Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA, June 14-19, 2009

A Universal Abstract-Time Platform for Real-Time Neural Networks

A. D. Rast, M.M. Khan, X. Jin, L.A. Plana and S.B. Furber

***Abstract*— High-speed asynchronous hardware makes it possible to virtualise neural networks' temporal dynamics as well as their structure. Through SpiNNaker, a dedicated neural chip multiprocessor, we introduce a real-time modelling architecture that makes the neural model run on the device independent of the hardware specifics. The central features of this modelling architecture are: native concurrency, ability to support very large ($\gg 10^9$ neurons) networks, and decoupling of the temporal and spatial characteristics of the model from those of the hardware. It circumvents a virtually fatal tradeoff in large-scale neural hardware between model support limitations or scalability limitations, without imposing a synchronous timing model. The chip itself combines an array of general-purpose processors with a configurable asynchronous interconnect and memory fabric to achieve true on- and off-chip parallelism, universal network architecture support, and programmable temporal dynamics. An HDL-like concurrent configuration software model using libraries of templates, allows the user to embed the neural model onto the hardware, mapping the virtual network structure and time dynamics into physical on-chip components and delay specifications. Initial modelling experiments demonstrate the ability of the processor to support real-time neural processing using 2 different neural models. The complete system is therefore an environment able, within a wide range of model characteristics, to model real-time dynamic neural network behaviour on dedicated hardware.**

I. The Need for Dedicated Neural Network Hardware Support

NEURAL networks use an emphatically concurrent model of computation. This makes the serial uniprocessor architectures upon which many if not most neural simulators run [1] not only unable to support real-time neural modelling with large networks, but in fact architecturally unsuited to neural simulation at a fundamental level. Given that biological neural networks, and likewise many interesting computational problems, demonstrate temporal dynamics, a serial computer imposing hard synchronous temporal constraints is at best a poor fit and at worst unable to model networks that change in real time. For this reason dedicated neural network hardware embedding the concurrent model and the time dynamics into the architecture has long appeared attractive [2], [3], [4]. Yet the traditional digital serial model offers a critical advantage: general-purpose programmable functionality that let it simulate (if slowly) any neural network model at least in principle [5]. There has been some experimentation with hybrid approaches [6], but these impose a significant speed penalty for models and functions not integrated onto the device. Lack of flexibility is probably the main reason why neural hardware in practice

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

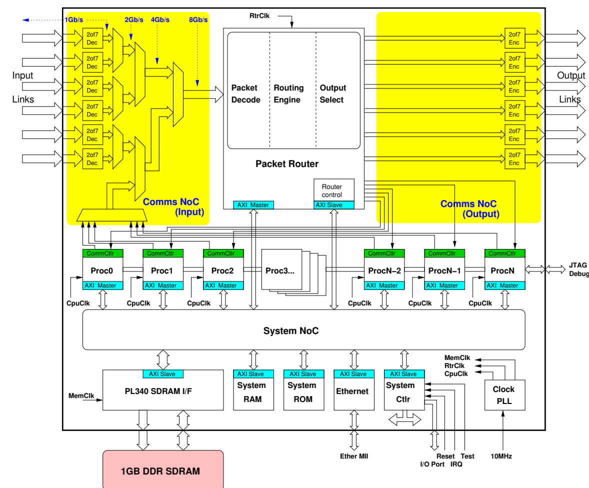

Fig. 1. SpiNNaker chip block diagram.

has had, at best, limited success: it is not very useful to have hardware for neural modelling if the hardware forces the user to make an *a priori* decision as to the model he is going to use. If it is basic that large-scale real-time neural modelling necessitates dedicated hardware [7], it is therefore equally essential that the hardware support concurrent processing with a time model having the same level of programmability that digital computers can achieve with function. Likewise, existing software models designed for synchronous serial uniprocessor architectures are unsuitable for parallel, real-time neural hardware, and therefore a fully concurrent, hardware-oriented modelling system is a pressing need. We propose a system that borrows its underlying model and development tools from the hardware design environment, using a describe-synthesize-simulate flow to develop neural network models possessing native concurrency with accurate real-time dynamics. Our approach to practical hardware neural networks develops software and hardware with a matching architectural model: a configurable "empty stage" of generic neural components, connectivitity and programmable dynamics that take their specific form from the neural model superposed on them.

