

An Event-Driven Model for the SpiNNaker Virtual Synaptic Channel

Alexander Rast, Francesco Galluppi, Sergio Davies, Luis A. Plana, Thomas Sharp, and Steve Furber

Abstract—Neural networks present a fundamentally different model of computation from conventional sequential hardware, making it inefficient for very-large-scale models. Current neuromorphic devices do not yet offer a fully satisfactory solution even though they have improved simulation performance, in part because of fixed hardware, in part because of poor software support. SpiNNaker introduces a different approach, the “neuromimetic” architecture, that maintains the neural optimisation of dedicated chips while offering FPGA-like universal configurability. Central to this parallel multiprocessor is an asynchronous event-driven model that uses interrupt-generating dedicated hardware on the chip to support real-time neural simulation. In turn this requires an event-driven software model: a rethink as fundamental as that of the hardware. We examine this event-driven software model for an important hardware subsystem, the previously-introduced virtual synaptic channel. Using a scheduler-based system service architecture, the software can “hide” low-level processes and events from models so that the only event the model sees is “spike received”. Results from simulation on-chip demonstrate the robustness of the system even in the presence of extremely bursty, unpredictable traffic, but also expose important model-level tradeoffs that are a consequence of the physical nature of the SpiNNaker chip. This event-driven subsystem is the first component of a library-based development system that allows the user to describe a model in a high-level neural description environment and be able to rely on a lower layer of system services to execute the model efficiently on SpiNNaker. Such a system realises a general-purpose platform that can generate an arbitrary neural network and run it with hardware speed and scale.

I. EVENT-DRIVEN HARDWARE: A NEED FOR AN EVENT-DRIVEN SOFTWARE MODEL

SIMULATION of large-scale neural networks is difficult and can generate tremendous demands on communications capacity, processing speed, and memory usage. If there is a need to run the network at real-time speeds, for example, when interacting with a real-world environment, these challenges become almost insurmountable with conventional synchronous hardware that, even below the heavy layering of necessary system software, uses a model of computation different from and rather unsuitable for spiking neural networks. Thus on the one hand there is a need for dedicated hardware that can implement neural networks explicitly, and on the other for an event-driven model of computation at both the hardware and software level that matches the spiking model of neural computation.

The authors are with the School of Computer Science, University of Manchester, Manchester, UK (email: {rast, fgalluppi, daviess, plana, sharp, sfurber}@cs.man.ac.uk).

This work was supported by the Engineering and Physical Sciences Research Council, ARM, Inc, and Siliistix.

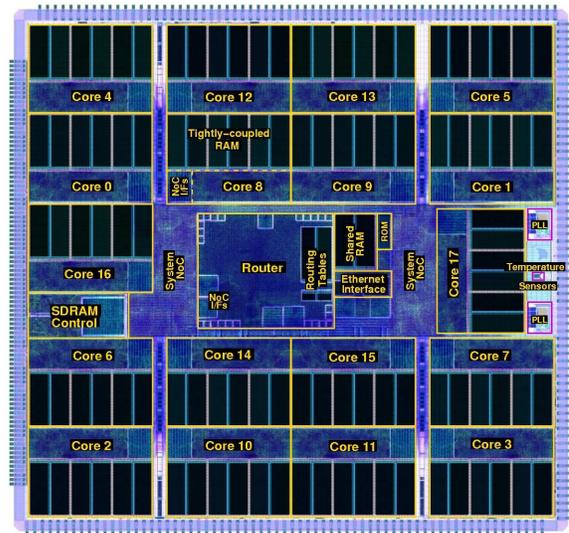


Fig. 1. SpiNNaker chip.

Dedicated neural hardware has been attractive for some time, forming the basis of the neuromorphic architecture and a series of associated chips [1], [2]. One difficulty with these chips is that they have typically included analogue components that are either fixed-function or offer at best limited configurability. This hard-wired nature severely constrains the ability to use neuromorphic chips to explore neural models, because they limit the experimenter to the class of neural model the chip has implemented in hardware, making inter-model comparisons impossible - and unless the chip matches the experimenter’s model of choice, it is of little value. Recently, we have introduced a new form of neural hardware: the SpiNNaker chip (fig. 1), a general-purpose platform containing programmable general-purpose processors embedded in a configurable asynchronous network. This architecture retains the essential massive parallelism and rewirable network topology but allows freedom in model choice - perhaps closer to the “ideal” neural hardware architecture.

There remains a major gap in tool support: neuromorphic architectures tend not to integrate seamlessly into existing simulators [3]. The user cannot therefore simply run a predeveloped model on the hardware: it needs translation, assuming, furthermore, that the target model is compatible with what the hardware supports [4]. Being outside mainstream commercial technology, such translation tools are scanty to nonexistent [3]. There has tended to be no tool chain

other than a basic interface, usually in the form of a device driver [5]. With minimal tool support, using such chips has, historically, been difficult [6]. The would-be modeller usually has to have intimate knowledge of the low-level details of the hardware. Integrating these chips, containing proprietary interfaces, into larger systems is equally difficult because the entire system must be built from the ground up [7]. Bare hardware without appropriate development tools is in effect useless: there is thus a need for an integrated tool chain for neural hardware.

The tool support problem is particularly acute if the architecture is event-driven; event-driven software is a specialist field with limited tools even at a basic level, and scant literature. Much of the knowledge seems to pass more by word of mouth than by publication. When it is difficult, such as here, to get even standard reference models to run, an understandable reaction of the biological modeller is to express scepticism about the validity of the models the chip implements [8] - because he cannot test it against a model he is familiar with and trusts [9]. Thus neuromorphic chips have not to date provided a compelling platform for discovering valid biological abstractions, because it has been difficult to make direct comparisons against detailed models. Here we introduce an event-driven tool chain, built from the ground up to support asynchronous, spiking neural networks on dedicated hardware platforms - an essential for large-scale real-time neural network modelling.

