Virtual Synaptic Interconnect Using an Asynchronous Network-on-Chip

Alexander D. Rast, Shufan Yang, Mukaram Khan, Steve B. Furber

***Abstract*— Given the limited current understanding of the neural model of computation, hardware neural network architectures that impose a specific relationship between physical connectivity and model topology are likely to be overly restrictive. Here we introduce, in the SpiNNaker chip, an alternative approach: a mappable virtual topology using an asynchronous network-on-chip (NoC) that decouples the "logical" connectivity map from the physical wiring. Borrowing the established digital RAM model for synapses, we develop a concurrent memory access channel optimised for neural processing that allows each processing node to perform its own synaptic updates as if the synapses were local to the node. The highly concurrent nature of interconnect access, however, requires careful design of intermediate buffering and arbitration. We show here how a locally buffered, one-transaction-per-node model with multiple synapse updates per transaction enables the local node to offload continuous burst traffic from the NoC, allowing for a hardware-efficient design that supports biologically realistic speeds. The design not only presents a flexible model for neural connectivity but also suggests an ideal form for general-purpose high-performance on-chip interconnect.**

I. How to Map Neural Networks to Hardware?

WHILE the dynamics of spiking neural networks at the neuron level are fairly well-understood, questions of network organisation at the functional unit level, data representation, and inter-system communication remain largely unanswered [1]. Important biological discoveries [2] have fuelled ongoing debates on the nature of neural structure and dynamics [3], which seem unlikely to lead to any resolution without powerful and sophisticated modelling probably incorporating dedicated hardware [4]. How to implement this hardware, however, is an almost equally contentious question.

One popular method is the "neuromorphic" chip, exemplified by [5], using analogue circuits to model neural properties directly. While these devices remain an important research area since analogue circuits are in principle more biologically accurate as well as more space-efficient, in practice analogue circuitry suffers from a significant device and process technology lag behind digital. The second method is the "neural accelerator", relying on carefully optimised digital technologies with shared synaptic memory. MASPINN [6] and SP^{2}INN [7] are examples of this approach. In principle digital devices are programmable and thus more flexible than analogue, however, to date, the need to use bit-mapped inputs over standard bus interfaces for synaptic memory access limits model choice. Historically, both models, relying on a direct mapping from the hardware to the physical structure of the neural network, encounter a severe limitation: decisive commitment to the neural model at the point of instantiation.

The authors are with the School of Computer Science, University of Manchester, Manchester, M13 9PL, U. K. (e-mail rasta@cs.man.ac.uk)

More recently, FPGA's [8] offer the possibility for configurable topologies and processing functions, but use a circuit-switched connectivity model that severely constrains utilisation of routing resources. If the mapping of the neural model to the FPGA remains one-to-one, then, just as in the custom chip case there will be some networks the device can model and others that it cannot. Lurking behind these limitations is a challenging barrier: direct-mapped neural network implementations rapidly become routing-area dominated.

Traditional bus-based interconnect designs scale poorly, because centralised arbitration and a single shared data channel constrains activity to an exclusive path. The packet-switched Network-on-Chip (NoC) model overcomes this scaling limitation while solving both the problem of routing area and hardwired neural structure elegantly. We introduce a routable asynchronous NoC onto which virtually any network can be mapped with run-time remapping. The neural network topology itself is then the *virtual* topology, impressed upon the *physical* topology of the NoC wiring itself. The NoC offers an additional significant advantage: it retains the concurrent dynamics that is the essential feature of neural computation. Such a network suggests a powerful model for a universal neural network chip: an array of programmable processors embedded in a sea of programmable connectivity.