II. SpiNNaker: A General-Purpose Neural Chip Multiprocessor

SpiNNaker (fig. 1) [8] is a dedicated chip multiprocessor (CMP) for neural simulation designed to implement the concept of the neural empty stage. A full scale SpiNNaker

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

system envisions more than a million processing cores distributed over these CMP's to achieve a processing power of up to 262 TIPS with high concurrent inter-process communication (6 Gb/s per chip). Such a system would be able to simulate a population of more than 10^9 simple spiking neurons: the scale of a small mammalian brain. Architecturally, it is a parallel array of general-purpose microprocessors (ARM968) embedded in an asynchronous network-on-chip with both on-chip and inter-chip connectivity. An off-chip SDRAM device stores synaptic weights, and an on-chip router configures the network-on-chip so that signals from one processor may reach any other processor in the system, (whether on the same chip or a different chip) if the current configuration indicates a connection between the processors. The asynchronous network-on-chip allows each processor to communicate concurrently, without synchronising either to each other or to any global master clock [9]. Importantly, this implies that the chip embeds *no explicit time model*. For real-time applications, time "models itself": the clock time in the real world is the clock time in the virtual simulation, and in applications that use an abstract-time or scaled-time representation, time is superimposed onto the SpiNNaker hardware substrate through the model's configuration, rather than being determined by internal hardware components. Spatially, as well, there is no explicit topological model. A given neural connection is not uniquely identified with a given hardware link, either on-chip or between chips, so that one link can carry many signals, and the same signal can pass over many different link paths in the physical hardware topology without affecting the model connection topology of the neural network itself. Nor are processors uniquely identified with a particular neuron: the mapping is not 1-to-1 but rather many-to-one, so that in general a processor implements a collection of neurons which may be anywhere from 1 to populations of tens of thousands depending on the complexity of the model and the strictness of the real-time update constraints. SpiNNaker is therefore a completely generic device specialised for the neural application: it has an architecture which is naturally conformable to neural networks without an implementation that confines it to a specific model.

III. The SpiNNaker hardware model

A. SpiNNaker local processor node: the neural module

The local processor node (fig. 2) is the system building block - the on-chip hardware resource that implements the neural model. SpiNNaker uses general-purpose low-power ARM968 processors to model the neural dynamics. Each processor also contains a high-speed local Tightly Coupled Memory (TCM), arranged as a 32K instruction memory (ITCM) and a 64K data memory (DTCM). A single processor does not implement a single neuron but instead a group of neurons (with number dependent on the complexity of the model); running at 200MHz a processor can simulate about 1000 simple yet biologically plausible neurons such as [10], using the ITCM to contain the code and the DCTM the

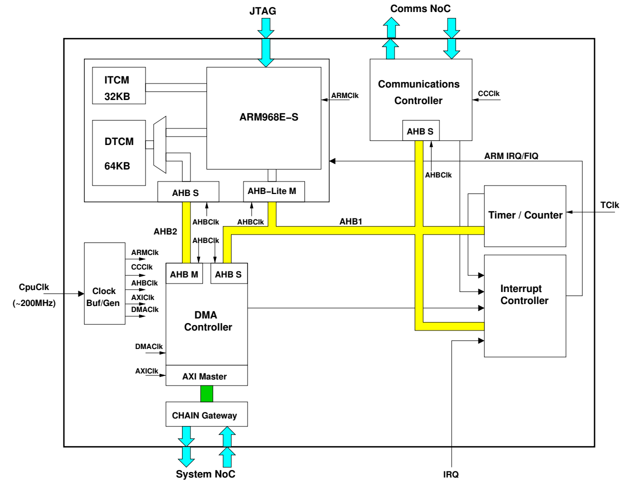

Fig. 2. ARM968E-S with Peripherals.

neural state data. We have optimised SpiNNaker for spiking neural networks, with an execution model and process communications optimised for concurrent neural processing rather than serial-dominated general-purpose computing. In the spiking model, neurons update their state on receipt of a spike, and likewise output purely in the form of a spike (whose precise "shape" is considered immaterial), making it possible to use event-driven processing [5]. An on-board communications controller embedded with each core controls spike reception and generation for all the neurons being simulated on its associated processor. In addition, the processor contains a programmable hardware timer, providing a method to generate single or repeated absolute real-time events. The processor node does not contain the synaptic memory, which resides instead in off-chip SDRAM and is made "virtually local" through an integrated DMA controller. The local node therefore, rather than being a fixed-function, fixed-mapping implementation of a neural network component, appears as a collection of general-purpose event-driven processing resources.

B. SpiNNaker memory system: the synapse channel

The second subsystem, the synapse channel, is the hardware resource that implements the synaptic model. Since placing the large amount of memory required for synapse data on chip would consume excessive chip area, we use an off-the-shelf SDRAM device as the physical memory store and implement a linked chain of components on-chip to make synapse data appear "virtually local" by swapping it between global memory and local memory within the interval between events that the data is needed. The critical components in this path are an internal asynchronous Network-on-Chip (NoC), the System NoC, connecting master devices (the processors and router) with slave memory resources at 1GB/s bandwidth, and a local DMA controller per node able to transfer data over the interface at 1.6 GB/s in sequential burst requests. We have previously demonstrated [11] that

the synapse channel can transfer required synaptic weights from global to local memory within a 1 ms event interval while supporting true concurrent memory access so that concurrent requests from different processors remain non-blocking. This makes it possible for the processor to maintain real-time update rates. Analogous to the process virtualisation the neural module achieves for neurons, the synapse channel achieves memory virtualisation by mapping synaptic data into a shared memory space, and therefore not only can SpiNNaker implement multiple heterogeneous synapse models, it can place these synapses anywhere in the system and with arbitrary associativity.

