms of memory are needed for inter-processor communication, system-level debugging and management purposes.

D. Energy Efficiency

To minimize energy costs, the CMP is built out of low-power ARM processors and mobile DDR SDRAM. On-chip and off-chip communications infrastructure employs asynchronous interconnect which has lower power consumption than equivalent synchronous buses. Also, event-driven processing is adopted whereby processing cores are put into a low-power *sleep* state while idle and are woken only when responding to events of interest. An efficient vectored-interrupt system with a large set of interrupt signals is provided to cater for this model. This corresponds with biological neural processing where events (spikes) are generated by the neurons and arrive at their destination where they are processed on arrival. Therefore, in this model, processing is done on-demand, rather than wasting energy doing continuous computation.

E. Fault Tolerance and Monitoring

To incorporate fault tolerance into the CMP design, redundancy is implemented in the processor cores, off-chip links, and PLLs. In addition, one processor core is designated as *monitor processor* for chip-level management, monitoring, and debugging tasks. The selection of the monitor processor is done at power-up, and any core that is healthy has an equal chance of being the monitor processor. To ensure smooth packet traffic and avoid congestion, mechanisms such as packet dropping and timeout are included. Failures in the inter-chip communication links are handled by an emergency routing procedure. A programmable CRC ensures data integrity during on-chip data transfer through the asynchronous interconnect.

F. Scalability

The CMP is sufficiently scalable to be a building block for a large-scale system, striving for the power efficiency of analog neural hardware but with the programmability and reconfigurability of general-purpose computers.

IV. SpiNNaker Architecture Overview

The SpiNNaker CMP, shown in Fig. 3, is a GALS system-on-chip (SoC) with 18 ARM968 processor nodes residing in synchronous islands surrounded by an asynchronous packet-switched communications infrastructure [22]. The GALS architecture not only simplifies timing closure in the SoC design but also facilitates isolation of faulty processor nodes [23]. A self-timed, delay-insensitive on-chip interconnect based on CHAIN technology [24] is the backbone for on- and off-chip communications. Two phase-locked loops (PLLs) with associated circuitry generate clock signals for the synchronous modules in the CMP. Clock signals are deliberately skewed to minimize electromagnetic interference (EMI) as well as peak demand on the power supplies. As shown in the figure, each CMP can communicate with the external world through a 100-Mbit Ethernet interface, but not all need be connected.

Data communications are handled by two separate hardware channels—the *Comms NoC* and the *System NoC*. The Comms NoC, shown in the top half of Fig. 3, implements system-wide inter-processor communications from any processor to any other processor in the system. The communication is handled through a bespoke packet router with six full-duplex links connecting to neighboring chips (usually in the North, Northeast, East, South, Southwest, and West directions) to form a 2-D triangular toroidal mesh. The System NoC, shown in the bottom half of the figure, enables the chip-wide sharing of system resources, viz. 32-KB system RAM, 32-KB boot ROM, system controller, watchdog timer, and Ethernet media-independent interface (MII) controller. It also provides access, through a memory controller, to a 128-MB off-die SDRAM, private to each CMP and accessible by all of its processors, housed in the same package. The shared system ROM and RAM are used for the boot process, for loading application code, and for inter-processor communication mailboxes.

The fabric in both NoCs is asynchronous: the System NoC implements CGP (Silistix's asynchronous custom protocol) with ARM's AXI [25] adapters whereas the Comms NoC presents an asynchronous interface and uses a 2-of-7 NRZ protocol for off-chip links and a 3-of-6 RTZ for on-chip links [26]. The asynchronous NoCs with their packet-switched fabric deliver scalable, high-bandwidth, power-efficient, multicast, and low-latency communications.

A. SpiNNaker Processor Node

The internals of the SpiNNaker Processor Node are shown in Fig. 4. The core selected for SpiNNaker is the ARM968E-S 32-bit processor which is an energy-efficient, small-footprint processor designed for low-power, data-intensive applications with a Dhrystone performance of 1.1 DMIPS/MHz. The core is fully synthesizable, so it can be ported quickly to different