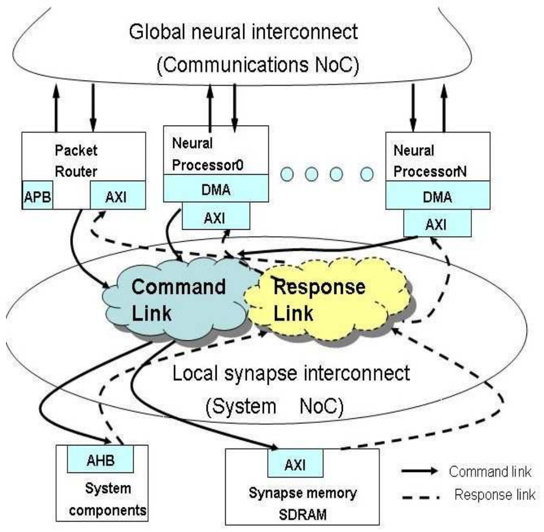

Fig. 1. Chip architecture

978-1-4244-1821-3/08/$25.00©2008 IEEE

II. SpiNNaker: a Universal Spiking Neural Network Chip

A. Parallel chip multiprocessor

SpiNNaker (fig. 1) is a chip designed to implement the general-purpose programmable neural device model. It forms a massively parallel computing system using chip multiprocessor (CMP) technology, where each chip contains about 20 ARM968 processing cores with on-chip and off-chip resources. Each chip functions as an independent functional unit, dynamically assigning one of the processor cores as a monitor processor to run a microkernel for chip management and configuring an on-chip routing mechanism to communicate with other chips. Each processor has a dedicated local memory attached to it: the tightly-coupled memory (TCM). The TCM has 2 parts: a 32KB instruction memory that contains the neuron modelling program itself, and a 64KB data memory that contains local synapse weight information and neuron activation information. When a neuron spikes, the CPU issues a spike packet that travels to a communications controller which then forwards the spike onto the network fabric. A single CPU would not in general implement a single neuron; rather it implements collections of neurons: in a typical configuration it might model 1000 individual neurons. Since the processor is programmable, it can implement essentially any model that will fit into the 32K instruction memory [9].

B. Virtual global network

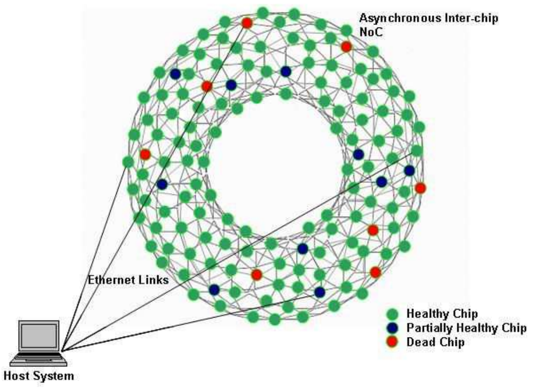

Fig. 2. SpiNNaker system architecture

The chip can function either alone or as part of a large parallel system of 1 to 65,536 interconnected SpiNNaker chips communicating over a long-range NoC connection fabric. When connected together, the chips form a closed toroidal mesh of interconnected chips as shown in fig. 2. SpiNNaker implements two distinct NoCs: a system NoC and a communications NoC. In the model, the communications NoC acts as the long-range connectivity fabric: the axonal connections between different neurons. The axons transmit purely spikes, which are simple events carried through the network as neural event packets carrying only the address of the source neuron that fired. The communications NoC has a throughput of 6 Gb/s and is responsible for carrying neural event packets between processing nodes, which can be in the same or different chips. A reconfigurable on-board router, capable of routing one packet per cycle at 200 MHz to its six external and 20 internal outputs, is responsible for routing packets containing spike events between nodes spread across the network. Each core contains a communications controller to formulate/decompose packets and send these to the router over the NoC. Together the NoC and routers in a complete system implement an arbitrary spiking neural network with a distributed connectivity map spread over the entire array of SpiNNaker chips.

C. Concurrent internal NoC

Where the communications NoC models axons, the second network, the system NoC, models the synapses and their dendritic connections. The system NoC provides a packet-switching infrastructure for synaptic weight updating, and replaces a conventional bus for on-chip component interconnect. Using an asynchronous fabric preserves the unclocked fire-and-forget dynamics of biological neural networks, isolating each processor from the others so that unrelated groups of neurons do not have implicit timing relations. It also confers some power savings and reduction of chip-level timing closure challenges [10]. As in fig. 1, the system NoC consists of two physical links based on Silistix' CHAIN technology [11]. The command link transmits address, control information, and any write data from processing nodes, to global memory and other system components. The response link returns any read data generated by the global memory to the nodes. Notice that any node can communicate with the global memory while a different node communicates with the other system components. With a large number of nodes, it becomes particularly important to share the interconnect.

D. Shared synapse memory

In a "typical" model, each core in SpiNNaker might implement 1000 neurons with 1000 synapses each. If each synapse requires 2-4 bytes to store its weight, this would mean each core needed at least 10^6 words (4MB) of storage, which would not be feasible using local memory alone. Synaptic weights therefore use a large, concurrently-accessed global memory for long-term storage. The memory is currently an off-chip mobile DDR SDRAM with 1 GB capacity. Since the SDRAM resides off-chip, it is easy to expand available global memory simply by using a larger memory device. Within SpiNNaker is a dedicated high-speed interface with support for multiple simultaneous requests, out-of-order request completion, and data buffering that can provide 1GB/s peak output bandwidth. This interface connects to the System NoC directly with an advanced AXI interface that allows concurrent multimaster access. Concurrent request support ensures that SDRAM accesses maintain the parallel processing model rather than imposing a sequential flow.

