

The Leaky Integrate-and-Fire Neuron: A Platform for Synaptic Model Exploration on the SpiNNaker Chip

A. D. Rast, F. Galluppi, X. Jin and S.B. Furber

Abstract—Large-scale neural hardware systems are trending increasingly towards the “neuromimetic” architecture: a general-purpose platform that specialises the hardware for neural networks but allows flexibility in model choice. Since the model is not hard-wired into the chip, exploration of different neural and synaptic models is not merely possible but provides a rich field for research: the possibility to use the hardware to establish useful abstractions of biological neural dynamics that could lead to a functional model of neural computation. Two areas of neural modelling stand out as central: 1) What level of detail in the neurodynamic model is necessary to achieve biologically realistic behaviour? 2) What is role and effect of different types of synapses in the computation? Using a universal event-driven neural chip, SpiNNaker, we develop a simple model, the Leaky-Integrate-and-Fire neuron, as a tool for exploring the second of these questions, complementary to the existing Izhikevich model which allows exploration of the first of these questions. The LIF model permits the development of multiple synaptic models including fast AMPA/GABA-A synapses with or without STDP learning, and slow NMDA synapses, spanning a range of different dynamic time constants. Its simple dynamics make it possible to expand the complexity of synaptic response, while the general-purpose design of SpiNNaker makes it possible if necessary to increase the neurodynamic accuracy with Izhikevich (or even Hodgkin-Huxley) neurons with some tradeoff in model size. Furthermore, the LIF model is a universally-accepted “standard” neural model that provides a good basis for comparisons with software simulations and introduces the minimal risk of obscuring important synaptic effects due to unusual neurodynamics. The simple models run thus far demonstrate the viability of both the LIF model and of various possible synaptic models on SpiNNaker and illustrate how it can be used as a platform for model exploration. Such an architecture provides a scalable system for high-performance large-scale neural modelling with complete freedom in model choice.

I. UNIVERSAL NEURAL HARDWARE: THE NEED FOR MODEL LIBRARIES

What is the purpose of dedicated neural network hardware? In the past, the answer was easy: model acceleration. However, rapid progress in standard digital hardware often outpaced the development cycle for first-generation neural chips, so that by the time they were available software simulators could outperform them anyway [1]. FPGA implementations offered hardware-like performance gains with greatly reduced design time [2], and added a new dimension: the ability to implement *different* models on the same platform and explore various architectural tradeoffs [3]. However, power and routability limitations with FPGAs effectively restricted their role to model prototyping [4]. Today, however,

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

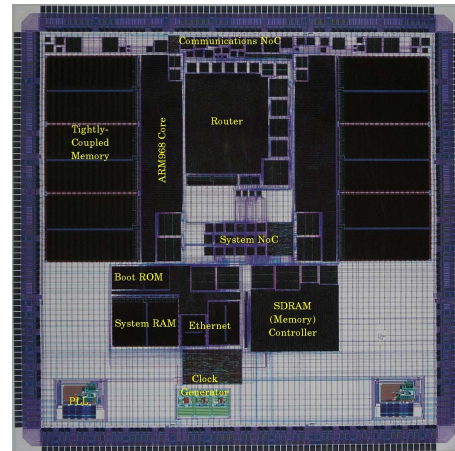


Fig. 1. SpiNNaker test chip

the rate of progress in standard digital hardware is flattening out, and the configurable-model concept that the FPGA introduced has given rise to new neural devices that offer various levels of model configurability without the scalability problems of the FPGA: “neuromimetic” chips [5]. These chips alter the nature of the research question: now it is not simply a matter of *how* to scale a neural model to large sizes but *what* model to scale.

The question of which neural network models are the “right” ones depends rather critically on what the purpose of the model is. A “one-size-fits-all” hardware neural model is thus unrealistic and somewhat irrelevant. Different research groups, with different goals, may want to use the hardware in different ways, and it is unrealistic to expect them to have, or even to acquire, the low-level hardware familiarity necessary to develop efficient models for universal neuromimetic chips. Configuring such chips can involve careful optimisation: the mapping of model to chip may be unusual and require intimate hardware knowledge [6]. Rather, the hardware developers should provide a library of different reference models, giving experimenters at minimum some models to get started with, and more generally a reference template that third-party developers could use to develop still further models and expand the hardware’s model support. The beginnings of a model library are therefore a crucial step in the adoption of neuromimetic hardware for neural simulation.

With the SpiNNaker chip (fig. 1), we have introduced a neuromimetic platform for large-scale neural modelling ideal for exploring the effects of different models. Previously,