C. SpiNNaker event-driven dynamics: the spike transaction

An *event*: a point process happening in zero time, is the unit of communication that implements the temporal model. SpiNNaker uses a vectored interrupt controller (VIC) in each processor core to provide event notification to the processor. Events are processor interrupts and are of 2 principal types. The more important type - and the only one visible to the neural model - is the spike event: indication that a given neuron has signalled to some neuron the current processor is modelling. Spikes use Address-Event Representation (AER): an abstraction of the actual spike in a biological neuron that simplifies it to a zero-time event [12] containing information about the source neuron, and possibly a 32-bit data payload (fig. 3). The second type is the process event: indication of the completion of an internal process running on a hardware support component. Process events make possible complete time abstraction by freeing the processor from external synchronous dependencies. Within the ARM CPU, a spike event is a Fast Interrupt Request (FIQ) while a process event is an Interrupt Request (IRQ) [13]. To program a time model the user programs the VIC, interrupt service routine (ISR), and, if needed, the internal timer. Since each processor has its own independent VIC, ISR, and timer, SpiNNaker can have multiple concurrent time domains on the same chip or distributed over the system. It is because interrupts, and hence events, are asynchronous, i.e. they can happen at any time, that SpiNNaker can have a fully abstract, programmable time model: interrupt timing sets the control flow and process sequencing independently of internal clocks.

D. SpiNNaker external network: the virtual interconnect

The external network is a topological wireframe - the hardware resource that implements the model netlist. Signals propagate over a second asynchronous NoC, the Communications NoC [9], supporting up to 6 Gb/s per chip bandwidth [8] that connects each processor core on a chip to the others, and each chip with six other chips. The hub of the NoC is a novel on-chip multicast router that routes the packets (spikes) to the internal on-chip processing cores and external chip-to-chip links using source-based associative routing. Each chip's Communications NoC interface links to form a global asynchronous packet-switching network where a chip is a node. A system of any desired scale can be formed by linking chips to each other with the help of these links, continuing this process until the system wraps itself around to form a toroidal mesh of interconnected chips as shown in Figure 4. Because the NoC is a *packet-switched* system, the physical topology of the hardware is completely independent of the connection topology of the network being modelled. To map physical links into neural connections each chip has a configurable router containing 1024 96-bit associative routing entries that specify the routes associated with any incoming packet's routing key. A default routing protocol for unmatched inputs in combination with hierarchical address organisation minimises the number of required entries in the table. By configuring the routing tables (using a process akin to configuring an FPGA) [14], the user can implement a neural model with arbitrary network connectivity on a SpiNNaker system. Since, like the System

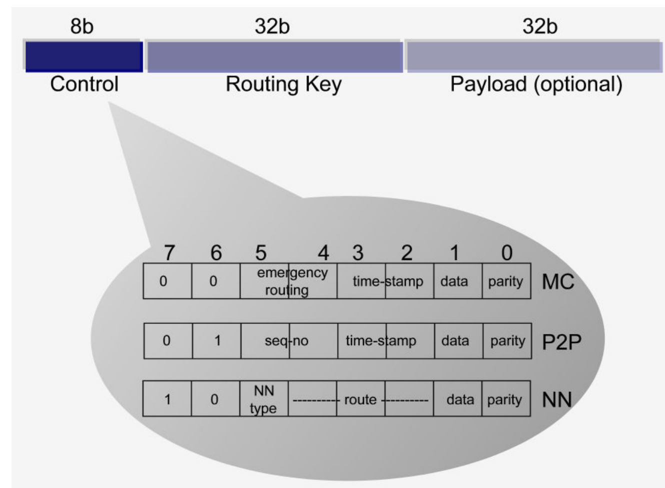

Fig. 3. SpiNNaker AER spike packet format. Spike packets are usually type MC. Types P2P and NN are typically for system functions.

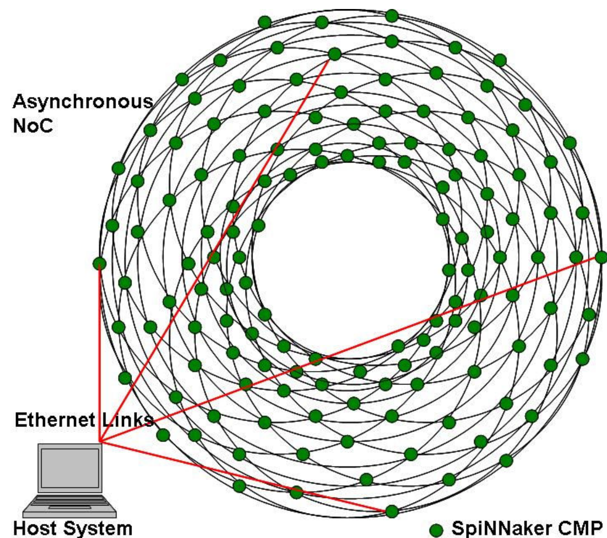

Fig. 4. Multichip SpiNNaker CMP System.