II. EVENT-DRIVEN NEURAL MODELLING

A. Pure Software Simulation

Software simulation of spiking neural networks tends to be slow and may require large computers for detailed simulations on large-scale models [10]. To improve performance, recent software tools have turned to event-driven computing [11] [12]. Most such simulators actually run an event-driven emulation by using a small timestep, recording events in an event queue, and updating all processes dependent upon the events in the queue at the appropriate timestep [13]. While this improves efficiency over fully synchronous approaches, it still encounters limitations with very large networks that require either using simple dynamics such as leaky integrate-and-fire, or modelling populations of neurons as a single object rather than each individual neuron.

B. Event-Driven General-Purpose Hardware

The emergence of various general-purpose devices supporting some level of parallel processing has generated numerous attempts to map various neural algorithms to the hardware. A remarkable early attempt using a processor with strong similarities to SpiNNaker, the Datawave chip [14] appears not to have been pursued further because of the limited commercial success and eventual disappearance of the hardware. While the increasing ubiquity of standard multicore microprocessors and graphics processors (GPU's) [15] introduces an obvious opportunity to exploit parallelism, these devices use an emphatically synchronous, coherent

model. Some approaches attempt to exploit the reconfigurability of field-programmable gate arrays (FPGA's) and were among the first to hint at an event-driven architecture [16]. However, attempts to create reconfigurable or event-driven dynamics do not appear in most cases to have proceeded beyond the design-exploration phase [13]. A severe limitation with FPGA's is the circuit-switched architecture, which hampers the ability to create the dense connectivity patterns realistic neural networks require [17]. Even more problematic has been power consumption: a typical large FPGA may dissipate $\sim 50W$. Thus adapting general-purpose FPGA's for neural networks has been useful for design exploration, but not widely pursued for full-scale modelling [17].

C. Neuromorphic Hardware

Dedicated "neuromorphic" devices have generated the greatest level of research activity in event-driven models. Spiking networks make it possible to abstract the signalling to a zero-time point process, and this forms the basis of the emerging neural data serialisation standard: Address-Event Representation (AER) [18]. AER uses packets that encode the source of the spike as an address and is a proven, efficient way to serialise and then multiplex multiple neural signals onto the same series of lines [19] while making the converters themselves trivial [20]. AER is well established on the way to becoming a defined standard [1], thus making it overwhelmingly the signalling method of choice for future neural designs. Nothing limits AER signalling to mixed-signal devices [21], and thus it can be the basis of a "template" for neural hardware. The system that emerges is a chip containing blocks of configurable functionality, possibly mixed-signal [22], embedded in a connectivity network using AER signalling, using a standard modelling tool chain that likewise uses an event-driven model: the "neuromimetic" architecture [23].

III. THE SPINNAKER ASYNCHRONOUS EVENT-DRIVEN ARCHITECTURE

SpiNNaker (fig. 3) integrates the essential elements of the neuromimetic architecture: a hardware model designed to support flexibility in model exploration while implementing as many known features of the neural model of computation explicitly in hardware for maximal performance ([24]). The SpiNNaker chip is the core building block component of a large-scale system using an array of chips arranged in a 2-dimensional triangular torus topology (fig. 2). Using this diagonal-link topology increases system robustness through the connection redundancy inherent in the toroidal physical topology, while permitting an arbitrary mapping of large-scale neural networks to physical chips and links.

SpiNNaker implements the key architectural features using a mixture of off-the-shelf and custom components. We identify four features as fundamental to the neuromimetic architecture.

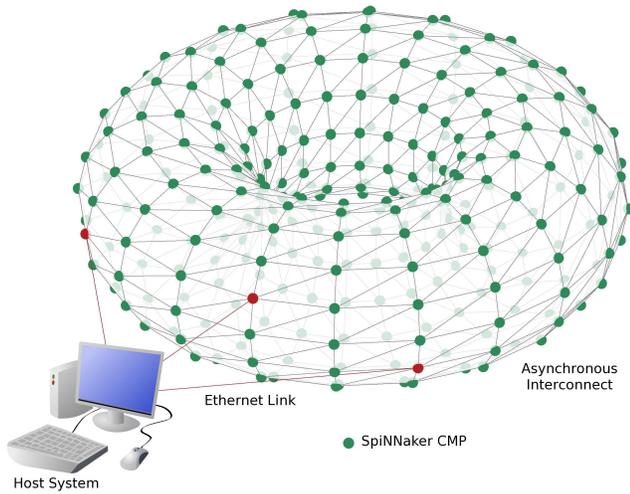


Fig. 2. SpiNNaker system topology.