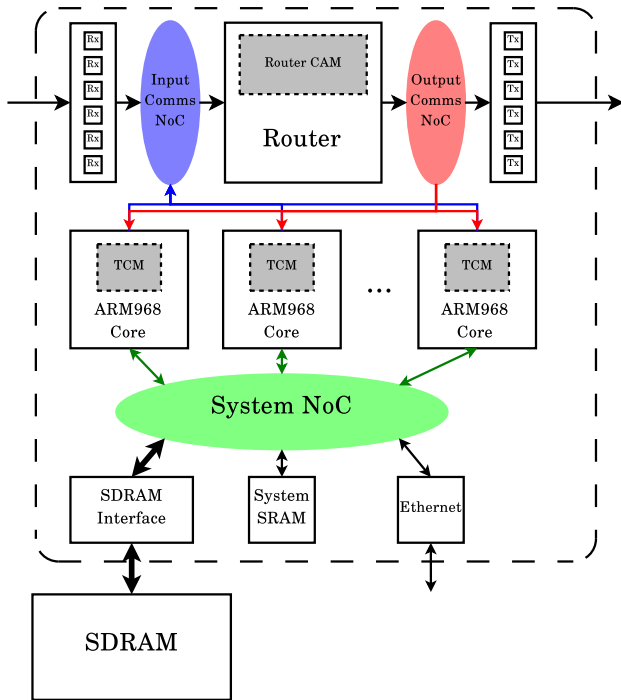


Fig. 2. SpiNNaker Architecture. The dashed box indicates the extent of the SpiNNaker chip. Dotted grey boxes indicate local memory areas.

we have introduced basic models for neural [7] and synaptic [8], [9] dynamics that illustrate the basic functionality of the device. However, these models are *illustrative* rather than *prescriptive*; having developed them we now wish to introduce further models to build a neural library and provide further examples to third-party developers of how to configure SpiNNaker. We have focussed on an important, popular neural model - the leaky-integrate-and-fire (LIF) model, and used it as a platform to develop other synaptic models, notably an NMDA synapse with slow voltage-gated dynamics. These models introduce a variety of techniques not yet demonstrated in the original reference models, while using and extending important core techniques from those models that show their general nature. Developing different neural models is a useful way not only to extend hardware capabilities, but to establish principles of efficient computation - a basis for approaching the question of what abstractions of neural function are useful.

II. THE SPINNAKER ASYNCHRONOUS EVENT-DRIVEN ARCHITECTURE

The SpiNNaker chip (fig. 2) is a universal neuromimetic platform containing programmable processing blocks embedded in a configurable asynchronous interconnect. Processors and interconnect are generic and configurable, but have a structure and function optimised for neural computation. SpiNNaker therefore shares features both with completely general-purpose FPGA’s and dedicated fixed-model devices while being distinctly different from either. The primary features of the architecture are:

Native Parallelism:

There are multiple (2 in the present implementation, 20 in a forthcoming version) general-purpose ARM968 processors per device, each operating completely independently from each other (on separate clocks). Each processor has its own private subsystem containing various devices to support neural functionality: a communications controller that handles I/O traffic in the form of AER packets, a DMA controller that provides fast virtual access to synaptic data residing off-chip in a separate (SDRAM) memory, and a Timer to generate fixed time steps where models need them. This “processing node” uses only local information to control execution and operates asynchronously from other processing nodes.

Event-Driven Processing:

SpiNNaker’s communications fabric is a configurable packet-switched asynchronous interconnect using Address-Event Representation (AER) to transmit neural signals between processors. AER is an emerging neural communication standard [10] that abstracts spikes from neurobiology into a single atomic event, transmitting only the address of the neuron that fired. SpiNNaker extends the basic AER standard with an optional 32-bit payload. This fabric implements the support infrastructure for the event-driven model.

Incoherent Memory:

Any processor may modify any memory location it can access without notifying or synchronising with other processors. SpiNNaker processors have access to 2 primary memory resources: their own local “Tightly-Coupled Memory” (TCM) and a global SDRAM device, neither of which require or have support for coherence mechanisms. The TCM is only accessible to its own processor and contains both the executing code (in the 32KB “Instruction TCM” (ITCM)) and local variables (in the 64KB “Data TCM” (DTCM)). The global SDRAM contains the synaptic data (and possibly other large data structures). The processor’s 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 [11].

Incremental Reconfiguration:

The structural configuration of the hardware can change dynamically while the system is running. SpiNNaker uses a distributed routing subsystem to direct packets through the Comms NoC. Each chip has packet-switching router that can be re-programmed in part or in full by changing the routing table, thus making it possible to reconfigure the model topology on the fly. Under the AER format, the router uses a source routing protocol to

direct traffic. Previous work ([12], [13]) describes the design of and configuration procedure for the Comms NoC.

III. LIF MODEL IMPLEMENTATION

A. Implementation Constraints

Implementation of the LIF model must consider SpiNNaker’s hardware architecture. In particular, the following details of the hardware act as model design constraints:

Elementary mathematical operations only

The ARM968 has basic add and subtract, logical, shift, and multiply operations, but does not have native support for division, transcendental functions, and other nonlinear computations. Therefore, the model must express its processing in terms of simple polynomial mathematical operations.

32-bit fixed-point representation

Similarly, the ARM968 has no floating-point unit. Models therefore cannot use floating-point notation, at least not without significant loss of efficiency. The model needs to translate any floating-point quantities into fixed-point numbers, while determining a position for the decimal point, hence assigning a fractional precision.

Limited local memory

With 64k data memory and 32k instruction memory per processor, SpiNNaker’s individual processors must operate within a limited memory space. If the processor is modelling a large number of neurons, this means perforce a minimal number of parameters, and arrays can become costly very quickly. This effectively prohibits having synaptic data local at all times. Memory management must therefore attempt to store as much information as possible on a per-neuron rather than per-synapse basis.