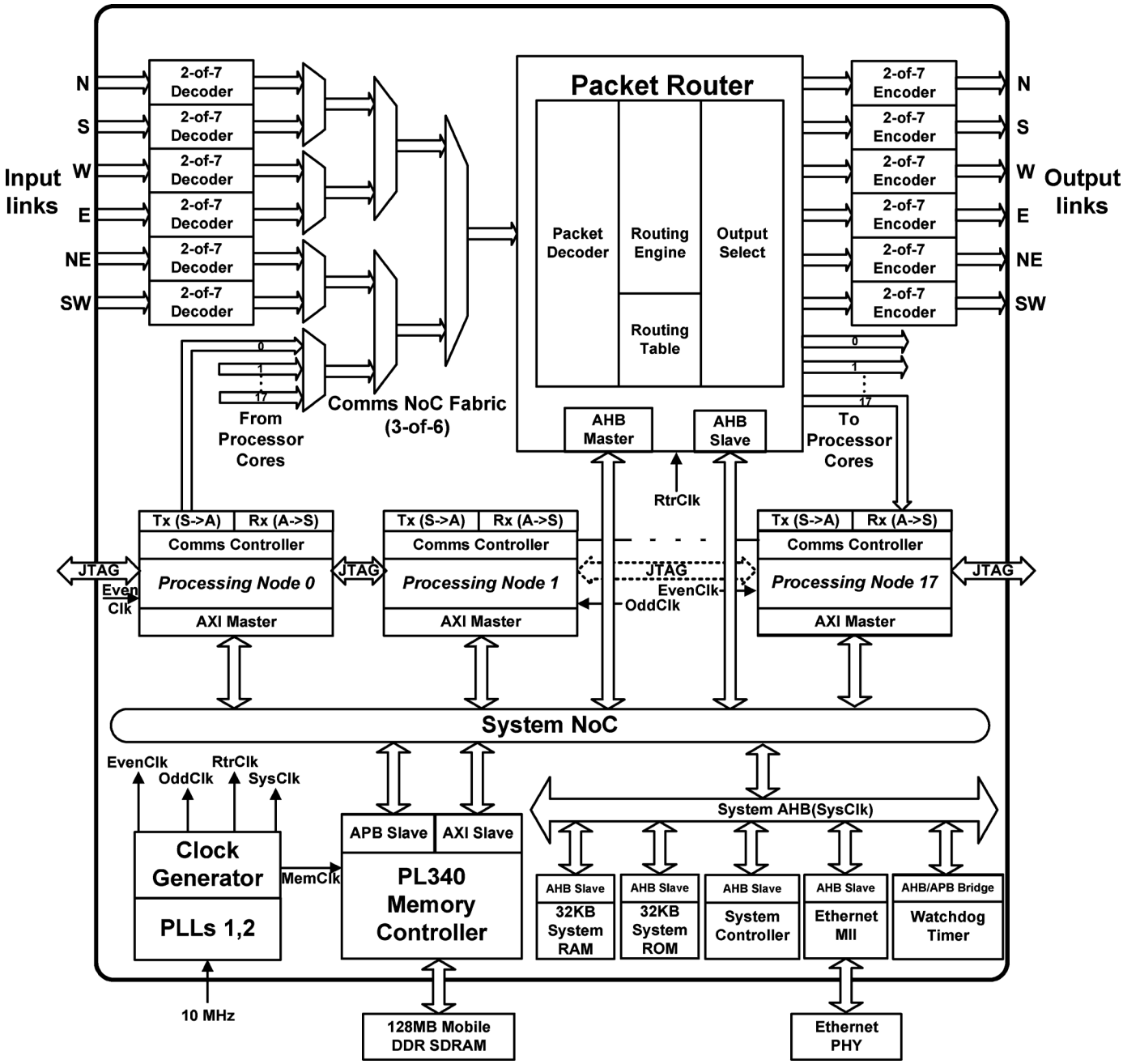


Fig. 3. Organization of the SpiNNaker CMP.

process technologies. The processor node performs both computation and communication functions. Each processor node has an ARM968 core, directly connected 32-KB instruction tightly coupled memory (ITCM), and a dual-banked 64-KB data TCM (DTCM), and peripherals such as timer/counter, vectored interrupt controller (VIC), communications controller (CC), and direct memory access controller (DMAC). The timer/counter generates simulation time step intervals for the neuronal models. The CC handles the packet-based traffic at the processing end. The custom-designed DMAC shares access to the TCMs with the processor core through a dedicated slave interface. Its main function is to offload communication tasks from the processor and provide transparent access to system resources. The VIC handles up to 32 interrupt requests with programmable priority from the node peripherals as well as system resources and generates interrupts to the processing core. An IEEE 1149.1-compliant JTAG port is also available for debugging purposes.

B. Inter-Processor Communications

The communications infrastructure has the primary role of carrying neural event packets between cores, which can be located in the same or different chips, but is also used to transport system configuration and monitoring information. Fig. 5 shows the GALS communications system with the synchronous router and on-chip processor nodes interconnected through a self-timed fabric. Each core node, which includes a local communications controller, has its own clock signal, as does the router. The network interface modules (NIMs) are responsible

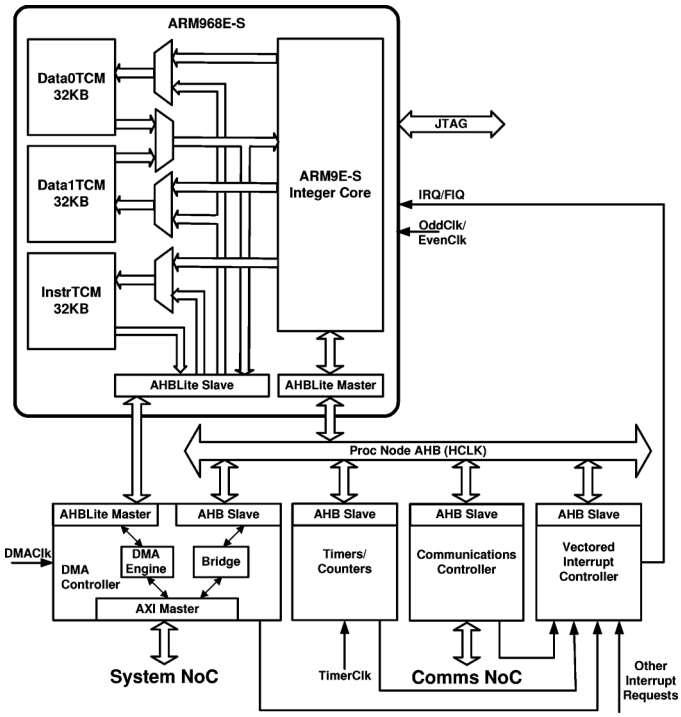


Fig. 4. Details of the SpiNNaker processor node.

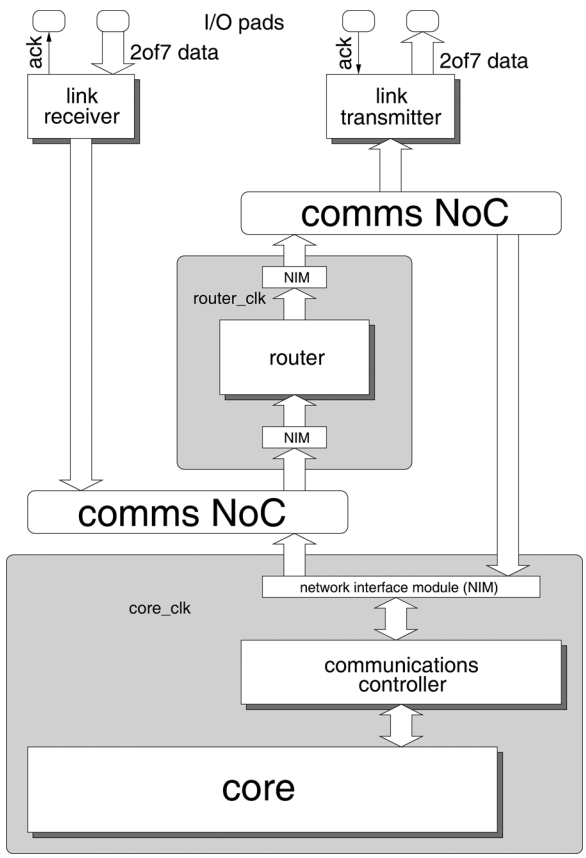


Fig. 5. SpiNNaker inter-processor communications infrastructure.

for serializing and de-serializing packets and synchronize the signals that cross timing domains.