NoC, the Communications NoC is asynchronous, there is no deterministic relation between packet transmission time at the source neuron and its arrival time at the destination(s). Spike timing in the model is a function of the programmed temporal dynamics, independent of the specific route taken through the network. Once again this decouples the communications from hardware clocks and makes it possible for a SpiNNaker system to map (virtually) any neural topology or combination of topologies to the hardware with user-programmable temporal model.

IV. The SpiNNaker software model

A. 3-level system

From the point of view of the neural modeller, SpiNNaker hardware is a series of generic processing blocks capable of implementing specific components of neural functionality. The user would typically start with a neural model description which needs to be transformed into its corresponding SpiNNaker implementation. Modellers will most likely not be familiar with, or necessarily even interested in, the low-level details of native SpiNNaker object code and configuration files. Users working at different levels of abstraction therefore need an automated design environment to translate the model description into hardware object code to load to the device. We have created a software model (fig. 5) based on the flow of hardware description language (HDL) tools, that use a combination of synthesis-driven instantiation [15] (automated generation of hardware-level netlists using libraries of templates that describe implementable hardware components) and concurrent simulation environments [16] (software that uses a parallel simulation engine to run multiple processes in parallel). This environment uses SystemC [17] as its base, the emerging standard for high-level hardware modelling. SystemC has several important advantages. It is a concurrent language, matching well the massive parallelism of neural networks. It contains a built-in simulator that eliminates the need to build one from the ground up. It has also been designed to abstract hardware details while providing cycle-accurate realism if necessary, making it possible to target specific hardware with a behavioural model while retaining accurate temporal relationships at each stage. Critically, SystemC supports the use of class templating, the ability to describe an object generically using the template parameters to specify the particular implementation. This makes it possible to use the same model at different levels of abstraction simply by changing the template parameter. The software environment defines 3 levels of abstraction: device level, system level, and model level. At the device level, software functions are direct device driver calls written mostly in hand-coded assembly that perform explicit hardware operations without reference to the neural model. The system level abstracts device-level functions to neural network functions, implementing these functions as SpiNNaker-specific operation sequences: templates of neural functionality that invoke a given hardware function. At the model level, there is no reference to

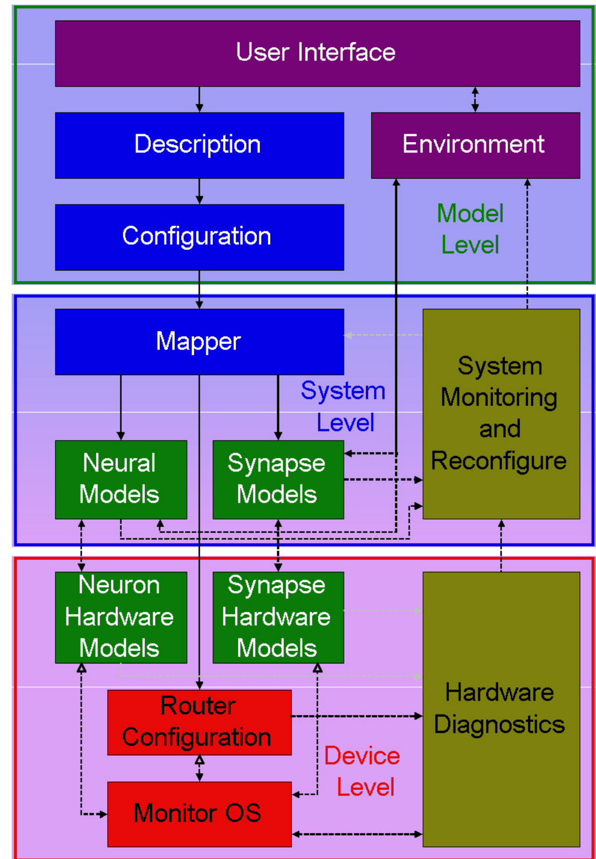

Fig. 5. SpiNNaker system software flow. Arrows indicate the direction in which data and files propagate through the system. A solid line represents a file, where dashed lines indicate data objects. Boxes indicate software components, the darker boxes being high-level environment and model definition tools, the lighter ones hardware-interfacing components that possess data about the physical resources on the SpiNNaker chip.

SpiNNaker (or any hardware) as such; the modeller describes the network using abstract neural objects that describe broad classes of neural behaviour. In principle a network described at the model level could be instantiated on any hardware or software system, provided the library objects at their corresponding system and device levels existed to "synthesize" the network into the target implementation. This 3-level, HDL-like environment allows modellers to develop at their own level of system and programming familiarity while retaining the native concurrency inherent to neural networks and preserving spatiotemporal relations in the model.