Limited time to process a neuron

To stay within the real-time update requirement, each neuron must be able to update its state before any external inputs could arrive that depend upon the updated state, and before any outputs consequent upon the updated state would occur. If a processor is modelling multiple neurons, in particular, this means updating at worst in $\frac{1}{N}R_{max}$, where N is the number of neurons modelled and R_{max} is the maximum event rate.

Synaptic data only available on input event

Because of the limited memory, SpiNNaker stores synaptic data off-chip and brings it to the local processor only when an input event arrives. Processes that depend on the synaptic value, therefore, can only occur in the brief window to process the new event. This means that synapse processing must be scheduled for a fixed time after the input, and can only depend on information knowable at the time the input arrived.

These constraints allow us to form a general specification for SpiNNaker neural models using a set of design guidelines in combination with an abstract processing model.

B. Implementation Rules

To meet SpiNNaker’s hardware constraints with an efficient, accurate model we introduce a set of design rules that help to define the model implementation. These rules are indicative but not forcing, so that while models generally obey this pattern they can in specific details deviate from it.

Defer event processing with annotated delays

The deferred-event model [8] is a method to allow event reordering. Under this scheme we only perform minimal processing at the time of a given event, storing state information in such a way as to be available to a future event, so that processes can wait upon contingent future events. *Future* events thus trigger state update relevant to the *current* event.

Solve differential equations using the Euler method

The Euler method is the simplest general way to solve nonlinear differential equations. In it, the processor updates the equations using a small fixed time step. The method then simply uses the formula $X(t + 1) = X(t) + \frac{dx}{dt}(t + 1)$. The time step is programmable (nominally 1 ms in our models), so modellers can choose finer time steps for better precision or coarser ones for more relaxed timing margins (and potentially more complex models).

Represent most variables using 16-bit values

Various studies indicate that 16-bit precision is adequate for most neural models [14], [15]. Since the ARM contains efficient 16-bit operations it makes sense to conserve memory space and use 16-bit variables throughout. Intermediate values, however, may use 32 bits to avoid unnecessary precision loss.

Precompute constant parameters where possible

By an astute choice of representation, it is often possible to transform a set of parameters in a neural equation into a single parameter that can be precomputed. Furthermore, it is often possible to choose this representation in a way that further simplifies the computation remaining. For example, in the expression $x(t) = Ae^{kt}$, we can use the substitution $\log_a b = \frac{\log_c b}{\log_c a}$, choose 2 for c and arrive at $x(t) = A(2^{(\log_2 e)kt})$, which allows us to precompute a new constant $\lambda = k \log_2 e$ and determine x with simple shift operations.

Compute non-polynomial functions by lookup table

Lookup tables provide a simple, and in fact the *only* general way of computing an arbitrary function. The ARM takes at most $2N$ instructions to compute a LUT-based function with N variables. Memory utilisation is a concern; even a 16-bit lookup table requires 64K entries (the entire DTCM) and is therefore impractical. However, an 8-bit lookup

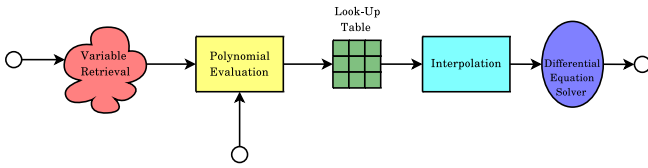


Fig. 3. A general event-driven function pipeline for neural networks. Variable retrieval recovers values stored from deferred-event processes as well as local values. Polynomial evaluation computes simple functions expressible as multiply and accumulate operations. These then can form the input to lookup 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).

table occupies only 256 entries (512 bytes if these are 16-bit values) and can access the value in a single ARM instruction. Where models need greater precision we implement various interpolations.

Exploit “free” operations such as shifting

Most ARM instructions can execute conditionally, many can shift an operand before doing the instruction, and there are built-in multiply-accumulate instructions. Taking advantage of such “free” operations is an obvious optimisation. An extreme example is the SMLAWx instruction. This performs a 32-bit * 16-bit multiply, truncates the low-order 16 bits of the 48-bit result, then accumulates it with a 32-bit value. If, then, the 16-bit value is a number representing a fraction with implied leading 1, by choosing it to represent the reciprocal of an integer it is possible to perform $w = \frac{x}{y} + z$ in a single instruction.

Using these rules, we can build up a generalised function pipeline to represent a neural process that is adequate for most models (fig. 3).

C. The LIF Model

The LIF model uses this instruction pipeline. Many of the techniques it uses are common to the “reference” Izhikevich model we have described earlier [7]. The basic approach applies to virtually any spiking model with voltage-variable differential-equation dynamics: an illustration of the universal design of the software as well as the hardware. We will walk through the function pipeline stage by stage.