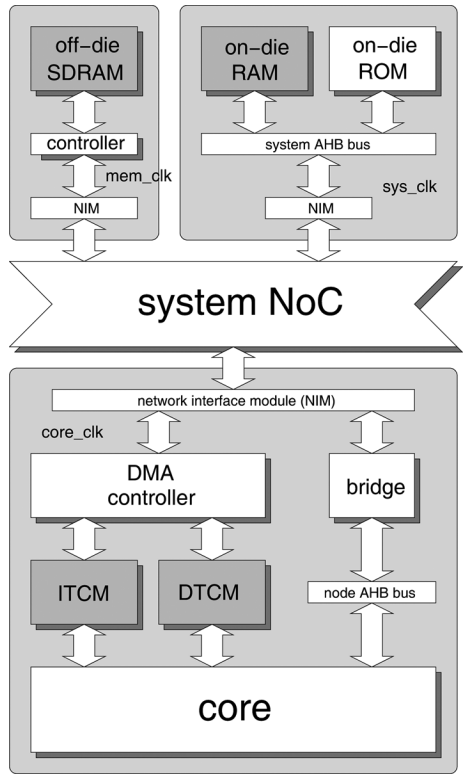


Fig. 6. SpiNNaker memory organization.

The router is responsible for routing all packets to one or more of its outputs. Packets are 40 or 72 bits long and consist of a control byte and one or two data words. The second data word is optional and its presence is indicated by a bit in the control byte. The router identifies and handles several error conditions such as packet parity and framing errors, timeout, and output link failure.

The router [27] combines different routing mechanisms. Multicast packets (used to represent neural events) are routed through a three-valued associative lookup table (LUT). The packet keys are structured in a hierarchical way to allow a single entry to represent one or more populations of neurons, avoiding the need for large tables. Point-to-point packets (used for control and monitoring operations) use a simple LUT and fixed-route packets (intended for communications with a host computer) simply follow a predefined route stored in a router register. These mechanisms require table and register setup and therefore are available only after SpiNNaker has booted. Nearest-neighbor packets (used to communicate with neighboring chips during boot-up) use a simple algorithmic process that does not require any setup and therefore are the only mechanism available during the boot process. The router also implements default routing (used when a multicast packet does not match any entry in the multicast table) and emergency routing (used when an output link is blocked due to congestion or hardware failure).

C. Memory Organization

The CMP memory organization is shown in Fig. 6. Program code and frequently used data are expected to reside in tightly coupled memory which runs at core speed. Accesses to off-die

TABLE I
INTERRUPT SOURCES PER CORE

Node Level
Communication controller packet reception (5x)
Communication controller flow control (3x)
Communication controller errors (3x)
DMA transfer completion
DMA transfer error (2x)
Local timer/counter (2x)
Software interrupt
Chip Level
Router diagnostics event
Router packet error
Router un-routable packet
Watchdog timer
Slow system clock
Ethernet controller (3x)
On-chip inter-processor interrupt
Off-chip interrupts (4x)

SDRAM go through the System NoC and are therefore long latency.

Each node includes a DMA controller to alleviate this problem. The TCM–DMAC–SDRAM system can be used as a software-managed cache. When needed, the DMAC transfers large blocks of data between SDRAM and TCM under control of the core. In addition, the DMAC can transfer data to/from other targets on the System NoC such as the System RAM and Boot ROM. DMA transfers are initiated by writing to control registers in the DMAC. They are executed in the background, and the relevant interrupt caught when the work is complete. The DMAC possesses dual buffers supporting pipelined transfers, provides support for CRC error detection in transferred blocks and completion interrupts.

The Bridge is a second path for memory accesses. Bridge transfers occur when the core initiates load or store requests directly to the needed memory address and the core retains full control of the transfer. If requested, the Bridge may buffer the data from write requests for increased efficiency. If an error occurs on such a buffered write the Bridge signals an error interrupt.

It is interesting to note that the NIM was designed to allow a single outstanding memory operation. This may have a negative impact on the network bandwidth utilization of each individual core but is used as a simple mechanism to provide fair access to the System NoC by all cores.

D. Event-Driven Operation

SpiNNaker operates in an entirely event-driven fashion to optimize performance and energy consumption. There is no conventional operating system running on the cores, simply a low-level interrupt service provider kernel. A core is normally in low-power sleep mode. When an interrupt arrives, the core wakes up to process any required data, possibly requesting DMA transfers or emitting packets. On completion of the interrupt service the core returns to sleep.

Table I lists the different sources that can interrupt the cores. A large number of distinct interrupt sources supports efficient interrupt-driven operation as it relieves the core from wasting clock cycles having to identify the source after being interrupted.

V. SPINNAKER CMP IMPLEMENTATION

The implementation of the SpiNNaker CMP was a big challenge given the size and complexity of the system. As explained earlier, SpiNNaker integrates several external IP devices, such as the ARM cores and SDRAM controller, with components developed in-house by the SpiNNaker team. Details of the design considerations and choices were presented in [28].