B. Model level: concurrent generic description

The model level considers a neural network as a process abstraction. The user describes the network as an interaction between 2 types of containers: neural objects and synaptic objects. Both types of containers represent groups of individual components with similar behaviour: for example, a neural object could represent a group of 100 neurons with identical basic parameters. This makes it possible to describe large neural networks whose properties might be specified statistically by grouping components generated from the

same statistical distribution within a single object. Although we use the terms "neural" and "synaptic" for convenience, a neural object need not necessarily describe only neurons: the principal difference between the objects is that the synaptic object is an active SystemC *channel* and therefore defines a communication between processes whereas a neural object is a *module*. These objects take template parameters to describe their functionality: object classes that define the specific function or data container to implement. The most important of these classes are functions (representing a component of neural dynamics), signal definitions (determining the time model and data representation) and netlists (to represent the connectivity). Thus, for example, the modeller might define a set of differential equations for the dynamic functions, a spike signal type, and a connection probability to generate the netlist, and instantiate the network by creating neural and synaptic objects referencing these classes in their templates. At the model level, therefore, specifying time is entirely a matter of the template definition: the user determines the time model in the specification of the dynamic functions and the signal type. Processes execute and communicate concurrently using SystemC's asynchronous event-driven model. Since at this level the model is a process abstraction, it could run, in principle, on any hardware platform that supports such asynchronous communications (as SpiNNaker does) while hiding low-level timing differences between platforms.

C. System level: template instantiation of library blocks

At the system level, the model developer gains visibility of the neural functions SpiNNaker is able to implement directly. System-level models can be optimised for the actual hardware, and therefore can potentially run faster; however, they run more slowly in software simulation because of the need to invoke hardware-emulation routines. Our approach uses template parameters to access the hardware components. A given system-level object is a generalised neural object similar to a model-level object, whose specific functionality comes from the template. At the system level, however, a template is a hardware "macro" - for example a function GetWeights() that requests a DMA transfer, performs the requisite memory access, and retrieves a series of weights, signalling via an interrupt when complete. Time at the system level is still that of the neural model, reflecting the fact that the only hardware event visible is the spike event. The user at system level specifies time as an input argument to the template functions: the "real time" the process would take to complete in the model. The processor can then arbitrarily reorder actual hardware timing as necessary to optimise resource use while preserving real-time-accurate behaviour. We have earlier shown [18] how to use this reordering capability to achieve accurate neural updating and STDP synaptic plasticity in an event-driven system. Both processes are interrupt-driven routines with a deferred process, programmed into instruction memory. System level descriptions are the source input for the SpiNNaker "synthesis" process: a bridge between the model level and the device level that uses templates as the crucial link to provide a SpiNNaker hardware abstraction layer.

D. Device Level: a library of optimised assembly routines

The device level provides direct interfacing to SpiNNaker hardware as a set of event-driven component device drivers. In the "standard SpiNNaker application model" events are interrupts to the neural process, triggering an efficient ISR to call the developer-implemented system-level neural dynamic function associated with each event. ISR routines therefore correspond directly to device-level template parameters. The device level exposes the process events as well as the spike event, and therefore the programmer specifies a time model by explicit configuration of the hardware devices: the timer, the DMA controller, and the communications controller, along with the ARM968 assembly code. Time at device level is the "electronic time" of the system, as opposed to the "real time" of the model. We have implemented an initial function library as part of the configuration process for the SpiNNaker system using ARM968 assembly language for optimal performance. This device driver library includes the functions needed by a neural application where it has to interact with the hardware to model its dynamics. We have optimised the functions to support real-time applications in an event-driven model with efficient coding and register allocation schemes [19]. We have also optimised the SpiNNaker memory map so that so that the processor will start executing the ISR in just one cycle after receiving the event. In our reference application sending a spike requires 4 ARM instructions while receiving a spike requires 27 instructions, including the DMA request to upload the relevant data block into the local memory. By providing a ready-made library of hardware device drivers we have given users access to carefully optimised SpiNNaker neural modelling routines while also presenting a template for low-level applications development should the user need to create his own optimised hardware drivers for high-performance modelling.

V. SpiNNaker system design and simulation

A. Design of hardware components

The SpiNNaker chip is a GALS system [9] - that is, a series of synchronous clocked modules embedded in an asynchronous "sea". Such a system typically requires a mix of design techniques. To minimise development time we have attempted where possible to use industry-standard tool flows and off-the-shelf componentry. Most of the processor node, including the ARM968 and its associated interrupt controller and timers, along with the memory interface, are standard IP blocks available from ARM. A further set of blocks: the communications controller, the DMA controller, and the router, were designed in-house using synchronous design flow with HDL tools. We implemented these components using Register Transfer Level (RTL)-level descriptions in Verilog and tested them individually using the industry-standard Synopsys VCS concurrent Verilog simulation environment. Finally,

the asynchronous components: the NoC's and the external communications link, used a combination of synthesis-like tools from Silistix and hand-designed optimisation to achieve required area and performance constraints. Where component design has used low-level cycle-accurate tools, system-level hardware testing and verification, by contrast, is being done using higher-level SystemC tools.