1) *Variable Retrieval*: Like the Izhikevich neuron, the LIF neuron uses the deferred-event model to place input spikes into a circular array of current buffers representing the total input in a given time step (fig. 4). At the actual time of arrival of an input, the model does nothing more than trigger a DMA operation which retrieves a source neuron-indexed synaptic row. Once this row has been retrieved, a deferred event, DMA complete, triggers a second stage of deferral by finding the delay associated with a given synapse and placing the synaptic weight (representing the current injection) into the bin in the circular array representing the delay value. Finally, when that delay expires, it triggers another deferred

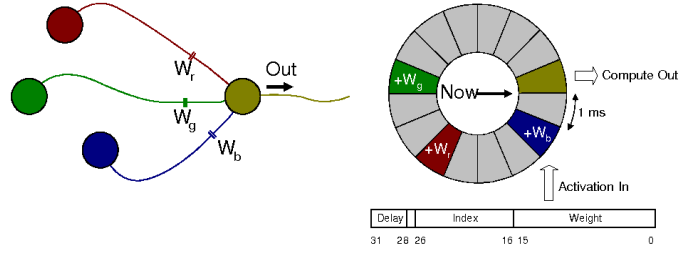


Fig. 4. SpiNNaker Neuron Binned Input Array. Inputs arrive into the bin corresponding to their respective delay. Output is computed from the bin pointed to by “Now”.

event, the Timer event, that recovers the total current from the bin. At the same time, the neuron retrieves its associated state block, containing the voltage variable and the parameters V_r (rest voltage), V_t (threshold voltage), V_s (reset voltage), R (membrane resistance), and $f_n (= \frac{1}{\tau})$ (natural frequency). Note that we precompute this last value from time constant τ in order to avoid division.

2) *Polynomial Evaluation*: The basic LIF neuron equation is [16]

$$\frac{dV}{dt} = f_n(V_r - V + IR)$$

The right-hand side is a simple polynomial, which we can easily compute (in 3 instructions using 2 SMLAxx multiply-accumulates). It is possible also to incorporate conductance-based synapses into this evaluation; the function then becomes

$$\frac{dV}{dt} = f_n(V_r - V + T(t)(V - V_n) + IR)$$

where $T(t)$ is the synaptic “transmissivity”, a value that incorporates the values of maximum synaptic conductance and specific membrane resistivity into the release probability by precomputing and storing the weights as this aggregate quantity. V_n is the synaptic rest voltage.

3) *Look-up table*: For the “normal” LIF neuron there is no need for a lookup table function since the differential equation is polynomial. However, if we include conductance-based NMDA synapses, there is a voltage gating term [16]:

$$G = \frac{1}{1 + \frac{Mg^+}{3.57} e^{(\frac{-V}{16.13})}}$$

where Mg^+ is the magnesium ion concentration. This function will use a (256-entry) LUT to compute its value.

4) *Interpolation*: Likewise in the standard LIF model there is no need for interpolation since the function can be computed exactly. If the model uses NMDA synapses, and if the model requires better than 8-bit precision in the gating factor, however, we have implemented a spline-based interpolation. While efficient, it is not computationally free, requiring 48 cycles.

5) *Differential Equation Solver*: The Euler-method process evaluates the differential equation at each Timer event. Timer events occur each millisecond. After evaluating the

equation, it also tests whether the potential has exceeded the threshold V_t . If it has, it resets V to V_s .

The LIF computation requires 10 instructions if the neuron does not fire, 21 if the neuron fires, and 38 if the neuron fires and synapses have spike-timing-dependent plasticity (STDP). The high efficiency of this model makes it an ideal test-bed for exploring different synaptic models.

IV. SYNAPTIC MODELS

Spiking neural networks can contain various different synapse types with different dynamics. At present the role of different synaptic types remains an area of intense research interest [17]. Equally significantly, the level of biophysical realism necessary to achieve useful behaviour or model actual brain dynamics is unclear. For instance, in the case of the well-known STDP plasticity rule, while many models exist describing the behaviour [18], [19], the actual biological data on STDP is noisy and of low accuracy. Observed STDP modifications exhibit a broad distribution for which the nominal functional form of STDP models usually constitute an envelope or upper bound to the modification [20], [21]. This suggests that high repeatability or precision in STDP models is not particularly important.

While SpiNNaker is capable of modelling synapses with biophysical realism down to the molecular level if necessary, such high biological fidelity is computationally expensive. In view of the observed synaptic variability exact biological replication, or fidelity to a precise functional form, appears to be unnecessary for understanding their computational properties. This gives considerable latitude for experimenting with different synaptic models in order to investigate various tradeoffs between computational cost and functional accuracy. Using the LIF model gives not only a default “reference” model with known and easily tuned dynamics that expose the synaptic dynamics clearly, it also provides a very-low-computational-cost model, minimising machine cycles to allow more time for complex synaptic models. We are using the LIF model to test 2 different models.

The first model has been the reference model for SpiNNaker development and is a temporally accurate STDP model. Details of the technique and algorithm are in [8], [9]. This model demonstrates 2 important techniques: use of the deferred-event model to reorder processes, and exploiting the ARM’s bit-shifting logic to maximise computational efficiency. In the deferred-event model, the processor performs the synaptic update only upon receipt of a new presynaptic spike, using a pair of time stamps to record presynaptic and postsynaptic events so that the processor can track postsynaptic spikes happening after a given input. It can then retroactively apply the weight changes that would have happened to the synapse between the last input and the current input, using shift-and-add operations, before transmitting the spike through the synapse.