III. Virtual Synaptic Processing Implementation

A. Synapses mapped as hardware channels

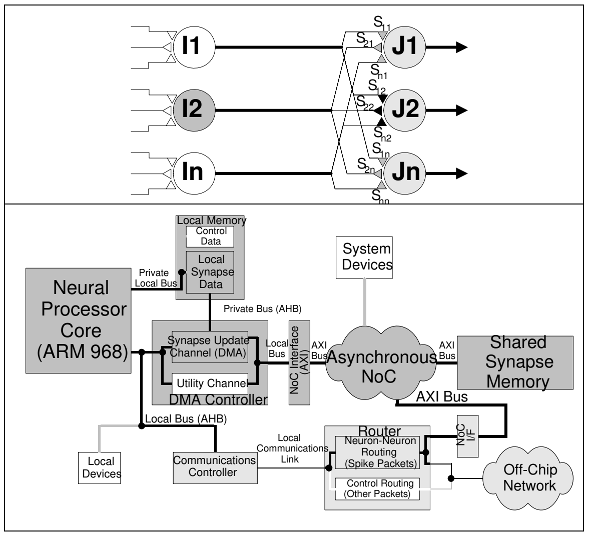

Fig. 3. The SpiNNaker synapse channel and its neural mapping

Within SpiNNaker, each processor performs the output and weight update calculations for all the synapses that connect to neurons resident in that particular processor. To implement these synapses, we introduce the hardware channel: a linked chain of associated components that together make up the synapse subsystem. While a shared SDRAM holds the long-term synaptic weights, the channel makes all synapses "virtually local" to the processor by swapping out the values in the dedicated memory local to each processor, at the time it performs the weight and output calculations for the synapse group in concern. As long as the time to swap in and out synapse values is small relative to the time between updates, each weight appears to be locally available, even though its stored value resides in SDRAM.

Fig. 3 details the channel and its correspondence to synapses. The channel consists of the local memory, a DMA controller for access to the SDRAM, an (AXI) interface to the asynchronous NoC, the NoC itself, an off-the-shelf SDRAM controller, and the SDRAM, together with their associated connections. In the figure, the ARM968 processor models neurons J1...Jn. A neuron I2 in another processor fires, activating the communications controller-router-off-chip network path in its chip. At the receiving chip, the processor loads synapse values into local memory for neurons J1...Jn sequentially. At any given time the local memory has values for active neurons. Thus, if J2 fires, synapses $S_{12}..S_{n2}$ will reside in local memory, while synapses $S_{11}...S_{nx\neq 2}$ are in global memory. The entire process would appear in the software model like Listing 1.

Because a channel has fixed data path widths throughout its system, the number of physical interconnect lines stays constant for all the neurons the local processor models, rather than scaling as a polynomial in the number of neurons. In contrast to previous shared-memory approaches, the channel removes, through the asynchronous NoC, the need for neurons to compete for serial synchronous access to shared memory. Provided that the channel can provide synapse data to the local memory before the time of the next update for that synapse, it retains concurrent spiking dynamics without introducing relative timing or data-dependent effects.

```
updateActivationLevels() {
receiveSpike(p);
//received packet p with source address
SDRAMB_lockBaseAddres=
    lookupTable.read(p.SourceID);
//DMA operation to read a block of memory
//from SDRAM to DTCM memory
requestDMAOperation(SDRAMB_lockBaseAddres,
 &LocalSynapseBlock, SynapticBlockSize, READ);
wait(DMACompletionInterrupt);
//for all neurons, update projected activation
    //Level as per the synaptic and delay
    //information
for (int i=0; i<SynapticBlockSize; i++)
{
    int delay=LocalSynapseBlock[i].Delay;
        //the real-time delay
    int neuronIndex=synapseBlock[i].NeuronIndex;
        //neuron index
    int weight=synapseBlock[i].SynapseWeight;
        //synaptic weight
    neuronsState[NeuronIndex].ActivationLevel[
 (currentTime+delay)] +=weight;
}}
```

Listing 1. Neuron update code

B. Biologically realistic synapse timescales

Firing rates for typical neurons are usually low. Although the extreme observed rate variability in real neurons preclude exact figures, upper limits for firing rates appear to be $\approx$ 100 Hz, with long-term averages being $\sim$ 10 Hz [14]. Furthermore, since neural networks use sparse population coding to encode information, typically only a small fraction of neurons are active at any one time - around 1 % in a typical case, to about 10 % maximum. A synapse is active only when the presynaptic neuron fires (although postsynaptic firings contribute to plasticity calculations) and thus the mean required update rate for the synapse is $\sim$ 0.1 Hz, rising to $\sim$ 10 Hz with active populations, and peak firing rates trending to the upper firing rate limit of 100 Hz. Meanwhile, axon propagation occurs with significant delay: up to 20ms. These time scales make it possible to exploit the great differences in speed between electronic signalling and neural response to achieve real-time modelling performance.