B. SystemC modelling and chip-level verification

One of the main objectives of this work has been to provide an early platform to develop and test applications for SpiNNaker while the hardware is still in the design phase. It is possible to verify the cycle accurate behaviour of individual components using HDL simulation. However, verifying the functional behaviour of a neural computing system on the scale of SpiNNaker would be unmanageably complex, either to demonstrate theoretically, or to simulate using classical hardware description languages such as VHDL and Verilog. In addition, industry-standard HDL simulators emphasize synchronous design, making verification of asynchronous circuits difficult and potentially misleading. Therefore, as part of the SpiNNaker project, we have created a SystemC system-level model for the SpiNNaker computing system to verify its functional behaviour - especially the new communications infrastructure. SystemC supports a higher level of timing abstraction: the Transaction Level Model (TLM), exhibiting "cycle approximate" behaviour. In a TLM-level simulation, data-flow timing remains accurate without requiring accuracy at the level of individual signals. This makes it possible on the one hand to integrate synchronous and asynchronous components without generating misleading simulation results, and on the other to retain timing fidelity to the neural model since the data-flow timing entirely determines its behaviour. We developed SystemC models for in-house components [20] and integrated them with a cycle-accurate instruction set simulator for the ARM968E-S processor and its associated peripherals using ARM SoC Designer. SoC Designer does not support real-time delays, therefore we captured the behaviour of the asynchronous NoC in terms of processor clock cycles. We then tested the complete system behaviour extensively in two neural application case studies [8]. For simplicity of simulation on a host PC with limited memory, the model simulates 2 processing cores per chip - the number of cores on the initial test chip. With all chip components in the simulation, we were able to achieve a simulation of 9 chips running concurrently, thus verifying both on-chip and inter-chip behaviour. With system-level functional verification we have thus been able to achieve both a strong demonstration of the viability of the SpiNNaker platform for real-world neural applications and a methodology for the development and testing of new neural models prior to their instantiation in hardware.

C. Performance and functionality testing

To verify the event-driven model and develop the neural library routines we used an ARM968 emulation built with ARMs SOC designer, containing one processing node together with its associated peripherals: DMA controller, router, interrupt controller, and memory system. Router links wrap around connecting outputs to inputs, so that all packets go to the emulated processor. We successfully tested a reference neural network using Izhikevich [10] neural dynamics for 1000 neurons, with random connectivity, initial states, and parameters, updating neural state once per ms. Using assembly code programming and 16-bit fixed point arithmetic (demonstrated in [21], [22]) it takes 8 instructions to complete one update. Modeling 1,000 neurons with 100 inputs each (10% connectivity) firing at 10Hz, requires 353 μs SpiNNaker time to simulate 1 ms of neural behaviour. Modelling 1 ms for 1,000 neurons with 1,000 inputs each (100% connectivity) firing at 1 Hz requires approximately the same computational time. The model can therefore increase the connectivity by reducing the firing rates as real neural network systems do, without losing real-time performance. This model ran on the SystemC model with a small population of neurons, reproducing the results presented in [23]. Using the results from this experiment, we ran a cycle-accurate simulation of the top-level model (fig. 6) to analyse the impact of system delays on spike-timing accuracy. Figures 7 and 8 show the results. Each point in the raster plot is one spike count in the histogram. The spike-raster test verified that there are no synchronous system side effects that systematically affect the model timing. We used the timing data from the simulation to estimate the timing error for each of the spikes in the simulation - that is, the difference between the model-time "actual" timing of the spike and the system-time "electronic" timing of the spike - by locally increasing the timing resolution in the analytic (floating-point) Izhikevich model to 50 μs in the vicinity of a spike and recomputing the actual spike time. Most spikes (62%) have no timing error, and more than 75% are off by less than 0.5ms - the minimum error necessary for a spike to occur ± 1 update off its "true" timing. Maximum error is, as expected, 1 ms, since the update period fixes a hard upper bound to the error. In combination with the raster plot verifying no long-term drift, the tests indicate that SpiNNaker can maintain timing fidelity within a 1 ms resolution.

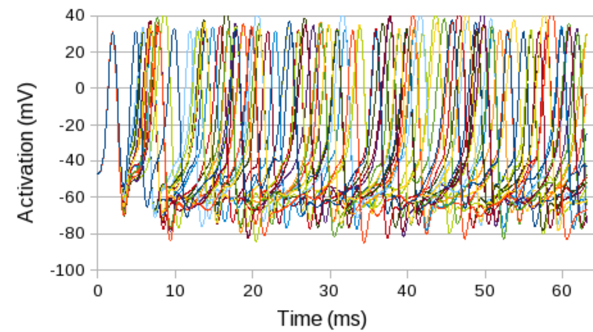

Fig. 6. SpiNNaker top-level model output of the spiking network. For clarity only 25 neurons are shown.