The second model adds NMDA-mediated synapses exhibiting voltage gating with slow dynamics. This model presents 2 challenges: first, how to model the slow dynamics without having to retrieve the synaptic values multiple times,

and second, how to implement the voltage gating process. 2 properties of NMDA synapses make it possible to implement an efficient algorithm. First, the decay of the current exhibits a *linear* kinetic, therefore instead of using sampled Euler-method evaluation it is possible to precompute the update for each input. Second, voltage gating is multiplicative and depends on the *post-synaptic* potential, thus the net contribution from all synapses with NMDA-mediated dynamics can be computed by multiplying their aggregate activation by the gating factor on a per (postsynaptic) neuron basis.

Precomputing the synapse open probability uses a variety of techniques. NMDA synaptic currents reach their maximum value quickly, usually within 1.5 ms of spike arrival. This allows us to neglect the effects of rise-time dynamics and simply incorporate it into the delay. The current then decays exponentially with a slow time constant τ . We can accommodate this slower dynamic by adding a second set of activation bins to each neuron that rotate every τ ms. Precomputing the value to place in each successive bin after the initial (T_{max}) bin is trivial by exploiting the exponential property of shifting and using precomputed parameter scaling: shift the “weight” stored in the synaptic block, right by one place for each successive bin.

At each (1ms) time step, the neuron then multiplies the current NMDA bin (representing aggregate NMDA stimulus) by the voltage gating factor and adds it to that of the normal fast (AMPA/GABA-A) bin using the previously-described LUT-based scheme. Once again this function can exploit extensive precomputation to simplify the LUT evaluation. The neuron also needs to test whether the NMDA bin should rotate, based on the system time t_{sys} . A series of transformations allows us to compute $t_{sys}REM\tau$ efficiently in order to perform this test.

The complete process then functions as follows:

Presynaptic input event

- 1) Retrieve the synaptic row corresponding to the presynaptic neuron.
- 2) Calculate the effective bin delay (position of bin P_{max}) using $T_{max} = \delta_w(\frac{1}{\tau})$, where δ_w is the real-time annotated delay in the synaptic weight.
- 3) Precompute the synaptic contribution (effective open probability) for each future bin.
- 4) Distribute the contributions into each bin by accumulating them with the current bin value.

Neuron timer event

- 1) Get the current system time.
- 2) Calculate whether the NMDA bin should rotate by computing $t_{sys}REM\tau \leq \frac{1}{2\tau}$.
- 3) If the bin should rotate, advance it and clear the previous bin.
- 4) Compute the gating factor by LUT.
- 5) Multiply the current NMDA bin by the gating factor.
- 6) Add to the (fast AMPA/GABA-A) activation.

The NMDA synaptic model takes advantage of the low instruction count of the LIF process. Updating the NMDA dynamics in a neuron requires 22 instructions. Thus the total

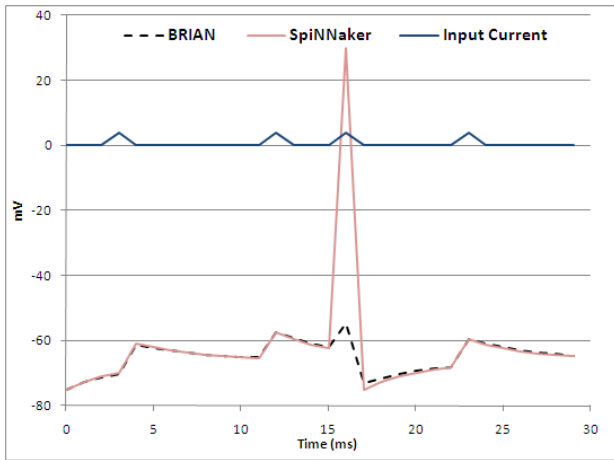


Fig. 5. Single neuron dynamics. We inject 4 pulses of current into the neuron. Traces compare the membrane potential of the simulation run on SpiNNaker (continuous line) with the same neuron implemented in Brian (dashed line).

instruction count rises only to 32 if the neuron does not fire, 43 if the neuron fires. This remains well within timing limits to permit ~ 1000 neurons per processor.

V. SIMULATING VARIOUS SPIKING MODELS ON SPINNAKER

A. Single Neuron Dynamics

To test the LIF model, we started with a single neuron. We tested single neuron dynamics by injecting short pulses of current into a neuron with the following parameters: $V_0 = V_s = -75mV$, $V_r = -66mV$, $f_n = \frac{1}{4}ms^{-1}$, $R = 8$, $V_t = -56mV$. Here V_s is the reset voltage, V_r is the rest voltage, f_n is the natural frequency ($=\frac{1}{T}$), R is the membrane resistance, and V_t is the threshold voltage. To test the accuracy of our implementation we compared it with the same neuron implemented with Brian [22], driving it with the same input current. Results are in fig. 5: the difference in the spiking region is due to the fact that we artificially set $V = 30mV$ when a neuron crosses the threshold in order to have a self-evident spike.

B. Spike Propagation

In order to test the time precision of spike generation and propagation processes we implemented a network capable of detecting inter-pulse interval between spikes generated by two neurons, reproducing the results in [23] (Section 2). Two input neurons (transponders) connect to 7 output neurons (detectors) with a delay proportional to the distance (fig. 6).