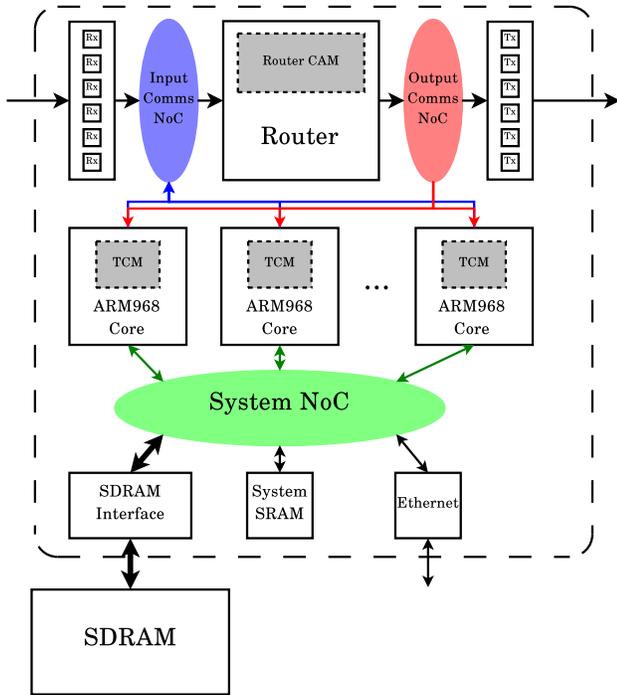


Fig. 3. SpiNNaker Architecture. The dashed box indicates the extent of the SpiNNaker chip. Dotted grey boxes indicate local memory areas.

A. Concurrent Multineuron Processing

SpiNNaker contains multiple (2 in the test chip used for current simulations, 18 in the full version) independent ARM968 processors, each simulating a variable number of neurons which could be as few as 1 or as many as 1,700. Each processor operates entirely independently (on separate clocks) and has its own private subsystem containing various event-generating devices to support neural functionality. These are: a communications controller that handles input and output traffic in the form of “spike” packets, a DMA controller that provides fast virtual access to synaptic data

residing off-chip in a separate memory, and a Timer that supports the generation of periodic events where models need them. This “processing node” operates asynchronously from other processing nodes using an event-driven model: local interrupts from devices and software control execution.

B. Asynchronous Event-Driven Communications

SpiNNaker’s communication network is a configurable packet-switched asynchronous interconnect using Address-Event Representation (AER) to transmit neural signals between processors. SpiNNaker extends the basic address-only standard with an optional 32-bit payload. The interconnect itself extends both on-chip and off-chip as the Communications Network-on-Chip (Comms NoC) [25]. At the processor node, the communications controller receives and generates AER spikes, issuing an interrupt (i.e., an event) to the processor when a new packet arrives. The communications fabric itself has very minimal buffering, therefore it is important that packets be serviced as quickly as possible. Such events, normally corresponding to spikes, are *asynchronous*: they may happen at any time as a process “dropping from the sky” so to speak, and therefore changing the process flow.

C. Reconfigurable Structure

SpiNNaker uses a distributed routing subsystem to direct AER packets through the Comms NoC. Each chip has a packet-switching router that handles these packets and distributes them seamlessly to all connected neurons through the GALS interconnect. The router incorporates a multicast diffusion mechanism devised to support biologically realistic neural fan-out (~ 1000 connections/neuron). A 1024-word associative routing table within each router defines the neural connectivity. Routes are fully reprogrammable by changing the routing table, making it possible, in principle, to reconfigure the model topology on the fly (although these capabilities have not yet been explored).

D. Virtually-Private Distributed Memory

SpiNNaker processors have access to 2 primary memory resources: their own local “Tightly-Coupled Memory” (TCM) and a chip-wide SDRAM device. The TCM is only accessible to its own processor and contains both the executing code (in the 32 KB “Instruction TCM” (ITCM)) and any variables that must be accessible on-demand (in the 64 KB “Data TCM” (DTCM)). The off-chip SDRAM contains the synaptic data (and possibly other large data structures whose need can be triggered by an event). Since synapses in the SDRAM always connect 2 specific neurons, which themselves individually map to a definite processor (not necessarily the same for both neurons), it is possible to segment the SDRAM into discrete regions for each processor, grouped by postsynaptic neuron, with incoming spikes carrying presynaptic neuron information. At the processor node level, the DMA controller handles synaptic data transfer, making the synapse appear virtually local to the processor by bringing it into DTCM when an incoming packet arrives ([26]). The DMA controller also generates an event - DMA complete

- when the entire synaptic block has been transferred into local memory. Overall therefore, the SDRAM behaves more as an extension of local memory into a large off-chip area than a shared memory area, and thus from a system point of view, effectively all memory is local.

IV. THE SYNAPSE CHANNEL IN SOFTWARE

A. The Synapse Channel Architecture

The subsystem - the “virtual synaptic channel” - that, triggered by an incoming spike, accesses the synaptic data, executes a DMA transfer, and brings the synaptic data into local memory, then stores them back when the update is complete, is one of the critical components of the SpiN-Naker execution model. We have previously introduced the hardware model for the synapse channel in [26], and various algorithms for the synapses themselves in [27], [28], [29]. At a low level, however, it interacts tightly with two interrupt sources (and devices): the communications controller and the DMA controller, even though from a model point of view, only one event - spike arrived - occurs. The underlying software model must therefore be able to hide the DMA operations and events from the user-level model, while at the same time generating additional internal (software) events to trigger writeback once the synaptic data has been processed. It also needs to use the limited local memory and processing resources efficiently, since there is a 64K local data memory limit, and a (nominally 1 ms) time limit to complete operations in order to maintain real-time update. These considerations lead to a fully event-driven architecture for the synapse channel at the software level.

B. Model-Level View

From the point of view of the *model*, there is only one event: “spike arrived”. However, rather than a single monolithic event-triggered process, it is better to break down the processing into a series of steps, executing in an event-driven pipeline. By considering what hardware can usually implement efficiently, in combination with observations about the nature of processing in typical neural models, we have created a generalised function pipeline to represent a neural process that is adequate for most models (fig. 4).