C. Synapse output calculations

The central function of the synapse is the input$\times$weight current injection function. In the case of the spiking model this operation is trivial from the processor's point of view: look up the value of the synapse (from its local memory) and accumulate it into the neuron's activation level. Given nonzero real axonal delays, if the processor can receive notification of input spikes to affected neurons sufficiently in advance of their real-time arrival at the neuron that the time to initiate a DMA request and load the synapses into local memory is less than the time from "electronic time" notification to real-time update, it will be able to respond as though the synapses had been continuously local. In the case of a single neuron this implies that

$$\delta + t_{DMA}(N_\nu) \leq t_{biol}/A_{\nu P} \quad (1)$$

where δ is the physical delay across the entire system, $t_{DMA}(N_\nu)$ is the time to execute a block DMA equal in size to the number of modelled synapses per neuron, t_{biol} is the modelling time step, and $A_{\nu P}$ is the number of active neurons modelled per processor time step. NoC wire delay is insignificant compared to DMA block transfer time, therefore a block transfer should take no more than 10 μs worst-case, 250 μs average-case, if the system models 1000 neurons per processor. It requires 8 clock cycles to initiate a block request, whereafter the system bursts data across the channel in 16-beat doubleword (64-bit) bursts running at 133 MHz at the NoC interface, 200 MHz 32-bit at the local memory interface. We see immediately that efficient DMA transfers that minimise protocol overhead will be essential if a processor is to have a realistic number of active neurons.

D. Weight updates

With spiking Hebbian models, synapse updates need only be calculated when the output neuron fires. The neural processor must be accumulating spike inputs in order to calculate neuron output, and from the considerations above, when it outputs a spike, the associated input synapses must be in local memory (since it is only the arrival of an input spike that can trigger the threshold-exceeding output spike). It can therefore compute the weight updates during the same processing cycle as it computes the neuron output, writing the new values back to memory with a block transfer at the end of the cycle. Spiking dynamics guarantees that synapses whose weights are to be updated will be in local memory, allowing the weight updates to occur rapidly and without incurring costly memory read-modify-write operations.

One model for weight updates we have examined envisions a row-compressed array of 16-bit weight values stored in memory within a region addressed by neuron number. Resident in the TCM for each processor is a neuron lookup table that gives the start address of its weight matrix in the SDRAM. When an input spike arrives, the local processor used the source neuron table to index the start addresses of the neurons connected to the source neuron. It then performs a DMA transfer to load the weights into memory, updates the output and weight values according to the weight update scheme of the particular neural model loaded into the local processor, and writes back the updated weights to the SDRAM with a DMA transfer. Importantly, although this model forms the basis of the design decisions, no aspect of the hardware itself limits the potential synaptic models to this specific implementation and thus it is possible to implement weight update mechanisms in a wide variety of approaches.

We have performed simulations of the system performance with 2 different neural network models. In one series we have kept closely to the reference spiking model, whereas in the other we have implemented a conventional feedforward multilayer perceptron to demonstrate SpiNNaker's capability to support radically different models employing distinctly different neuron and weight update mechanisms. Separate papers [12], [13] describe these experiments, therefore we will not repeat the details here. However, the main results indicate the following. A given processor can hold the values for approximately 1400 neurons within the local memory space. Modelling 1000 neurons per processor appears feasible and supported firing rates of up to 67 Hz, assuming 10% connectivity. The corresponding weight update rate is 100,000/ms, with each weight update taking 110 ns. We emphasize, however, that these are very preliminary results. Much work remains to be done to verify learning performance and develop effective dynamic-update schemes.

IV. NoC: Architectural Considerations

A. Local data pump: datapath configuration

Directly performing large, block-oriented transfers between local memory and system memory under CPU control would be inefficient. The obvious solution is a DMA controller to manage the memory transfers. While virtual synapse updates generally use DMA, shorter memory accesses, or accesses to other devices in the system, are, conversely, inefficient using this mechanism. Furthermore, they would require an expensive multichannel controller if accesses were to be non-blocking. The DMA controller therefore implements a second, pass-through channel that permits direct processor access to the asynchronous NoC for short transactions.

Design of the data buffer for the controller is critical. Since the buffer must support both burst DMA transfers and non-burst direct-access transfers, it consists in fact of 2 independent buffers. However, to increase flexibility the buffers are both reversible in direction and in interface identification. Therefore either buffer can act as a burst-mode buffer or a direct-mode buffer, operating either for read or for write. The buffer also supports an optional double-buffered mode, allowing one interface to use both buffers at the expense of blocking transactions on the other interface until the current transaction completes. Both word width and buffer depth, in number of entries, are configurable parameters so it can adapt to different bus interfaces and utilisation rates. Interconnect timings remain indeterminate; therefore by designing the buffer as a synchronised dual-clock FIFO, there need be no fixed relation between input and output timings. Thus the design of the buffer structure itself accommodates various rates and asymmetries of transfers between different interfaces.