A. Implementation Methodology

The Synopsys Galaxy Design Platform was used for the design and implementation tasks. The implementation employs architecture and logic-level clock gating. The design methodology has been fine-tuned with special emphasis on the power efficiency of the clock networks. Power-aware synthesis was used throughout the flow. The ARM processor cores were brought in as IP and their physical implementation fine-tuned through numerous iterations to achieve the most compact, power-efficient and best achievable performance design using the ARM-Synopsys Galaxy reference design methodology. Other blocks such as the memory controller, router, and system AHB (containing the system resources) were also implemented adopting the same design methodology. The NIMs (that interface the synchronous islands with the asynchronous fabric) were laid out as hard macros to safeguard certain timing assumptions. They were then integrated into the processor nodes, router, system AHB and SDRAM controller modules during the P&R flow to ensure minimum design area. At the top level, these synchronous modules were combined with the asynchronous fabric (laid out as a combination of hard macros and soft sea-of-gates) in a bespoke GALS design methodology [23] to create the CMP layout.

Physical layout of a complex many-core GALS chip such as the SpiNNaker CMP with conventional CAD tools, tailored for synchronous systems, is a difficult task. For asynchronous circuitry, a hierarchical methodology was employed [23]. Even though the asynchronous logic is delay-insensitive, certain timing assumptions had to be safeguarded for correct operation. Hence, encapsulating small sections of the logic in customized macros and using these as component building blocks for the larger sections was the adopted approach for the successful implementation of the SpiNNaker CMP. Even then, this approach needed minor modifications during the scaling up of the SpiNNaker CMP from the test chip with two cores to the current chip with 18 cores.

B. Verification and Testing

SystemC was used to validate the architecture and design of the SpiNNaker CMP. The synchronous models were cycle accurate, but asynchronous networks could be modelled only with early delay estimates. Building the SystemC model took more time than initially anticipated, given that different IP providers use different SystemC conventions and interfaces and adapting them took significant effort.

The synchronous modules followed a standard verification path. External IP was delivered in Verilog, which was also used to develop most of the in-house designs. Equivalence checking was used to verify RTL synthesis and optimization. Gate-level simulation with extracted parasitics and annotated delays were used for simulations. Initially, top-level modules, such as the cores and the router, were simulated individually. After all

modules met their targets and were deemed correct, a complete system model was used to verify the top-level interconnect. The GALS nature of the design made this easier given that no clock-related timing had to be met and verified. Also, it allowed the use of a combination of models to speed the simulation, e.g., a single core was simulated using a detailed, back-annotated gate-level model and the other 17 used a much faster behavioral model. When using the simplified models, simulations with back-annotated delays from extracted parasitics took several hours to complete. An early version of the chip, with only two cores, was produced and used for detailed verification [23].

The asynchronous modules had to follow a different verification path. External IP was delivered in technology-mapped, gate-level Verilog, avoiding the need for formal verification at this stage. Given the *reluctance* of most CAD tools to work without clock-related timing constraints, detailed transistor-level simulations were used to verify the operation and estimate the timing characteristics of these modules, which were converted into soft macros.

Finally, the mesochronous interface to the off-die SDRAM was carefully verified. The in-house designed asynchronous delay-locked loop (DLL) [29] that controls the interface was simulated and characterized at transistor-level while the synchronous memory controller was simulated at gate level with detailed parasitics, which allowed the correct alignment of interface signals in the presence of possibly unbalanced PCB tracks.

VI. Results

A. Fabricated SpiNNaker CMP

The SpiNNaker CMP was designed by a small team of academic researchers and postgraduate students with the associated restrictions and constraints regarding fabrication cost and access to process technologies, standard cell libraries and IP. Overall, a design effort of approximately 40 years has gone into the design, implementation, and verification of the SpiNNaker CMP. A test chip with two cores was taped out in August 2009 followed by the 18-core chip in December 2010.

Shown in Fig. 7 is the die micrograph of the SpiNNaker CMP with its major components highlighted. The CMP was fabricated in UMC 130-nm L130E process technology and has over 100 million transistors and a power consumption of 1 W when all of the processor cores are operating at 180 MHz. The SpiNNaker CMP achieves a peak performance of 3.96 GIPS; key figures are listed in Table II.

B. Building SpiNNaker Systems

So far, four variants of small-scale SpiNNaker systems have been built and used for neural modelling and simulation. Fig. 8 shows the fourth generation system, a 48-node board that forms the basic block for the construction of larger SpiNNaker systems. Forty-eight SpiNNaker chips connected in a hexagonal array are mounted on an extended depth, double height Eurocard (220 mm $\times$ 233.4 mm) with a Board Management Processor and three FPGAs to implement SpiNNlinks (high-speed inter-board channels). A SpiNNlink multiplexes/demultiplexes eight asynchronous channels into a serial 3-Gbps I/O channel using

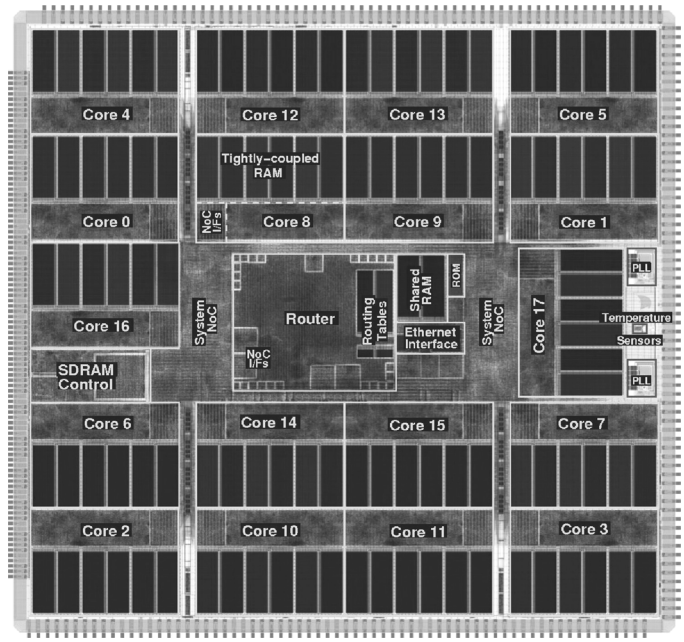


Fig. 7. Die plot of the SpiNNaker CMP.