The weights are set so that a detector will fire only if two coincident spikes arrive within a millisecond interval; detector neuron τ_i only fires when the inter-pulse interval $t_a - t_b = i$, where t_a and t_b are the absolute firing times of neurons a and b respectively (eg. neuron τ_{-3} fires when neuron b fires 3 msec *after* neuron a , neuron τ_{+2} fires when neuron b fires 2 msec *before* neuron a etc). Fig. 7 represents the network structure, specifying the delays. Fig. 8 presents the simulation results. It can be seen that the network is able

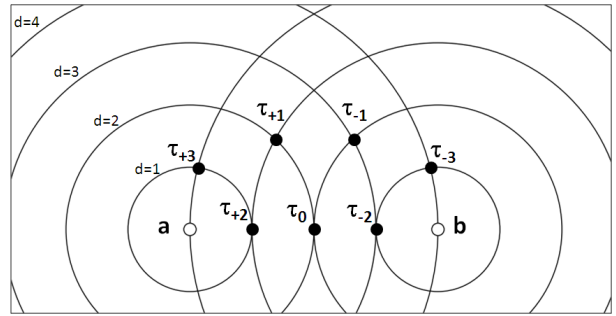


Fig. 6. Detection of the inter-pulse interval between spikes from neurons a and b . d denotes the synaptic delay between transponders and detectors. The detector neuron τ_i fires when the inter-pulse interval $t_a - t_b = i$.

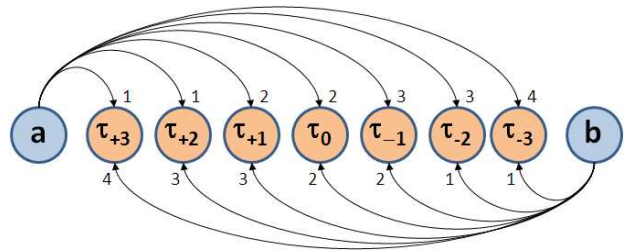


Fig. 7. Network Structure. Arrows represent connections between neurons. Values at the end of each arrow represent synaptic delays, set according to detector placement (cf. fig. 6).

to discriminate the inter-pulse interval between the firings of neuron a and neuron b with millisecond precision by producing a spike from the corresponding detector neuron. This result confirms the millisecond precision of the LIF module implementation.

C. Oscillatory Network Activity

We tested network dynamics by simulating the network in fig. 9: the network contains 100 neurons ($V_0 = V_s = -75mV$, $V_r = -65mV$, $f_n = \frac{1}{16}ms^{-1}$, $R = 8$, $V_t = -56mV$) divided into 80 excitatory neurons and 20 inhibitory neurons. Each neuron in the excitatory group connects to 56 (70%) excitatory neurons and 2 (10%) inhibitory neurons with random delay between 1 and 8 ms. Each inhibitory neuron connects to every excitatory neuron (full connection - 100%) with a delay of 1 or 2 ms. We selected 8 neurons from the excitatory group to be input neurons and injected them with a constant current of 3 nA, making them fire approximately every 10 ms. Excitatory weights are set in order to build up the background activity of the network slowly. Once there is sufficient activity, the whole excitatory group starts firing, causing the inhibitory neurons to fire a few ms later (due to the dense connectivity). Inhibitory weights are set to quench network activity quickly. Fig. 10 presents the results of the simulation.

VI. CONSIDERATIONS FOR EFFICIENT MODELLING LIBRARIES

Implementing the LIF model has made it possible to consider efficiency limits within the SpiNNaker hardware.

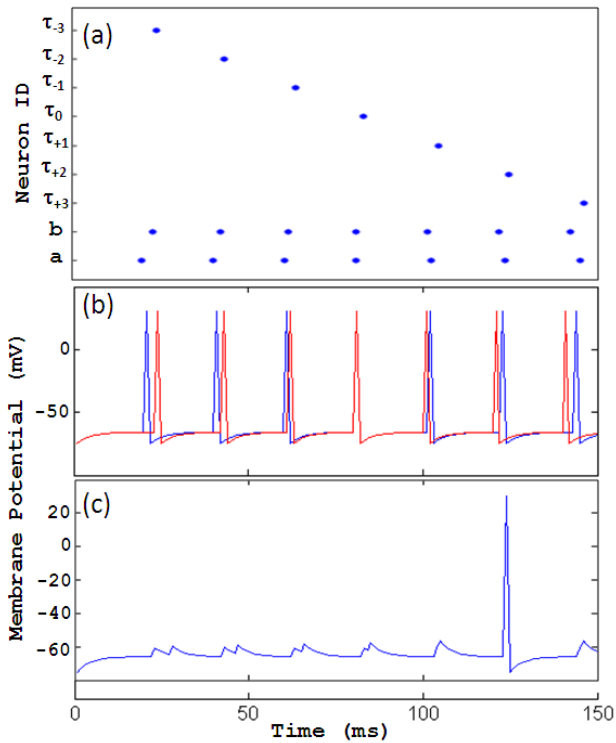


Fig. 8. Spike Propagation. (a) Raster Plot. Only the neuron detecting the correct interspike interval fires. (b) Membrane potential of neurons a (blue) and b (red). (c) Membrane potential of detector neuron τ_{+2} . The neuron only fires when spikes from neurons a and b converge on with coherent synaptic delays

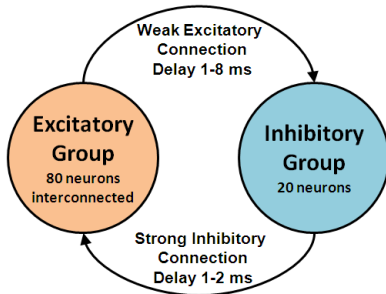


Fig. 9. Oscillatory Network Structure. Excitatory and inhibitory groups are connected so that when the activity of the excitatory group gets high the inhibitory group shuts the activity of the whole network.