B. Local data pump: protocol translation

SpiNNaker's asynchronous NoC uses ARM's AXI interface specification to connect to the DMA controller [15]. AXI supports multimaster burst-mode transactions with "fire-and-forget" out-of-order completion, ideal for an asynchronous interconnect where responses from different requests arrive independently of each other. The "back-end" interface to the ARM and to local memory, however, is not AXI, and in particular it does not have native support for "fire-and-forget" out-of-order transaction completion. To translate between bus protocols, we have developed a universal master/slave controller designed to interface between the AXI bus and a system device with minimal restrictions on the back-end interface. Since this component is replicated multiple times across the chip, it is essential to implement a "thin" interface, occupying minimal area per device. This makes it especially important that the size of data buffers and state-holding registers stay small. By parameterising the implementation to omit support for exclusive transactions and big-endian/little-endian reordering, accept a maximum of 2 outstanding transactions, and buffer only the current word in the transaction (thus containing no internal burst buffering), the controller retains AXI compliance while eliminating 3 area-expensive register banks in both master and slave components.

C. Asynchronous NoC: fair bandwidth allocation

As we discussed in "Synapse output calculations", for synaptic output updates to retain real-time behaviour, a neuron must not receive the next input until it completes calculating the sum of synaptic weights. Because typical neural input-current models perform this calculation iteratively, the transmission rate on the asynchronous NoC is crucial for real-time synaptic output performance as well as weight updating.

In the SpiNNaker chip, each processing node has a local memory for storing weights and activation information corresponding to neurons of the neural network. The SDRAM stores the long-term weights for the nodes, providing access via the system NoC. To use computing resources fairly, each node requires the asynchronous NoC to coordinate bandwidth allocation to SDRAM in such a way as to ensure fair sharing. This is nontrivial, because since the NoC is a packet-switched, not a circuit-switched network infrastructure, packets have different physical transmission paths. The asynchronous mutex arbiter is nondeterministic and therefore by itself can provide no QoS guarantees. However, if the depth of the SDRAM command FIFO is long enough to accommodate all requests, and one processing node only sends one outstanding request, the risk of unfair sharing disappears. Consider a case where initially, all nodes have an equal probability of gaining access to the SDRAM (with no outstanding requests). Now, nodes initiate requests. Once the SDRAM has buffered all current requests from processing nodes, it dequeues the current request of a given node from the SDRAM FIFO. At this point, that node cannot immediately access the SDRAM due to the first-come-first-served servicing model in the FIFO. This node must wait until the SDRAM controller has serviced current requests from the other active processors. Therefore, all nodes gain equal bandwidth allocation. This observation allows us confidently to simplify the hardware design and decrease unnecessary hardware overhead.

D. Asynchronous NoC: concurrent accessibility

The key feature of the asynchronous NoC is a partial crossbar topology between processing nodes and SDRAM. This topology enables multiple nodes to access SDRAM simultaneously. However, the single SDRAM obviously becomes a bottleneck when *all* nodes require access to the SDRAM interface.

We can ease the physical bottleneck by taking biologically realistic values of the time step for synaptic output and weight updating. Synapses do not only transmit the signals; they may change their value during processing. Synaptic weight updating reads a large amount of data (typically $\sim 2-4$KB) from the global memory (SDRAM) into the local memory and writes it back into the global memory again when the neuron spikes. Hence, the bandwidth *to the local node* dominates synaptic weight updating, instead of the *average* bandwidth of the SDRAM. In addition, the asynchronous NoC guarantees concurrent accessibility even when the SDRAM interface is operating at capacity. In other words, saturation of a single SDRAM can not have a significant impact upon the asynchronous NoC because of the two physical links in the NoC. For example, all processing nodes require block reads to occur in bursts of 16 words. The amount of read data is enough to saturate the response link. However, the command link is not likely to be saturated because each node only issues one command. Successive commands can go through the asynchronous NoC to the SDRAM interface without causing congestion. The SDRAM interface can achieve full bandwidth utilisation because it can still receive continuous commands.

V. NoC: Implementation Analysis

A. Nonblocking throughput at the local node

Burst traffic on the asynchronous NoC sends only one response (acknowledgement) at the end of the transaction. An important consequence of this is that burst data is blocking with respect to the fabric until the receiving device accepts the data, and this would introduce the potential for severe stalling in a high-utilisation scenario, with multiple masters each attempting to compete for the same finite NoC resource. In turn this means that individual masters must be able to offload a complete data burst from the network without stalling. Furthermore, requests on the back-end bus are always blocking with respect to ARM instruction execution, leading to the equally undesirable situation where the local processor stalled awaiting an asynchronous response from a DMA request in transit when it initiated a new direct-access transfers. This combination of potential stall situations led to the following implementation decisions.

- 1) Each of the 2 data buffers in the DMA controller can support a full burst up to the maximum AXI burst length: 16 64-bit doublewords.