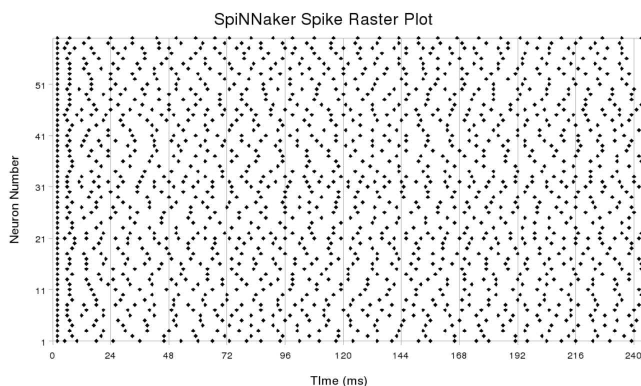

Fig. 7. SpiNNaker spiking neural simulation raster plot. The network simulated a random network of 60 neurons, each given an initial impulse at time 0. To verify timing synaptic plasticity was off.

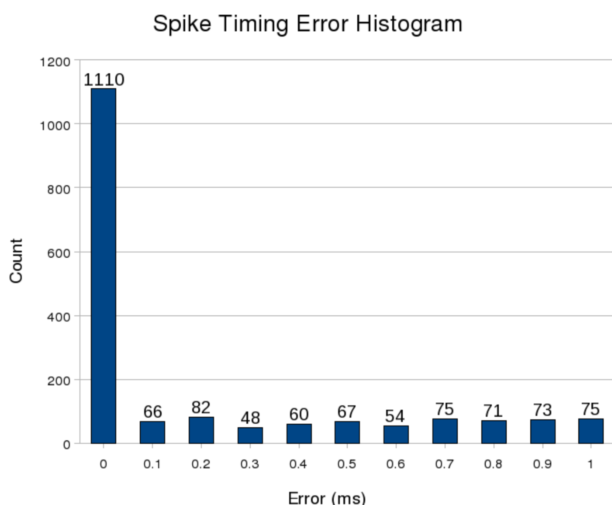

Fig. 8. SpiNNaker spike error histogram. Estimated spike-timing errors are from the same network as the raster plot.

VI. Neural modelling implications

An important observation of the tests is that in a system with asynchronous components, the behaviour is nondeterministic. While most of the spikes occurred without timing error, some had sufficient error to occur at the next update interval. The effect of this is to create a ± 1 ms timing jitter during simulation. Biological neural networks also exhibit some jitter, and there is evidence to suggest that this random phase error may be computationally significant [24]. It is not clear whether asynchronous communications replicates the phase noise statistics of biological networks, but its inherent property of adding some phase noise may make it a more useful platform for exploration of these effects than purely deterministic systems to which it is necessary to add an artificial noise source. In addition, the modeller can (statically) tune the amount of noise, to some degree, by programming the value of the update interval acting as an upper bound on the phase noise. A GALS system like SpiNNaker therefore appears potentially capable of reproducing a wider range of neural behaviours than traditional synchronous systems, while supporting the programmability that has been a limiting factor in analogue neuromorphic devices.

The combination of design styles we used within a concurrent system of heterogeneous components is very similar to typical situations encountered in neural network modelling. This synergy drove our adoption of the hardware design flow as a model for neural application development. Notably, the environment we are implementing incorporates similar principles: off-the-shelf component reuse through standard neural library components; synthesis- directed network configuration using automated tools to generate the hardware mapping; and mix of design abstraction levels. The essential feature of hardware design systems: native support for concurrent description and simulation, is likewise essential in real-time neural network modelling, and we also note, *not* essential in ordinary software development. In particular, most software development does not incorporate an intrinsic notion of time: the algorithm deterministically sets the process flow which then proceeds as fast as the CPU will allow. Hardware design systems, by contrast, must of necessity include a notion of time, given that timing verification is one of the most important parts of the process. HDL-like systems also provide a logical evolutionary migration path from software simulation to hardware implementation, since the same model, with different library files, can be used to simulate at a high level, to develop hardware systems, or to instantiate a model onto a developed hardware platform.

Considerable work remains to be done both on SpiNNaker and generally in the area of neural development tools. Immediate future efforts in the hardware will focus on fabrication and testing of the chips prior to scaling system size. In addition we will verify simulation results on the physical hardware. For larger systems statistical description models as well as formal theories for neural network model design may be necessary. There remains also an open question of verification in a GALS system such as SpiNNaker: with nondeterministic timing behaviour, exact replication of the output from simulation is both impossible and irrelevant. It is important to develop meaningful test criteria for the finished device, focussing on replication of timing in the neural model independent of system-level timing. With respect to the software model, developing a high-level mapping tool [14] to automate the “synthesis” process is a priority. We are also working on creating a SystemC-based user development environment that allows the modeller to implement the neural model with a high-level graphical or text description and use the automated generation flow to instantiate it upon SpiNNaker. Work is ongoing on extending the neural library with additional neural models, including extensions to non-spiking representations. Of particular interest are the popular, time-independent MLP networks. While it is important that the chip and the software work with dynamic models incorporating a concept of time, demonstration of a “timeless” model extends the capabilities of the system towards a truly

general-purpose neural modelling environment.