The core computation time of 10 instructions without spiking, 21 with, is probably the minimum possible for any model with broad acceptance within the spiking neural model community. If we assume a mean activity rate throughout the network of 1% (number of active neurons in a given processor), and provide for a timing margin of 0.2 ms in a 1 ms Timer (Euler sample) interval, this enables us to examine the maximum number of neurons a single SpiNNaker processor could model.

For most models, memory capacity rather than processing overhead is the limiting factor: most configurations easily supported rates in excess of 2000 neurons/ms; considerably over the memory limit of 1724 AMPA/GABA-A-only

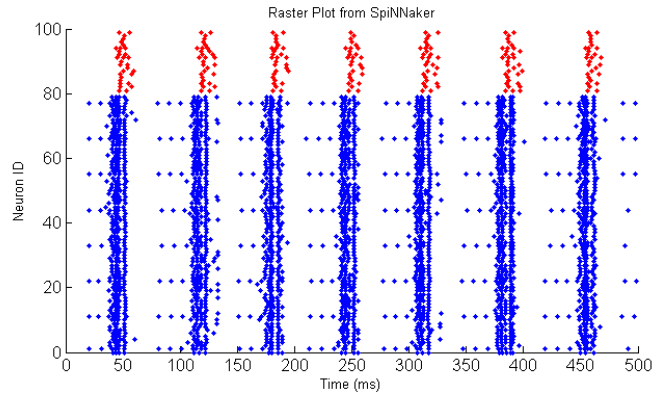


Fig. 10. Oscillatory Network Raster Plot. Input neurons (ID's 1, 11, 22, 33, 44, 55, 66, 77) feed excitatory neurons (ID's 0-79), slowly building up the activity until excitatory neurons start to fire with high frequencies. Inhibitory neurons (ID 80-99) then activate, quenching the activity of the network.

neurons/processor, or 910 neurons with NMDA synapses. Of the various factors the mean spiking rate of the active population has the greatest impact on performance. This was particularly noticeable for NMDA synapses where high firing rates increase the number of iterations necessary to precompute future NMDA activation; a multiplicative effect. High rates also increase the mean activation, making it more probable that the NMDA gate will turn on, increasing the downstream firing rate. The result is a processing cascade - and at the maximum firing rate of 100 Hz the number of neurons a processor can model drops drastically.

The LIF model can support more complex synaptic models that other spiking models such as the “reference” Izhikevich model or conductance-based models, simply because it has fewer instructions to perform. This in itself is a strong reason to make it the model of choice for large-scale studies of synaptic dynamics, but there is another and equally powerful motivation: ease of analysis.

Any research project must of necessity make a definite choice as to the scope of the research: *what* is being investigated. Unsurprisingly, the majority of investigations into new synaptic models have used the LIF neuron so as not to introduce too many experimental variables simultaneously. It is ideal because it is very simple, exhaustively analysed, and does not introduce complexities that might obscure the effects of a given synaptic model. Developing the LIF model for SpiNNaker adds it to the “neural library” so that modellers can insert the neuron as a “drop-in” component without having to spend any time developing or implementing the neurons themselves on SpiNNaker.

Implementing a function pipeline has given us a standard “template” for library components. It considers what functions SpiNNaker can implement efficiently. SpiNNaker’s processors can easily compute polynomial functions but it is usually easier to implement other types, e.g. exponentials as a look-up table with polynomial interpolation. This adequately covers the right-hand-side of differential equations. For very simple cases it may be possible to solve such

equations analytically, but for the general case, the Euler method evaluation we have used appears to be adequate. Creating a new component for the library is simply a matter of plugging in appropriate models for the pipeline stages, allowing for extensive software reuse because most of the support libraries, low-level utilities, and “housekeeping” code can be general across all models. Only the dynamics need change. The library therefore takes the form of a general infrastructure with model-specific core routines.

VII. CONCLUSIONS

The LIF model we have created is an important reference model for developing new synaptic models, and has been instrumental in creating and refining the function pipeline, as a base standard for neural and synaptic libraries. Developing more models for these libraries on SpiNNaker is one of our major research foci. We are currently reworking the library into a series of C++ template classes, where the template parameters can indicate the model type. This will further simplify future model development and provide a specification for third-party model building. The other major area of research is in increasing the scale of neural models. We are creating a large-scale neural model, based on the LIF neuron with AMPA/GABA-A and NMDA synapses, to simulate attentional control and response, with future development of short- and long-term memories. Such a system should form an effective, scalable demonstration network for a multichip SpiNNaker system.