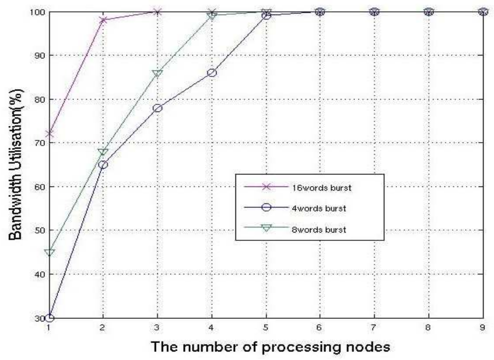

Fig. 4. Bandwidth Utilisation vs. Number of Processing Nodes

- 2). The AXI interface buffers a maximum of 2 double-words, the second buffer only being used as an overflow in the case of an AXI stall.
- 3) The direct request channel includes support for write buffering so that the ARM can complete a pending write even if the AXI interface is busy.
- 4) Direct transactions when the AXI interface is not busy bypass the DMA buffer entirely and use only the AXI interface buffer.
- 5) Transaction priority on the DMA controller's AXI interface is: first, buffered direct writes; second, unbuffered direct transactions; third, DMA transactions.

Analysis of latencies under various transaction scenarios with this combination of options revealed only one where potential stalling could become severe: an ARM direct read request when the AXI interface is busy on a DMA burst transaction; other transactions had worst-case latencies of 4 local clock cycles.

B. Interconnect performance evaluation

The asynchronous NoC is generated by the CHAINworks tool suite, based on delay-insensitive communication links, as described above, with an additional wire to encode an end-of-packet symbol [11]. The implementation of the asynchronous NoC employs multiple delay-insensitive links to deliver the throughput required by the processing node [16]. On 130nm process technology, simulations show a throughput of 1Gbit/s per link(4 bit for 3-of-6 encoding). For a 48 bit wide network it can reach 12Gbit/sec raw bandwidth.

In SpiNNaker, we use a single 1 Gbit external mobile DDR SDRAM device, providing a large shared memory resource for storing synaptic weights. The bandwidth available at the SDRAM interface is around 1GByte/s - less than the asynchronous raw bandwidth. Even though SDRAM capability limits the average bandwidth, provided we can fully utilise the SDRAM interface, the 1GByte/s SDRAM bandwidth is sufficient communication for synaptic weight updates under realistic biological time scales.

Figure 4 shows we can achieve 100% SDRAM bandwidth utilisation as the number of processing nodes increases. It is critical with larger numbers of nodes (5) that the spiking neural model itself be maximally efficient.

C. Buffer placement

In theory, neural modelling is scalable in SpiNNaker, both in terms of the number of neurons and the number of synapses. In practice, silicon area overhead limits the scale of achievable neural networks. With respect to area, buffers dominate the overhead. It is necessary to consider tradeoffs between system performance and silicon area overhead carefully: even though buffers could potentially speed up the transmission, they come at considerable area cost.

In our system, there are two buffer placement possibilities: one is in the asynchronous adapters; the other is in devices connecting to the NoC. Adding request reordering buffers in the asynchronous adapter connecting to the SDRAM controller would improve data throughput by supporting multiple addresses pending in the adapter before their acceptance into the SDRAM FIFO. In table I, the number of reorder buffers indicate the number of outstanding requests the target device can support. If the number of processing nodes is equal to the number of reorder buffers at the SDRAM controller, concurrent read commands from all processors can arrive at the SDRAM simultaneously, achieving fine-grained parallelism.

No. of Reorder Buffers	Cell Size
1	0.04mm^2
3	0.24mm^2
3	0.33mm^2
4	0.43mm^2
5	0.55mm^2

TABLE I
COMPARISON OF NOC REORDER BUFFER NUMBER TO CELL AREA

Synthesis results, however, show that implementing buffers in the adapter is inefficient. Per table I, under UMC 130nm process technology, the cell size of the adapter with 2 reorder buffers is approximately twice as large as that with 1 reorder buffer. By contrast, placing the buffers in the DMA controller is considerably more efficient. Table II shows that, even a large buffer with 2 16-entry FIFOs, synthesized for 200 MHz, occupies about 0.055mm^2. Therefore, most buffers should be in the devices to achieve optimised bandwidth with minimal area.

Number Of Buffers	Area
1	9893μm^2
2	12699μm^2
4	19515μm^2
8	31642μm^2
16	55736μm^2
32	104198μm^2

TABLE II
DMA DATA BUFFER AREA SCALING

D. Protocol implications and restrictions

AXI's advanced transaction features create several potential deadlock scenarios, and these in combination with the area and utilisation constraints we have already observed make it necessary to impose certain additional protocol restrictions. One of the most unusual emerged during design of the AXI interface. AXI has separate handshake and payload channels for data and address. It is highly desirable to retain AXI's support for out-of-order processing, and this means that requests, responses, and data may arrive in any sequence. To associate responses and data with their corresponding requests, slave AXI devices must have a transaction scoreboard with a number of entries equal to the number of concurrent requests the slave can handle. If, however, the scoreboard is full, the slave would have to refuse new requests, and with multiple masters communicating independently there can be no guarantees such a new request would not arrive. Meanwhile, the same slave device must accept incoming data, for otherwise it would deadlock existing uncompleted transactions that were in the scoreboard. Simulations confirmed that CHAIN does not supply master-sensitive data acknowledge, so that if a master initiated a new request to a slave whose scoreboard were full, presenting data and addresses simultaneously, the slave would have to accept the data while refusing the address, eventually deadlocking the system.