For the implementation of the neural and synaptic models, three considerations emerge that interact with the event processing. First, the spike arrived event is a high-priority FIQ interrupt, because spikes have a “use-it-or-lose-it” nature: new packets must receive immediate, pre-emptive servicing or they will be lost. Second, since we use a periodic hardware Timer event to drive an Euler-method differential equation solver in the last stage, all other stages must complete within the interval (nominally 1 ms) of a pair of Timer events or the system will lose real time. Third, the virtual synaptic channel model stores synaptic data off-chip in SDRAM, bringing it into local memory via DMA operations [26] which generate events upon completion. There needs to be a system software layer that controls SpiNNaker’s hardware devices to make all these events and processes transparent to the model.

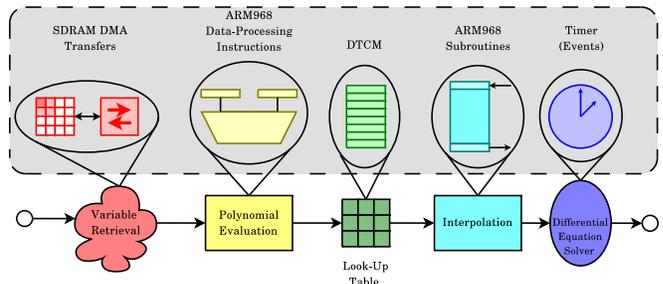


Fig. 4. A general event-driven function pipeline for neural networks. Variable retrieval recovers values stored from deferred-event processes as well as local values. Polynomial evaluation computes simple functions expressible as multiply and accumulate operations. These then can form the input to lookup table evaluation for more complex functions. Polynomial interpolation improves achieved precision where necessary, and then finally the differential equation solver can evaluate the expression (via Euler-method integration). Each of these stages is optional (or evaluates to the identity function).

C. Device-Level View

At the device level, the synapse channel consists of the SDRAM (where synaptic weights reside), the System NoC (simply the data link between the SDRAM and the processor node) the DMA controller (which transfers the data over the System NoC), and the local DTCM. The communications controller activates the synapse channel when a packet arrives. Obviously, the communications controller and DMA controller provide 2 interrupt sources (events) to control the synapse channel. To complete the channel, the software needs to provide, at minimum, one other event: Update Complete, indicating that all processing that needs the current local synaptic buffer has completed, and the buffer can be thus invalidated and possibly be written back, freeing it for use by another arriving packet. This event uses the ARM’s software interrupt (SWI) mechanism. To ensure best performance for packet servicing, we add an additional software queue for incoming packets, and a pair of additional events (also SWI’s). This software functionality corresponds to a second, higher level: System Level, which provides a series of services including the software implementation of the synapse channel.

D. System-Level View

The architectural model for the SpiNNaker system layer is that of a scheduler and a series of services (fig. 5). Its core component is the event handler, which acts as an efficient scheduler of self-contained processes. Interrupt handlers (including SWI’s) trigger the scheduler, which schedules an appropriate service that then executes. The scheduler is pre-emptive, allowing high-priority events such as packet received to receive immediate servicing.

Libraries implement individual services. Thus, a given model can load only the services it needs. Services themselves are independent modules that simply terminate when complete rather than returning to a “background” process. Within the synaptic channel, there are 3 relevant services:

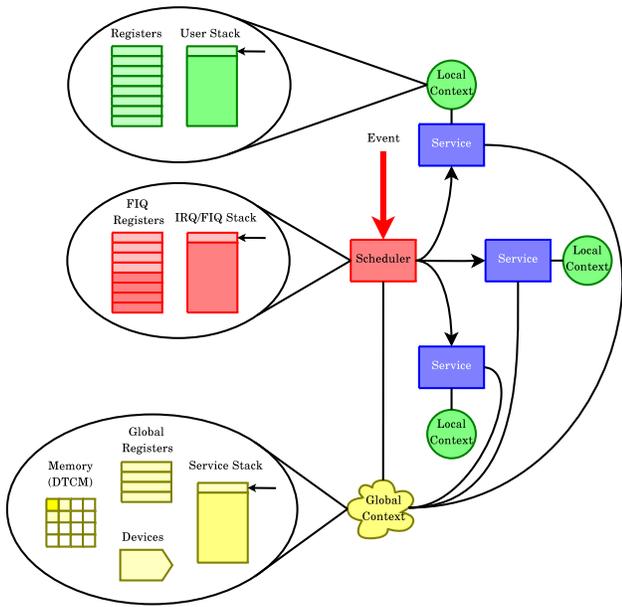


Fig. 5. SpiNNaker event-driven software model. Events are asynchronous signals that trigger the scheduler from the interrupt service routine. It then in turn triggers various services, which may run concurrently. Services start in ARM System (SYS) mode but may drop into user (USR) mode after completing critical entry point tasks. Each service has its own local context: the register file and user stack; that the scheduler must preserve. The scheduler does not need to preserve the global context between service calls; it can (asynchronously) update global context, consisting of main memory (DTCM), device registers, and any registers marked as global. (The number of global registers should be kept to an absolute minimum.) Updates to the global context are the primary method of inter-process communication. The ISR has its own private context. The ARM968 has 3 interrupt modes, FIQ, IRQ and SWI; in addition to separate stacks for each, the FIQ mode has its own private registers. Thus no interrupt need preserve scheduler context.