In a larger context, the function pipeline model we developed may be a useful abstraction for neural hardware, regardless of platform. To create the function pipeline we attempted to decompose the general form of neurodynamic state equations into platform-neutral components that hardware can typically implement easily. Digital hardware can readily implement memories to form variable retrieval and LUT stages, and both analogue and digital hardware have effective blocks for polynomial evaluation and interpolation. Both DSPs and various analogue blocks offer efficient computation of differential equations. Thus one could build a neural system in building-block fashion, by chaining together various components using AER signalling, allowing for the construction of hybrid systems in addition to integrated approaches like SpiNNaker. More than anything else, SpiNNaker is valuable as a test bed for hardware architectures for neural models.

ACKNOWLEDGEMENTS

The Spinnaker project is supported by the Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Platform Partnership at the University of Manchester, and also by ARM and Silistix. Steve Furber holds a Royal Society-Wolfson Research Merit Award.

REFERENCES

[1] A.R.Omondi, “Neurocomputers: a dead end?” *Int’l J. Neural Systems*, vol. 10, no. 6, pp. 475–481, Dec. 2000.

[2] D.Roggen, S.Hofmann, Y. Thoma, and D. Floreano, “Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot,” in *Proc. 2003 NASA/DoD Conf. Evolvable Hardware*. IEEE Press, 2003, pp. 189–198.

[3] A.Upegui, C. Peña-Reyes, and E.Sanchez, “An FPGA platform for on-line topology exploration of spiking neural networks,” *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211–223, Jun. 2005.

[4] L. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, “Challenges for large-scale implementations of spiking neural networks on FPGAs,” *Neurocomputing*, vol. 71, Dec. 2007.

[5] S. Renaud, J. Tomas, Y. Bornat, A. Daouzli, and S. Saighi, “Neuromimetic ICs with Analog Cores: an Alternative for Spiking Neural Networks,” in *Proc. 2007 IEEE Int’l Symp. Circuits and Systems (ISCAS2007)*. IEEE Press, 2007.

[6] J.Fieres, 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.

[7] X. Jin, S. Furber, and J. Woods, “Efficient modelling of spiking neural networks on a scalable chip multiprocessor,” in *Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[8] 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)*, 2009.

[9] X. Jin, A. Rast, F. Galluppi, M. M. Khan, and S. Furber, “Implementing learning on the SpiNNaker universal neural chip multiprocessor,” in *Proc. 2009 Int’l Conf. Neural Information Processing (ICONIP 2009)*, 2009.

[10] D. Goldberg, G. Cauwenberghs, and A. Andreou, “Analog VLSI spiking neural network with address domain probabilistic synapses,” in *Proc. 2001 IEEE Int’l Symp. Circuits and Systems (ISCAS2001)*. IEEE Press, 2001, pp. 241–244.

[11] 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. on Neural Networks (IJCNN2008)*, 2008.

[12] M. M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, “SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor,” in *Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[13] A. Brown, D. Lester, L. Plana, S. Furber, and P. Wilson, “SpiNNaker: The design automation problem,” in *Proc. 2008 Int’l Conf. Neural Information Processing (ICONIP 2008)*. Springer-Verlag, 2009.

[14] T. Daud, T. Duong, M. Tran, H. Langenbacher, and A. Thakoor, “High resolution synaptic weights and hardware-in-the-loop learning,” in *Proc. SPIE*, vol. 2424, 1995, pp. 489–500.

[15] H.-P. Wang, E. Chicca, G. Indiveri, and T. J. Sejnowski, “Reliable computation in noisy backgrounds using real-time neuromorphic hardware,” in *Proc. 2007 IEEE Biomedical Circuits and Systems Conf. (BIOCAS2007)*, 2008, pp. 71–34.

[16] P. Dayan and L. Abbott, *Theoretical Neuroscience*. Cambridge: MIT Press, 2001.

[17] D. Durstewitz, “Implications of synaptic biophysics for recurrent network dynamics and active memory,” *Neural Networks*, vol. 22, no. 8, pp. 1189–1200, Oct. 2009.

[18] H. Z. Shouval, M. F. Bear, and L. N. Cooper, “A unified model of NMDA receptor-dependent bidirectional synaptic plasticity,” *Proc. Nat. Acad. Sci. USA*, vol. 99, no. 16, pp. 10 831–10 836, Aug. 2002.

[19] M. Hartley, N. Taylor, and J. Taylor, “Understanding spike-time-dependent plasticity: A biologically motivated computational model,” *Neurocomputing*, vol. 69, no. 16, pp. 2005–2016, Jul. 2006.

[20] H. Markram and M. Tsodyks, “Redistribution of synaptic efficacy between neocortical pyramidal neurons,” *Nature*, no. 382, pp. 807–810, 1996.

[21] G. Bi and M. Poo, “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type,” *J. Neurosci.*, vol. 18, no. 24, pp. 10 464–10 472, Dec. 1998.

[22] D. Goodman and R. Brette, ““Brian: a simulator for spiking neural networks in Python”,” *Frontiers in Neuroinformatics*, vol. 2, no. 5, Nov 2008.

[23] E. M. Izhikevich and F. Hoppensteadt, “Polychronous wavefront computations,” *International Journal of Bifurcation and Chaos*, vol. 19, pp. 1733–1739, 2009.