TABLE II
SpiNNaker CMP Summary

	Process
Process technology	UMC 130 nm 1P8M CMOS
Die area	101.64 mm^2
Power supply	1.2 V (Core), 1.8 V (I/O)
	Cores
Processor cores	18 ARM968s (1 monitor 16 application & 1 spare)
Processor frequency	180 MHz
Processor node area	3.75 mm^2
	Memory
Local memory per core	32 KB (Instruction) 64 KB (Data)
On-die shared RAM	32 KB
Off-die shared RAM	128 MB DDR2 SDRAM
	Communications
On-chip interconnect	Asynchronous NoCs
Off-chip link b/w	250 Mb/s
On-chip comms Link b/w	5.0 Gb/s
Off-die SDRAM b/w (DMA)	7.2 Gb/s
On-die shared RAM b/w (DMA)	3.2 Gb/s
On-die shared RAM b/w (bridge)	200 Mb/s
Router input b/w	5.3 Gb/s
57K-node sys. bisection b/w	4.6 Gpackets/s
	Power Consumption
Peak (chip)	1 W
Idle (chip)	360 mW
Idle (core)	20 mW
Off-chip link (full speed)	6.3 mW (25pJ/bit)
SDRAM	170 mW
	Implementation
Transistor count	$\sim$100 million
Design effort	$\sim$40 person-years

SATA cables for board-to-board interconnect. These boards operate without any cooling system.

The next-generation SpiNNaker is a 576-node (10,368-core) system, consisting of 12 boards in a 19-in frame. The full-size

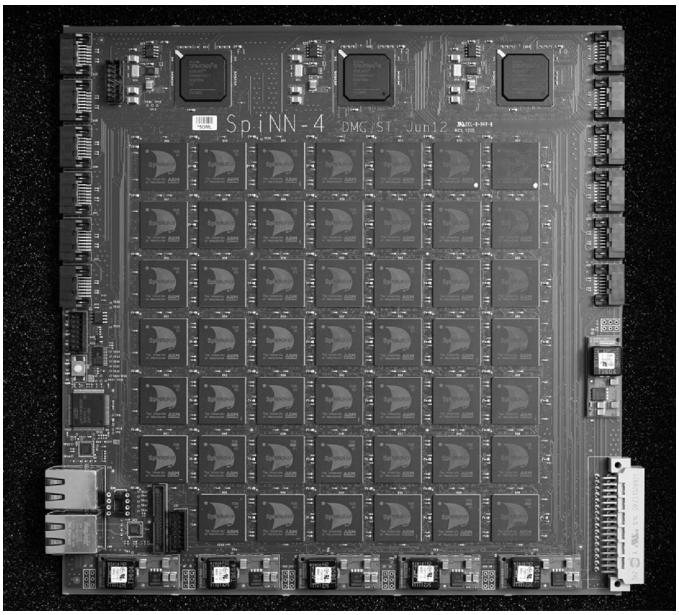


Fig. 8. Fourth-generation SpiNNaker system with 864 processor cores housed in 48 chips having a peak power consumption of 60 W and capable of simulating a million neurons and a billion synapses.

SpiNNaker system, targeted for 2013, will be constructed using 1200 PCBs containing 57 600 chips, i.e., 1 036 800 processors. They will use off-the-shelf, forced-air cooling systems.

C. SpiNNaker in Action

SpiNNaker systems built so far have been used in robotics applications, which require low-power neural computation along with the ability to process sensory inputs and generate motor outputs in real time. Galluppi *et al.* [30] described how a SpiNNaker system interacts with an artificial silicon retina in real time to model attention control in visual processing. Similarly, Davies *et al.* [31] used SpiNNaker to implement an SNN which successfully controlled a line-following robot through an AER interface. These examples demonstrate that SpiNNaker is capable of real-time SNN computation whilst drawing very little power.

VII. SpiNNaker Evaluation

A. SDRAM Bandwidth Utilization

The optimal utilization of the SDRAM bandwidth is crucial to the performance of SNN simulations, Fig. 9 illustrates the results of an experiment to evaluate this key indicator. The figure shows the aggregate SDRAM bandwidth utilization of a number of cores on a single SpiNNaker CMP, increased progressively from 1 to 14. The experiment is based on DMA acceses to memory, with the first seven cores reading from memory and the last seven performing write operations. It is clear from the figure that the read channel from memory saturates at just over 600 MB/s and the write channel adds around 300 MB/s on top of that, for a total aggregate bandwidth of 900 MB/s. As the number of writes increases the bandwidth available for read requests decreases, given that both are transported on the same network channel. This results in an unfortunate reduction in the read data channel bandwidth utilization. The figure also shows that, as expected, the simple mechanism of restricting each core to a single outstanding memory access results in a fair share of the bandwidth for all cores.

B. SNN Simulations

To evaluate SpiNNaker performance in SNN simulations, we built a controlled neuron model in which events, rather than neural quantities, are the experimental variables. This approach lets us control the dynamics of the experiment and evaluate results in a precise way. In particular, the number of packets emitted by every core, and the number of ARM instructions needed to process a neural event, can be varied, and the number of spikes and synaptic events can be measured. This neural model is integrated in the mapping which automatically configures the number of neurons, synapses and the topology of the network.

Table III shows the results from three experiments. Populations are groups of neurons and projections are interconnections between populations. The first experiment consists of many simple neurons, firing at a low firing rate, and few interconnections between them. The other two involve more complex neurons, interconnections and higher population firing rates. A core needs 33 instructions to update the state of each neuron in the simple model and 93 in the complex one. Connections, also known as synaptic events, need six instructions in all cases. All of the experiments run in real time on a prototype board equipped with 4 SpiNNaker chips.