RequestDMA, StartDMA, and UpdateWeights. The first two of these are system-level services we introduce to process the incoming packet queue and compute DMA requests. RequestDMA services the packet queue. When an incoming spike arrives, it simply places the neuron ID and any payload into a queue, then exits to the scheduler with the RequestDMA service. This process then dequeues a packet and computes the necessary parameters (in particular, the location in SDRAM of the synaptic data) for the DMA. It then triggers an event, which in turn reaches the scheduler again with the StartDMA service. StartDMA issues the DMA transaction, then terminates silently. The UpdateComplete event has two variants. The first variant, active if synapses have plasticity which require writeback to SDRAM, triggers a StartDMA service. The second, UpdateBypass, retains the event-driven model by providing a software event *in lieu* of the DMA interrupt that would complete the weight update. Thus either when the DMA completes, or when UpdateBypass triggers its SWI, the system can invalidate the current buffer and move to the next. Figure 6 captures the behaviour of the entire event-driven synapse channel software layer.

The critical benefits of a fully event-driven model for the software side of the synapse channel are: 1) It allows both the system and higher-level model code to handle updates

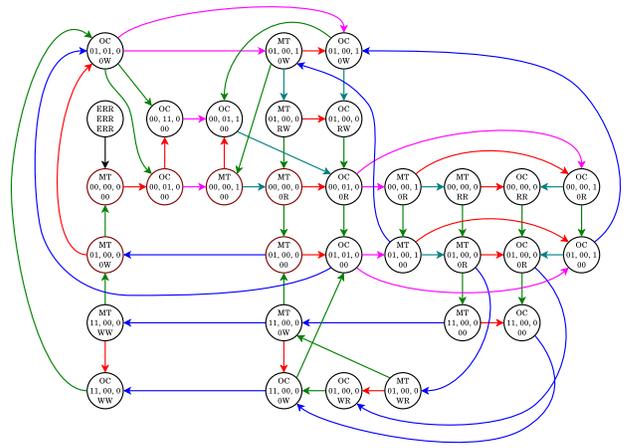


Fig. 6. Representation of the states and possible event-driven transitions within the SpiNNaker synapse channel software system. States (circles) have 3 values indicating the condition of the packet queue (empty or occupied), the condition of the local data buffers, and the number and type of DMA transactions currently in flight. Arrows indicate events, which trigger state transitions: red = packet arrived; magenta = DMA request; green = DMA complete; blue = update complete.

regardless of the order and timing of events; 2) It permits efficient use of DMA hardware and buffering while keeping track of what operations are in progress; 3) it permits a reentrant, prioritisable model of process scheduling, allowing high-priority events like incoming packets to pre-empt long, low-priority tasks like synaptic update; 4) it permits an absolute minimum of time to be spent in critical sections during interrupt processing, so that the system can quickly reenact event handling when an event occurs. These benefits are essential to permit real-time simulation with neurons having high fan-ins and reasonable spike rates, as we show in the next section.

V. SIMULATION OF SPIKING MODELS ON SPINNAKER

A. Resilience to Input Bursting

To test the new model under a variety of high-stress conditions we created two scenarios. In the first scenario we tested the response of the system in the presence of intense traffic with bursting behaviour. We implemented a synfire chain model [30] as follows. We connected 50 pools of 10 neurons in a feed-forward one-to-one fashion, each spike arriving from a presynaptic neuron being strong enough to elicit a spike in the corresponding postsynaptic neuron (weight=30). Neurons are modelled as LIF with the following parameters: $\tau_m = 16msec$, $V_{rest} = -65mV$, $V_{reset} = -75mV$, $V_{thresh} = -55mV$. We simulated the first population with a current $I = 1$. Results are in fig. 7 where it can be seen that the volley of spikes are propagated to each pool in a feed-forward fashion. The rate plot shown in fig. 8.

To stress test the packet-handling system we adapted the synfire chain model in the following way: we connected 3 pools of 300 neurons each with fixed delays, and stimulated the first population of neurons so to make them all fire together. Fig. 9 shows the results of this test. The activity

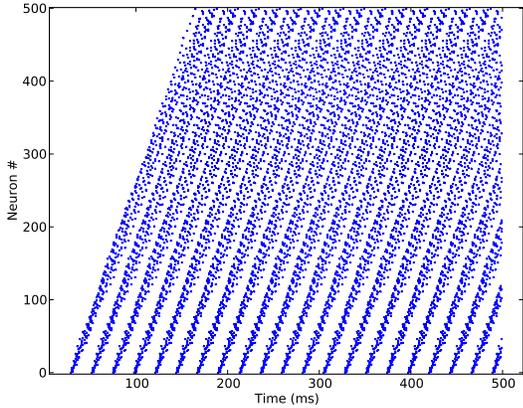


Fig. 7. Synfire chain with 500 neurons

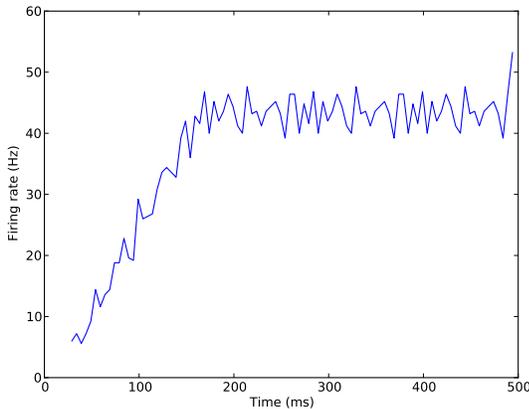


Fig. 8. Synfire chain: mean population rate rises up to a mean of 45Hz

propagates regularly across the 3 pools, each neuron firing in the same millisecond generating 300 synchronous packets. The ability of the system to handle this (much-higher-than-normal) input rate demonstrates the robustness of the event-driven mechanism.