VII. Conclusions

Design of an integrated hardware/software system like SpiNNaker provides a powerful model for neural network simulation: the hardware design flow of behavioral description, system synthesis, and concurrent simulation. With this work we also emphasize one of the most important features of our hardware-design-flow methodology: the ability to leverage existing industry-standard tools and simulators so that it is unnecessary to develop a complete system from the ground up. The concept SpiNNaker embodies: that of a plastic hardware device incorporating dedicated neural components but neither hardwired to a specific model nor an entirely general-purpose reconfigurable device such as an FPGA, is the hardware equivalent of the software model, and is a new and perhaps more accessible neural hardware architecture than previous model-specific designs. By making the hardware platform user-configurable rather than fixed-model, we introduce a new type of neural device whose architecture matches the requirements of large-scale experimental simulation, where the need to configure and test multiple and potentially heterogenous neural network models within the same environment is critical. Such a model, we propose, is more suitable to neural network modelling than existing systems, especially when the model has asynchronous real-time dynamics. An asynchronous event-driven communications model makes it easier to design for arbitrary model timing and delays since it is not necessary to sample updates according to a global clock. Exploration of nondeterministic time effects also seems likely to occupy a growing interest within the neural research community, and devices such as SpiNNaker that have similar properties could reveal behaviours unobservable in conventional processors. In time it would also be ideal to move from a GALS system to a fully asynchronous system, ultimately, perhaps, to incorporate analogue neuromorphic components. Such a hybrid system would offer configurable processing with the speed and accuracy of analogue circuits where appropriate, instantiatable using a descendant of the software model we have developed. This system is the future version, as much as SpiNNaker is the present version, of a neural network matching the development model and environment to the computational model.

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. Jahnke, U. Roth, and T. Schönauer, *Pulsed Neural Networks*. Cambridge, MA: MIT Press, 1999, ch. Digital Simulation of Spiking Neural Networks, pp. 237–257.

[2] W. C. Westerman, D. P. M. Northmore, and J. G. Elias, "Antidromic spikes drive hebbian learning in an artificial dendritic tree," *Analog Circuits and Signal Processing*, vol. 19, no. 2–3, pp. 141–152, Feb. 1999.

[3] N. Mehrtash, D. Jung, H. H. Hellmich, T. Schönauer, V. T. Lu, and H. Klar, "Synaptic plasticity in spiking neural networks (SP^2INN) a system approach," *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 980–992, Sep. 2003.

[4] 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, no. 1, pp. 211–221, Jan. 2006.

[5] S. B. Furber, S. Temple, and A. Brown, "On-chip and inter-chip networks for modelling large-scale neural systems," in *Proc. International Symposium on Circuits and Systems, ISCAS-2006*, May 2006.

[6] M. Oster, A. M. Whatley, S.-C. Liu, and R. J. Douglas, "A hardware/software framework for real-time spiking systems," in *Proc. 15th Int'l Conf. Artificial Neural Networks (ICANN2005)*. Springer-Verlag, 2005, pp. 161–166.

[7] C. Johansson and A. Lansner, "Towards cortex sized artificial neural systems," *Neural Networks*, vol. 20, no. 1, pp. 48–61, Jan. 2007.

[8] 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.

[9] 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, Sept.-Oct. 2007.

[10] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Networks*, vol. 14, pp. 1569–1572, Nov. 2003.

[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] 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.

[13] *ARM9E-S Technical Reference Manual*, ARM Limited, 2002.

[14] 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.

[15] D. Lettnin, A. Braun, M. Bodgan, J. Gerlach, and W. Rosenstiel, "Synthesis of embedded SystemC design: A case study of digital neural networks," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE'04)*, vol. 3, 2004, pp. 248–253.

[16] S. Modi, P. Wilson, A. Brown, and J. Chad, "Behavioral simulation of biological neuron systems in SystemC," in *Proc. 2004 IEEE Int'l Behavioral Modeling and Simulation Conf.*, 2004, pp. 31–36.

[17] P. Panda, "SystemC - a modeling platform supporting multiple design abstractions," in *Proc. Int'l Symp. on System Synthesis*, 2001.

[18] 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.

[19] A. N. Sloss, D. Symes, and C. Wright, *ARM System Developer's Guide - Designing and Optimizing System Software*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[20] M. M. Khan, X. Jin, S. Furber, and L. Plana, "System-level model for a GALS massively parallel multiprocessor," in *Proc. 19th UK Asynchronous Forum*, 2007, pp. 9–12.

[21] 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.

[22] T. Daud, T. Duong, M. Tran, H. Langenbacher, and A. Thakoor, "High resolution synaptic weights and hardware-in-the-loop learning," in *Proc. SPIE - Int'l Soc. Optical Engineering*, vol. 2424, 1995, pp. 489–500.

[23] 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.

[24] P. H. E. Tiesenga and T. J. Sejnowski, "Precision of pulse-coupled networks of integrate-and-fire neurons," *Network: Computation in Neural Systems*, vol. 12, no. 2, pp. 215–233, May 2001.