NoC fair access considerations imply a restriction to one outstanding DMA transaction per master, a condition already guaranteed by the DMA controller design. Direct-request transactions, however, are not intrinsically subject to this restriction, and since such transactions are blocking on a per-word basis, we impose the additional requirement that direct requests be limited to a single word. By implementing the AXI-side burst counter in the DMA as a shared counter between the 2 sources of potential request we have been able to enforce these restrictions in hardware. While the protocol implications we have observed are nonintuitive, they are consistent with the expected use of the 2 channels, the DMA channel being used for dedicated synapse burst access and the direct channel for general-purpose, but incidental, non-burst requests.

E. Future Work

Where the implementation to date has focussed on developing a working basic system, future work will aim in the first place at extending the model to larger and more complex systems, and in the second on detailed investigation of optimal mappings and translation of real neural architectures onto the physical hardware. Thus far our considerations have assumed a "reference" Izhikevich [17] model neuron with simple STDP learning mechanisms, but it will be essential to generalise the configuration methodology to other neural network models. We are developing a configuration framework that allows description of the neural model at an abstract level and plan to demonstrate the implementation of multiple different types of neural network exemplifying radically different models to validate the ability of the design to implement an arbitrary neural network.

More work remains on identifying efficient methods to implement the weight updating. Of critical significance is the timing, for if the synaptic values expire from local memory before the end of a learning widow (e.g. in the case of an input spike arriving after its target neuron has fired), it will require an expensive second read to load the synapse back into memory. The synapse channel also exists in the context of a system whose global connectivity, through the routing table, is in principle dynamically reconfigurable at run time. How synapse updates may be maintained consistent in such an environment remains an open issue. Meanwhile, models exhibiting completely different update dynamics, such as the MLP model we considered, require methods to correlate weight update times with those synapses whose values change in a given update epoch, since they do not have the association with input events that the reference spiking model does. How learning should be scheduled in a large, complex system will therefore be a critical future research focus.

Thus far the largest model size we have been able to consider is a 500-chip system implementing approximately 4500 neurons, a function of memory limitations in the simulation environment. We are implementing a fully-functional test chip in advance of the full-scale device, and by migrating the simulations to the test platform aim both to increase the model size and identify potential future enhancements to the hardware components. Work continues on performance evaluation of the NoC under fully-loaded conditions, and we intend to generate approximate upper bounds on numbers of neurons and synapses per chip, and number of updates per second. Future SpiNNaker devices may employ variant NoC or DMA interfaces with capabilities and protocols optimised for neural modelling applications. One important research direction is topologies, memory update methods, and network protocols for managing densely connected networks such as cerebellar Purkinje cells which may require distributed implementations spread over several processors on a chip or several chips in a system. Work to date suggests certain features of the optimal hardware architecture for parallel applications like neural networks, and we plan to pursue these ideas further.

VI. Conclusion: An NoC Map for Spiking Neurons

A. Asynchronous fire-and-forget interfacing

We have demonstrated here an architecture to circumvent the problem of quadratically increasing wiring densities that is adequate for synaptic processing in large neural networks at biologically realistic speeds. Our multiprocessor system was not only introduced in an attempt to provide large-scale neural modelling, but also to solve the fundamental computational problem of parallel communication for multiprocessor chips. Analysis of the tradeoffs to be made in a real implementation of the architecture reveals general insights that make it productive to ask: what would the *ideal* NoC look like? Given neural networks' inherently parallel structure, the key requirement on the NoC in neural applications is to provide *nonblocking concurrent access to shared resources*. If this is to be possible, transactions must be fire-and-forget: once initiated, the initiator does not await any response from the target. Ideally, indeed, both initiator and target should not await any form of handshake from another device: once ready to present data, they present it and release the NoC immediately. One way to implement this is for responders to accumulate requests in a buffer whose depth is $n + 2 * \delta_{resp}, n$ being a number of requests after which the device must issue a response before continuing, δ_{resp} being the propagation delay of the response to the most distant requestor.

B. Out-of-order multimaster burst handling

Concurrent initiators can provide no ordering guarantees to the NoC, and therefore if the NoC supports nonblocking access it must interleave burst data from different sources of request on the same fabric without regard to request order. With no other way to identify a given request, slaves must be able to decode the address before accepting write data, and this in turn implies that addresses can arrive no earlier that at the same time as the data. Ideally, then, the NoC should combine address and data into a single channel, and likewise the interface to devices should present this channel with a single handshake (in contrast, e.g. to AXI which separates address and data channels)

C. Symmetric dedicated transfer backoff

Utilisation and stalling can present severe concerns if a given device can hog the interconnect fabric, while deadlock is a risk with any system whose slave devices cannot simultaneously service every possible device. It should therefore be possible for any given device in the system to refuse or abandon a transaction from another given device, without affecting its ability to service transactions from other devices. Therefore masters and slaves must have the ability to assert a request or acknowledge on a device-wise basis. If, furthermore, both sides use asynchronous fire-and-forget protocol, request/response lines should be a field, translating to a packet on the NoC, containing the encoded device address to which the other is responding, and the standard request-acknowledge pair of bits. There is a correlation here with Hebbian learning: in STDP mechanisms, postsynaptic neurons weaken or strengthen individual connections depending on whether they have just fired, just as NoC devices that have reached the request limit suppress new inputs while permitting data from existing requests, by selective acknowledge assertion.