B. Buffer Failure

During testing we noticed certain potential failure modes, which emphasize the need for careful design and testing of event-driven systems, and demonstrates the effects of asynchronous behaviour. We connected 3 input LIF neurons (with the same parameters as the previous test) to 15 output neurons in an all-to-all fashion. The 15 output neurons connect back to the input neurons providing mild inhibition (weight = -1). We then stimulated the 3 input neurons so as to make them spike with a mean frequency of 40 Hz. In some tests, we saw the expected behaviour (fig. 11). Others, however, gave the output in fig. 10. An analysis revealed that the triggering factor was the presence or absence of debugging messages; a common symptom in event-driven design. This pathology does not affect the “release” code base but it does provide an illuminating example of the need

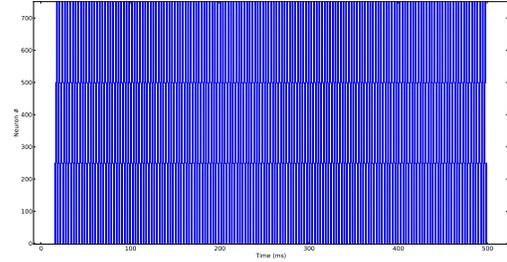


Fig. 9. Simulation of a highly synchronous synfire chain model. The system simulates 3 populations of 300 neurons each. All neurons fire in precise synchrony - a worst-case test for the packet-handling system

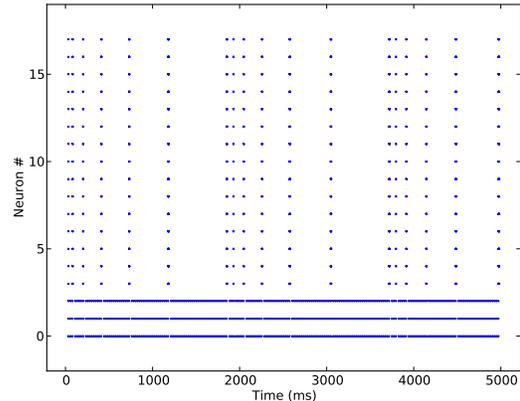


Fig. 10. One-to-all: failure of the network

for system libraries: a SpiNNaker user could not be expected to identify, much less debug, such behaviour if they had to write their own system functions.

VI. A GENERAL EVENT-DRIVEN FRAMEWORK FOR THE SPINNAKER SYSTEM

In principle, SpiNNaker can implement virtually *any* network, but, by virtue of being a hardware device, it is naturally subject to definite limitations which place bounds on its universality. In a previous work ([24]), we reflected, “from the models that have successfully run it is clear that SpiNNaker can support multiple, very different neural networks; how general this capability is remains an important question.” The experiments we have performed using the event-driven synapse channel software model put us in a position to start to examine this question.

The central theme of the answer is this: SpiNNaker does have limitations, but not in the fixed sense typical of most neuromorphic devices, such as having a definite maximum number of neurons X or number of connections Y . Rather, the chip offers a variety of design tradeoffs to the neural modeller, depending on the complexity of the model, the expected rate of operation (real-time; accelerated-time; retarded time), and the dynamics of activity. As it is, for most models, memory capacity rather than processing overhead is

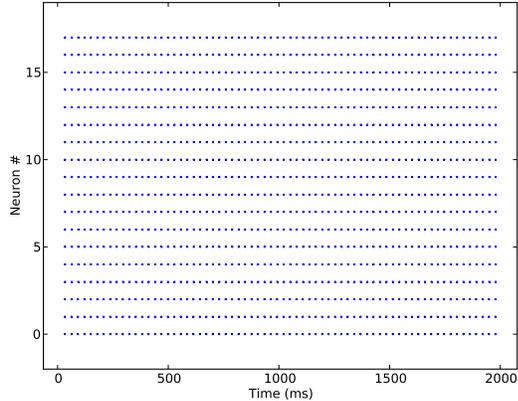


Fig. 11. One-to-all: Expected behaviour

the limiting factor: a limit of 1724 AMPA/GABA-A-only neurons/processor. However, this assumes uniform traffic patterns. With very bursty traffic, the experiments revealed that in fact the size of the synaptic row, i.e. the mean connectivity per axon, tends to become a limiting factor before the system exhausts the packet-handling capacity of the network. This is consistent with the model, in that synaptic updating dominates the time to process the complete function pipeline. This creates a tradeoff between number of neurons, number of connections per neuron, and peak activity rate. Careful pre-instantiation analysis is therefore important for networks with the potential to generate large, synchronous resonances, and users of such networks will probably need to follow an incremental approach, testing first on small networks before building large and complex networks. However, the simple fact that the user is presented with tradeoffs rather than hard model limits is new for hardware, and is one of the distinguishing features of the “neuromimetic” architecture as opposed to a traditional “neuromorphic” architecture.

The synapse channel is only one part of the complete SpiNNaker software environment, and is the first component we have transformed from early prototype software using conventional sequential programming into a fully event-driven system. We are working on rebuilding the entire software “stack” into an event-driven architecture, with standard interfaces that handle the low-level implementation details transparently to the model and permit a much more general system: a system framework for neural models that abstract as much of SpiNNaker system as is not integral to the neural model itself.