The results verify that small-scale SpiNNaker systems meet their SNN simulation performance targets and highlight the flexibility and programmability of the system by modelling networks with different characteristics, regimes, dynamics, and topologies. We used the energy consumption of a chip after a reset and with no software loaded onto it as the baseline to measure the energy associated with a neuron and with a connection event. We measured the mean power for the three experiments, allowing neurons to be updated, packets to be sent, and synapses to be retrieved from SDRAM. We then disabled packet sending, leaving periodic neural update as the only process running on the system. As a result we were able to differentiate between the power associated with neuron activity and that associated with connections. Results for neuron power consumption reflect the difference in complexity of the model used in experiment 1 with that used in experiments 2 and 3, while the connection energy stays constant as the number of instructions per synapse does not change. For an example of more biologically-plausible experiments on SpiNNaker, refer to Sharp *et al.* [32].

C. SpiNNaker Assessment

As no full-size SpiNNaker system has yet been built it is still early to predict what the architectural shortfalls may be but an *obvious* limit, given that the routing keys for system administration packets are 16 bits long, is that the system can expand only to 64 K nodes and there is no straightforward mechanism to overcome this limit.

With small-scale systems it has become apparent that the network optimization for neural traffic (short, fast packets whose delivery is not guaranteed) makes SpiNNaker less suitable for some applications. The ability to carry larger payloads

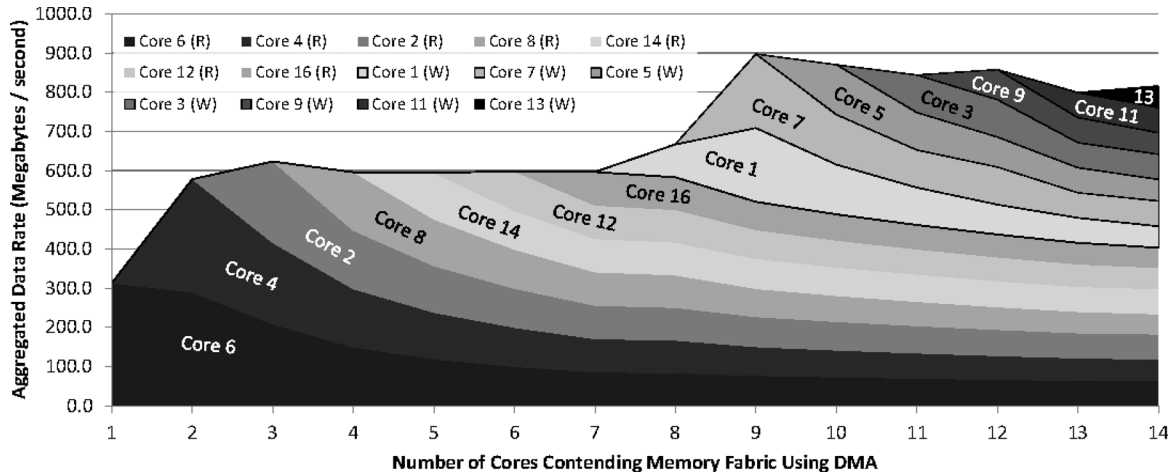


Fig. 9. SDRAM bandwidth utilization.

TABLE III
SPIKING NEURAL NETWORK SIMULATIONS

	Exp. 1	Exp. 2	Exp. 3
Neurons per population	1,000	100	100
Populations	16	16	64
Incoming projections/pop.	1	5	5
Synapses/population	10^6	50,000	50,000
Total number of neurons	16,000	1,600	6,400
Total number of synapses	16×10^6	800,000	3.2×10^6
Spikes/population/s	4,000	10,000	10,000
Total spikes/s	64,000	160,000	640,000
Total connections/s	64×10^6	80×10^6	320×10^6
			Mapping
Cores used	16	16	64
Chips used	1	1	4
	Processing and communications per core		
Neurons	1,000	100	100
Neuron updates/s	10^6	100,000	100,000
Instructions/neuron update	33	93	93
(Neuron up.) instructions/s	33×10^6	9.3×10^6	9.3×10^6
Synapses	10^6	50,000	50,000
Connections/s	4×10^6	5×10^6	5×10^6
Instructions/connection	6	6	6
(Connection) instructions/s	24×10^6	30×10^6	30×10^6
Total instructions/s	57×10^6	39.3×10^6	39.3×10^6
Packets processed/s	4,000	50,000	50,000
			Energy efficiency
Power/neuron (nJ/ms)	12	44	45
Energy/connection (nJ)	4	4	4

per packet would alleviate this. The risk that a control packet may be dropped *irrecoverably* in the network also puts an additional burden on the software, forcing acknowledgements and timeouts to be monitored.

Gratifyingly, there have not been any major bugs discovered to date due to the chip's detailed implementation. However, experience in use has revealed several issues which, in hindsight, could have been better addressed. In particular, opportunities for additional power savings were missed, even though minimizing power consumption was a significant design criterion.

Because the chip was designed primarily for real-time operation it is necessary, always, to have some processing "in hand," i.e., cores have to be ready to process when the need arises. When a core runs out of tasks it puts itself into sleep mode in which its clock is gated off, minimizing its power consumption. Provision for this is part of the ARM968 core. This halts the processor but not its associated peripherals running from the same basic clock. The peripherals cannot be halted directly with the processor because their activity is independent. DMA operations may be running whilst the processor sleeps, incoming communications still need to be received and the local timer must continue. Therefore, no provision was made to halt the peripherals. The missed opportunity is that applications do not necessarily use all the cores and those idle can be turned off, including their associated peripherals.

The clock speeds of the various cores are not independent; this precludes tailoring their individual speeds to their loads. Because the cores are intended to run in real time–alternating bursts of activity with sleeping–this is not perceived as a handicap. In a future design, it may save power to implement dynamic voltage scaling (DVS) on a core-by-core basis and the GALS interconnect means the logic design would accommodate this.

Some issues of multiprocessor operation have become apparent as the chip's system software has been developed. Foremost amongst these is the limited hardware support for inter-core *control* communications. It is possible for any processor to interrupt another using a shared interrupt source. The reason for the interrupt must be conveyed in a data structure in shared RAM and a hardware bit-flag register supports atomic accesses to this structure. It is now clear that additional such registers, combined with multiple distinct interrupts, would enable simpler and faster inter-core signalling.