D. Neural networks, asynchronous NoC's: models for each other?

We conclude here with an interesting observation: if one combines these properties for the ideal NoC, we arrive at a structure remarkably analogous to spiking neurons. Consider that asynchronous fire-and-forget corresponds well to spike signalling, that out-of-order combined data/address processing is comparable to the integration function of a neuron, and that symmetric transfer backoff is a similar process to Hebbian learning in synapses. Although far from identical in function, as the airplane is to the bird, such striking similarities suggest that in addition to being a useful practical tool for synaptic interconnect implementation, the virtual synapse channel may be a model for examining the signalling dynamics of neural networks.

ACKNOWLEDGMENT

We would like to thank the Engineering and Physical Sciences Research Council (EPSRC), Silistix, and ARM for generous support of this research. Further thanks is due to Silistix for continuing technical and tool update support, and to ARM for clarification of protocol issues related to the AXI interface. S. Yang would also like to thank the School of Computer and Communication, Hunan University, Hunan, China for ongoing financial support. S.B. Furber is the recipient of a Royal Society Wolfson Merit Award.

REFERENCES

[1] S. B. Furber and S. Temple, "Neural systems engineering" *J. Royal Society Interface*, vol. **4**(13), pp. 193-206, April 2007
[2] H. Markram, M. Tsodyks, "Redistribution of Synaptic Efficacy Between Neocortical Pyramidal Neurons" *Nature*, vol. **382**, pp. 807-810, 1996.
[3] E.M. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?" *IEEE Trans. Neural Networks*, vol. **14**(6), Nov. 2003.
[4] A. Jahnke, T. Schönauer, U. Roth, K. Mohraz, H. Klar, "Simulation of Spiking Neural Networks on Different Hardware Platforms" *Proc. 7th Int'l Conf. on Artificial Neural Networks (ICANN '97)*, 1997.
[5] G. Indiveri, E. Chicca and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity" *IEEE Trans. Neural Networks*, vol. **17**(1), Jan. 2006.
[6] T. Schönauer, N. Mehrtash, A. Jahnke and H. Klar, "MASPINN: Novel Concepts for a NeuroAccelerator for Spiking Neural Networks" *Proc. SPIE - VIDYNN '98*, vol. **3728**, 1999.
[7] N. Mehrtash, D. Jung, H.H. Hellmich, T. Schönauer, Vi Thanh Lu and H. Klar, "Synaptic plasticity in spiking neural networks (SP^{2}INN): a system approach" *IEEE Trans. Neural Networks*, vol. **14**(5), Sept. 2003.
[8] A. Upegui, C. A. Peña-Reyes, E Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks" *Microprocessors and Microsystems*, vol. **29**(5), June 2005.
[9] S. Furber , S. Temple, A. Brown, "On-Chip and Inter-Chip networks for Modelling Large-Scale Neural Systems" *Proc. Int'l Symp. on Circuits and Systems, (ISCAS-2006)*, pp. 21-24, May 2006.
[10] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Yebin Shi, Jian Wu and Shufan Yang, "A GALS Infrastructure for a Massively Parallel Multiprocessor" *IEEE Design & Test of Computers*, vol. **24**(5), pp. 454-463, July 2007.
[11] J. Bainbridge and S. B. Furber, "CHAIN: a Delay-insensitive chip area interconnect" *IEEE Micro*, vol. **22**(5), pp. 16-23, Sept. 2004.
[12] Xin Jin, S. B. Furber and J.V. Woods, "Efficient Modelling of Spiking Neural Networks on a Scalable Chip Multiprocessor" To appear in *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.
[13] S.B. Furber, M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin and E. Painkras, "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor" To appear in *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.
[14] P. Dayan and L. F. Abbott, *Theoretical Neuroscience*, pp. 3-44, Cambridge, MA: MIT Press 2001.
[15] ARM Ltd, "AMBA Specification", *www.arm.com*, 2007.
[16] S. B. Furber, "Future Trends in SoC Interconnect" *Proc. IEEE Int'l Symp. on Design, Automation and Test (VLSI-TSA-DAT)*, pp. 290-293, 2005.
[17] E.M. Izhikevich, "Simple Model of Spiking Neurons", *IEEE Trans. Neural Networks*, vol. **14**(6), 2003.