Fig. 12 shows the basic architecture of the event-driven framework. The neural model developer writes callback routines associated with the events that are of interest, *e.g.* arrival of a spike (packet), completion of a memory transfer or periodic time interval. These callbacks are registered with the system kernel to be executed when the corresponding event takes place. The kernel is responsible for the initial handling of events, servicing hardware devices, and scheduling the

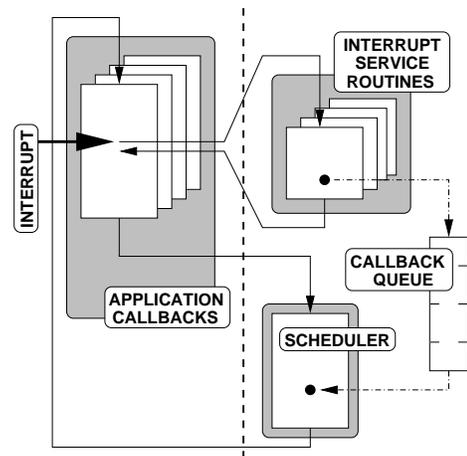


Fig. 12. An event-driven framework. The left side is the model space, consisting of callback routines. The right side is the system space, consisting of a scheduler and interrupt-service routines.

corresponding callback for execution. The callback routine, executing entirely in model space, transfers control back to the kernel on completion allowing the scheduler to transfer control to the next pending callback.

The flexibility of the SpiNNaker architecture depends in part upon node homogeneity and an absence of dedicated, fixed-structure control hardware but this presents challenges when loading heterogeneous neural models and structures into disparate chips. A distinct “configuration” phase, necessarily preceding model execution, uses a packet-based, event-driven approach [31]. In this phase scalability is a major concern. Our current approach: loading each chip in turn with individual binaries, would incur unacceptable overheads in machines of only hundreds of chips. Furthermore, it implies that the human and computational resources are available to hand-craft billion neuron models “offline”. We are working on a method to distribute a high-level description of model functions and data as a generic binary, using a flood-fill mechanism which loads code concurrently onto chips. Each packet in the flood-fill will generate an event in receiving chips triggering local storage and forward retransmission on all forward links. Receipt of the final packet will trigger chip ID-dependent unpacking of the descriptions into neuron and synapse data structures and computation of the corresponding routing tables. Even more advanced research is underway to identify appropriate load and model execution architectures for very-large-scale systems of tens of thousands of chips.

VII. CONCLUSIONS

We have demonstrated a fully event-driven implementation of the software component of the SpiNNaker synapse channel, matching the hardware’s event-driven model and insulation of model synapses from the low-level details of data location and access. This implementation is the first component of a systematic programme to create a standard, library-based, event-driven system layer for SpiNNaker development. In light of the complexities of event-driven design

and the different model of computation neural networks represent, such a library can be considered almost mandatory for widespread hardware adoption. Steinkraus ([5]) is gloomy on the prospects for dedicated hardware: “Using dedicated hardware to do machine learning most often ends up in disaster. The hardware is typically expensive, unreliable, without libraries, poorly documented, and obsolete within a few years”. With the SpiNNaker library we hope to start to change this, at least for one chip. However just as importantly, we are developing a reference *methodology* for event-driven neural hardware development. This is a need just as crucial. Event-driven software design is hard. The literature is thin and scattered. Practitioners are a small community, often relying on empirical knowledge that has become part of the “unwritten folklore” of the field. Into this world the neural modeller, interested in running models rather than relearning programming, enters to face frustration. With the SpiNNaker model we hope to de-mystify development through a standard interface. If future neural hardware will be event-driven, it makes sense that its software environment should also be event-driven.