There are several other places where hardware registers could relieve the need to build software structures. A good example is the DMA controllers which are internally pipelined. A simple identifier, carried with each DMA transfer would save the need to build a queue when DMA transfers are started and finished by different threads.

In a massive system, fault tolerance was envisaged to be an issue from the start. Although the fault-tolerant features included in the chip have not yet been fully explored new ideas for optimizations are still being generated. For example, the GALS communication necessarily exerts *back pressure* if a receiver is slow. If a receiver is inoperative then an increasing

amount of the network could become fatally congested. This is alleviated by a programmable timeout in the routers. However, a mode in which receivers simply discard packets would prove a simple addition, particularly as a start-up default in case a faulty processor subsystem fails to boot correctly. There are, no doubt, numerous other such features which could be included.

VIII. Conclusion

SpiNNaker is a high-performance, application-specific architecture highly optimized for running neuroscience applications. It can also be used for other distributed computing, such as ray-tracing and protein folding, and it provides a cost-effective means of achieving over 10^{14} operations per second, provided that floating-point arithmetic is not required.

Experimental results show that, for massively parallel neural network simulations, a customized multicore architecture can be energy efficient while keeping the flexibility of software-implemented neuronal and synaptic models, absent in current neuromorphic hardware.

As a message-passing system, the greatest performance bottleneck is the communications between processors and, therefore, SpiNNaker was optimized for the short, multicast messages (spikes) associated with neural network simulation. This optimization has resulted in some additional overhead for other applications, including loading and control of neural networks which is done by sharing the run-time network. In hindsight, provision for larger payloads with guaranteed delivery would relieve the software burden in sharing these disparate tasks. However, the decision to share a single network still appears sensible, and this relieves some of the system-level problems.

Acknowledgment

The authors appreciate the support of their sponsors and industrial partners. Photographs in Fig. 1 courtesy of Unisem (Europe) Ltd. and Fig. 8 courtesy of Norcott Technologies Ltd.

References

[1] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, "Overview of the SpiNNaker system architecture," *IEEE Trans. Comput.*, vol. PP, no. 99, 2012.

[2] R. Silver *et al.*, "Neurotech for neuroscience: Unifying concepts, organizing principles, and emerging tools," *J. Neurosci.*, vol. 27, no. 44, pp. 11 807–11 819, 2007.

[3] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, pp. 73:1–73:23, May 2011.

[4] J. Schemmel *et al.*, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1947–1950.

[5] M. Richert *et al.*, "An efficient simulation environment for modeling large-scale cortical processing," *Frontiers Neuroinf.*, vol. 5, pp. 19:1–19:15, Sep. 2011.

[6] B. Glackin *et al.*, "A hardware accelerated simulation environment for spiking neural networks," in *Proc. 5th Int. Workshop Reconfigurable Computing: Architectures, Tools Applicat.*, 2009, pp. 336–341.

[7] C. Gao and D. Hammerstrom, "Cortical models onto CMOL and CMOS – Architectures and performance/price," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2502–2515, Nov. 2007.

[8] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA, USA: MIT, 2005.

[9] J. Torrellas, "Architectures for extreme-scale computing," *Comput.*, vol. 42, no. 11, pp. 28–35, Nov. 2009.

[10] H. Markram, "The blue brain project," *Nat. Rev. Neurosci.*, vol. 7, no. 2, pp. 153–160, Feb. 2006.

[11] M. Djurfeldt *et al.*, "Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer," *IBM J. Res. Develop.*, vol. 52, no. 1/2, pp. 31–41, Jan. 2008.

[12] D. Modha *et al.*, "Cognitive computing," *Commun. ACM*, vol. 54, pp. 62–71, Aug. 2011.

[13] M. Versace and B. Chandler, "The brain of a new machine," *IEEE Spectr.*, vol. 47, no. 12, pp. 30–37, Dec. 2010.

[14] A. Gorchetchnikov, M. Versace, H. Ames, B. Chandler, J. Leveille, G. Livitz, E. Mingolla, G. Snider, R. Amerson, D. Carter, H. Abdalla, and M. Qureshi, "Review and unification of learning framework in Cog Ex Machina platform for memristive neuromorphic hardware," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 2601–2608.

[15] S. Furber, "To build a brain," *IEEE Spectr.*, vol. 49, no. 8, pp. 39–43, Aug. 2012.

[16] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[17] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[18] S. W. Moore *et al.*, "Bluehive – A field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Proc. IEEE 20th Int. Symp. Field-Programmable Custom Computing Mach.*, 2012, pp. 133–140.

[19] C. Patterson, J. Garside, E. Painkras, S. Temple, L. Plana, J. Navaridas, T. Sharp, and S. Furber, "Scalable communications for a million-core neural processing architecture," *J. Parallel Distrib. Computing*, vol. 72, no. 11, pp. 1507–1520, 2012.

[20] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. Boston, MA, USA: Kluwer, 1994.

[21] X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber, "Algorithm and software for simulation of spiking neural networks on the multi-chip Spinnaker system," in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–8.

[22] L. Plana, S. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS infrastructure for a massively parallel multiprocessor," *IEEE Des. Test Comput.*, vol. 24, no. 5, pp. 454–463, 2007.

[23] L. 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 multicore system-on-chip," *ACM J. Emerg. Tech. Comput.*, vol. 7, no. 4, pp. 17:1–17:18, 2011.

[24] J. Bainbridge and S. Furber, "CHAIN: A delay-insensitive chip area interconnect," *IEEE Micro.*, vol. 22, no. 5, pp. 16–23, 2002.

[25] AXI Protocol Specification. Advanced Microcontroller Bus Architecture (AMBA). [Online]. Available: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php, Rev.1.0

[26] Y. Shi, S. Furber, J. Garside, and L. Plana, "Fault tolerant delay insensitive inter-chip communication," in *Proc. 15th IEEE Symp. Asynchronous Circuits Syst.*, 2009, pp. 77–84.

[27] J. Wu, S. Furber, and J. Garside, "A programmable adaptive router for a GALS parallel system," in *Proc. 15th IEEE Symp. Asynchronous Circuits Syst.*, 2009, pp. 23–31.