REFERENCES

- [1] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. del Giudice, R. J. Douglas, and G. Indiveri, “A Multichip Pulse-Based Neuromorphic Infrastructure and Its Application to a Model of Orientation Sensitivity,” *IEEE Trans. Circuits and Systems*, vol. 54, no. 5, pp. 981–993, May 2007.
- [2] J. Fierres, J. Schemmel, and K. Meier, “Realizing biological spiking network models in a configurable wafer-scale hardware system,” in *Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008)*. IEEE Press, 2008, pp. 969–976.
- [3] D. Brüderle, E. Müller, A. Davison, E. Müller, J. Schemmel, and K. Meier, “Establishing a novel modeling tool: a python-based interface for a neuromorphic hardware system,” *Frontiers in Neuroinformatics*, vol. 3, no. 17, Jun. 2009.
- [4] D. Brüderle, A. Grübl, K. Meier, E. Müller, and J. Schemmel, “A Software Framework for Tuning the Dynamics of Neuromorphic Silicon Towards Biology,” in *Proc. 2007 Int’l Wkshp on Artificial Neural Networks (IWANN 2007)*, 2007, pp. 479–486.
- [5] D. Steinkraus, I. Buck, and P. Y. Simard, “Using GPUs for machine learning algorithms,” in *Proc. 8th Int’l Conf. Document Analysis and Recognition*, 2005, pp. 1115–1120.
- [6] P. Inne, T. Cornu, and G. Kuhn, “Special-Purpose Digital Hardware for Neural Networks: An Architectural Survey,” *J. VLSI Signal Processing Systems*, vol. 13, no. 1, pp. 5–25, Aug. 1996.
- [7] L. R. Kern, “Design and development of a real-time neural processor using the intel 80170nx etann,” in *Proc. 1992 Int’l Joint Conf. Neural Networks (IJCNN1992)*, 1992, pp. 684–689.
- [8] G. L. Camera, W. Senn, and S. Fusi, “Comparison between networks of conductance- and current-driven neurons: stationary spike rates and subthreshold depolarization,” *Neurocomputing*, vol. 58–60, pp. 253–258, Jun. 2004.
- [9] M. Rudolph and A. Destexhe, “How much can we trust neural simulation strategies?” *Neurocomputing*, vol. 70, no. 10–12, pp. 1966–1969, Jun. 2007.
- [10] M. Migliore, C. Cannia, W. W. Lytton, H. Markram, and M. L. Hines, “Parallel network simulations with NEURON,” *J. Computational Neuroscience*, vol. 21, no. 2, pp. 119–29, Oct. 2006.
- [11] M. Mattia and P. D. Giudice, “Efficient Event-Driven Simulation of Large Networks of Spiking Neurons and Dynamical Synapses,” *Neural Computation*, vol. 12, no. 10, pp. 2305–2329, Oct. 2000.
- [12] A. Delorme and S. Thorpe, “SpikeNET: an event-driven simulation package for spiking neural networks,” *Network: Computation in Neural Systems*, vol. 14, no. 4, pp. 613–627, Nov. 2003.
- [13] E. Ros, E. M. Ortigosa, R. Agís, R. Carrillo, and M. Arnold, “Real-Time Computing Platform for Spiking Neurons (RT-Spike),” *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 1050–1063, Jul. 2006.
- [14] M. James and D. Hoang, “Design of Low-Cost, Real-Time Simulation Systems for Large Neural Networks,” *J. Parallel and Distributed Computing*, vol. 14, no. 3, pp. 221–235, Mar. 1992.
- [15] F. Bernhard and R. Keriven, “Spiking Neurons on GPUs,” in *Proc. 6th Int’l Conf. Computational Science (ICCS 2006)*, 2006, pp. 236–243.
- [16] M. Gshwind, V. Salapura, and O. Maischberger, “RAN²OM A Reconfigurable Neural Network Architecture Based on Bit Stream Arithmetic,” in *Proc. 1994 Int’l Conf. Field Programmable Logic and Applications (FPL 1994)*, 1994, pp. 1861–1864.
- [17] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid, “Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks,” in *Proc. 2008 Int’l Conf. Field Programmable Logic and Applications (FPL 2008)*, 2008, pp. 483–486.
- [18] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Silvotti, and D. Gillepsie, “Silicon Auditory Processors as Computer Peripherals,” *IEEE Trans. Neural Networks*, vol. 4, no. 3, pp. 523–528, May 1993.
- [19] K. A. Boahen, “Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events,” *IEEE Trans. Circuits and Systems 2: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, May 2000.
- [20] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, “Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance-Based Synapses,” *IEEE Trans. Neural Networks*, vol. 18, no. 1, pp. 253–265, Jan. 2007.
- [21] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, “Fully Digital AER Convolution Chip for Vision Processing,” in *Proc. 2008 IEEE Int’l Symp. Circuits and Systems (ISCAS2008)*, 2008, pp. 652–655.
- [22] J. Tomas, Y. Bornat, S. Saïgli, T. Lévi, and S. Renaud, “Design of a modular and mixed neuromimetic ASIC,” in *Proc. 13th IEEE Int’l Conf. Electronics, Circuits, and Systems (ICECS2006)*, 2006, pp. 946–949.
- [23] S. Renaud, J. Tomas, Y. Bornat, A. Daouzli, and S. Saïgli, “Neuromimetic ICs With Analog Cores: An Alternative for Simulating Spiking Neural Networks,” in *Proc. IEEE Int’l Symp. Circuits and Systems (ISCAS2007)*, 2007, pp. 3355–3358.
- [24] A. D. Rast, X. Jin, F. Galluppi, L. A. Plana, C. Patterson, and S. B. Furber, “Scalable Event-Driven Native Parallel Processing: The SpiNNaker Neuromimetic System,” in *Proc. 2010 ACM Conf. Computing Frontiers (CF’10)*, 2010, pp. 20–29.
- [25] L. A. Plana, S. B. Furber, S. Temple, M. M. Khan, Y. Shi, J. Wu, and S. Yang, “A GALS Infrastructure for a Massively Parallel Multiprocessor,” *IEEE Design & Test of Computers*, vol. 24, no. 5, pp. 454–463, Sep.-Oct. 2007.
- [26] A. Rast, S. Yang, M. M. Khan, and S. Furber, “Virtual Synaptic Interconnect Using an Asynchronous Network-on-Chip,” in *Proc. 2008 Int’l Joint Conf. Neural Networks (IJCNN2008)*, 2008, pp. 2727–2734.
- [27] A. Rast, X. Jin, M. Khan, and S. Furber, “The deferred-event model for hardware-oriented spiking neural networks,” in *Proc. 2008 Int’l Conf. Neural Information Processing (ICONIP 2008)*. Springer-Verlag, 2009, pp. 1057–1064.
- [28] X. Jin, A. D. Rast, F. Galluppi, S. Davies, and S. B. Furber, “Implementing Spike-Timing-Dependent Plasticity on SpiNNaker Neuromorphic Hardware,” in *Proc. 2010 Int’l Joint Conf. on Neural Networks (IJCNN2010)*, 2010, pp. 2302–2309.
- [29] A. D. Rast, F. Galluppi, X. Jin, and S. Furber, “The Leaky Integrate-and-Fire Neuron: A Platform for Synaptic Model Exploration on the SpiNNaker Chip,” in *Proc. 2010 Int’l Joint Conf. Neural Networks (IJCNN2010)*, 2010, pp. 3959–3966.
- [30] Y. Aviel, C. Mehring, M. Abeles, and D. Horn, “On Embedding Synfire Chains in a Balanced Network,” *Neural Computation*, vol. 15, no. 6, Jun. 2003.
- [31] M. M. Khan, A. D. Rast, J. Navaridas, X. Jin, L. Plana, M. Luján, S. Temple, C. Patterson, D. Richards, J. V. Woods, J. Miguel-Alonso, and S. B. Furber, “Event-Driven Configuration of a Neural Network CMP System over an Homogeneous Interconnect Fabric,” *Parallel Computing*, vol. 36, Nov. 2010.