[28] E. Painkras, L. A. Plana, J. Garside, S. Temple, S. Davidson, J. Pepper, D. Clark, C. Patterson, and S. Furber, "SpiNNaker: A multi-core system-on-chip for massively-parallel neural net simulation," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2012, pp. 1–4.

[29] J. Garside, S. Furber, S. Temple, D. Clark, and L. Plana, "An asynchronous fully digital delay locked loop for DDR SDRAM data recovery," in *Proc. 18th IEEE Int. Symp. Asynchronous Circuits Syst.*, 2012, pp. 49–56.

[30] F. Galluppi, K. Brohan, S. Davidson, T. Serrano-Gotarredona, J. Carrasco, B. Linares-Barranco, and S. Furber, "A real-time, event driven neuromorphic system for goal-directed attentional selection," in *Proc. 19th Int. Conf. Neural Inf. Process.*, Nov. 2012, pp. 226–233.

[31] S. Davies, C. Patterson, F. Galluppi, A. Rast, D. Lester, and S. Furber, "Interfacing real-time spiking I/O with the SpiNNaker neuromimetic architecture," in *Proc. 17th Int. Conf. Neural Inf. Process.*, 2010, vol. 11, pp. 7–11, 1.

[32] T. Sharp, F. Galluppi, A. Rast, and S. Furber, "Power-efficient simulation of detailed cortical microcircuits on SpiNNaker," *J. Neurosci. Meth.*, vol. 210, no. 1, pp. 110–118, 2012.

Eustace Painkras received the B.Tech degree from Kerala University and the M.S. degree from the State University of New York at Stony Brook, Stony Brook, NY, USA.

She is currently a Lecturer with the University of Bedfordshire, Bedfordshire, U.K. Previously, she was a Research Fellow with the School of Computer Science, University of Manchester, Manchester, U.K. Her research interests include VLSI design, wireless communications systems, and reliable and power-aware computer architectures.

Luis A. Plana (M'97–SM'07) received the Ingeniero Electrónico degree from Universidad Simón Bolívar, Venezuela, the M.Sc. degree in electrical engineering from Stanford University, Stanford, CA, USA, and the Ph.D. degree in computer science from Columbia University, New York, NY, USA.

He was with Universidad Politécnica, Venezuela, for over 20 years, where he was a Professor and Head of the Department of Electronic Engineering. Currently, he is a Research Fellow with the School of Computer Science, University of Manchester, Manchester, U.K.

Jim Garside received the B.Sc. degree in physics and Ph.D. degree in computer science from The University of Manchester, Manchester, U.K., in 1983 and 1987, respectively. 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. Manchester, U.K., 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.

Steve Temple received the B.A. degree in computer science and Ph.D. degree for research into local area networks from the University of Cambridge, Cambridge, U.K., in 1980 and 1984, respectively.

He was subsequently a Research Fellow with 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 with the School of Computer Science at the University of Manchester, Manchester, U.K.

Francesco Galluppi (S'12) received the B.Sc. degree in electronic engineering from the University of Rome "Roma Tre," Rome, Italy, in 2005, and the M.Sc. degree in cognitive psychology from the University of Rome "La Sapienza," Rome, Italy, in 2010. He is currently working toward the Ph.D. degree at the University of Manchester, Manchester, U.K.

He joined the SpiNNaker group at the University of Manchester, Manchester, U.K., in 2010. His interests lie in exploring computational intelligence by modelling spiking neural networks on dedicated hardware.

Cameron Patterson (S'10–M'13) received the B.Eng. honors degree from the University of Lancaster, Lancaster, U.K., in 1996, and the Ph.D. degree from the University of Manchester, Manchester, U.K., in 2012.

He is a Postdoctoral Researcher within the Advanced Processor Technologies Group, School of Computer Science, University of Manchester, Manchester, U.K. His interests include massively parallel computers, their interconnection and management, and real-time visualization of both their hardware and software. Prior to beginning his doctoral studies, he worked in industry on large-scale telecommunications network design and maintenance, and in process and product development.

Dr. Patterson is a Chartered Engineer and maintains an active Cisco CCIE certification in routing and switching.

David R. Lester received the M.A. degree in mathematics from the University of Oxford, Oxford, U.K., in 1983, the M.Sc. degree in computer science from University College, London, U.K., in 1984, and the D.Phil. degree in computer science from the University of Oxford, Oxford, U.K., in 1988.

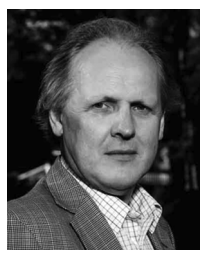
From 1988 to 1990, he was with GEC-Marconi working on functional language compilers for parallel machines, validating communication protocols on transputer hardware. He joined the University of Manchester, Manchester, U.K., as a Lecturer in 1990.

Dr. Lester is a member of the IET.

Andrew D. Brown (M'90–SM'96) received the degree in physical electronics and the Ph.D. degree in microelectronics from Southampton University, Southampton, U.K., in 1976 and 1981, respectively.

He has been a member of academic staff with Southampton University, Southampton, U.K., since 1980. He spent time with IBM Hursley Park U.K. as an IBM Visiting Scientist in 1983, with Siemens NeuPerlach (Munich) as a Visiting Professor in 1989, and with 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, U.K., as a Visiting Professor.

Prof. Brown is a Fellow of the IET and the BCS, a Chartered Engineer, and a registered European Engineer.



Steve B. Furber (M'98–SM'02–F'05) received the B.A. degree in mathematics and Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K., in 1974 and 1980, respectively.

He was with the R&D Department, Acorn Computer Ltd. from 1981 to 1990 and was a Principal Designer with BBC Micro and the ARM 32-bit RISC microprocessor. He moved to his current position as ICL Professor of Computer Engineering with the University of Manchester, Manchester, U.K., in 1990.

Prof. Furber is a Fellow of the Royal Society, the Royal Academy of Engineering, the BCS, and the IET.