

# Real Time On-Chip Implementation of Dynamical Systems with Spiking Neurons

Francesco Galluppi, Sergio Davies and Steve Furber  
Advanced Processor Technologies Group  
University of Manchester, United Kingdom  
Email: francesco.galluppi@cs.man.ac.uk

Terry Stewart and Chris Eliasmith  
Centre for Theoretical Neuroscience  
University of Waterloo, Ontario, Canada

**Abstract**—Simulation of large-scale networks of spiking neurons has become appealing for understanding the computational principles of the nervous system by producing models based on biological evidence. In particular, networks that can assume a variety of (dynamically) stable states have been proposed as the basis for different behavioural and cognitive functions.

This work focuses on implementing the Neural Engineering Framework (NEF), a formal method for mapping attractor networks and control-theoretic algorithms to biologically plausible networks of spiking neurons, on the SpiNNaker system, a massive programmable parallel architecture oriented to the simulation of networks of spiking neurons. We describe how to encode and decode analog values to patterns of neural spikes directly on chip. These methods take advantage of the full programmability of the ARM968 cores constituting the processing base of a SpiNNaker node, and exploit the fast Network-on-chip for spike communication.

In this paper we focus on the fundamentals of representing, transforming and implementing dynamics in spiking networks. We show real time simulation results demonstrating the NEF principles and discuss advantages, precision and scalability. More generally, the present approach can be used to state and test hypotheses with large-scale spiking neural network models for a range of different cognitive functions and behaviours.

## I. INTRODUCTION

Construction of large-scale spiking neural models is possible thanks to the emergence of unified approaches that are able to scale up seamlessly. These models can be simulated taking advantage of recent developments in computational infrastructure that can equally be scaled up. Some models aim to find emerging functions from the structural data known from biology. For example, quantitative descriptions of cortex based on anatomical data [4] can be used to model systems that naturally scale up [19], due to the regularity of the laminar organization of the thalamo-cortical system [35]. Other approaches can be considered more functional, where neural dynamics and quantities act as biological constraints in modelling specific cognitive functions [7]. Some functions can be modelled using attractor networks [1], networks that can represent information by settling to a (dynamically) stable state with their self-sustained, persistent activity. For example functions like memory can be associated to brain areas that are believed to use attractor representation such as the hippocampus [37].

Simulating large scale networks of biologically plausible neurons is a challenging task which require scalable compu-

tational and communication resources. Therefore simulations usually take place on supercomputers [2], general purpose hardware as FPGAs [24] or dedicated neuromorphic hardware [28] [36]; every approach has different scalability, programmability, precision and power consumption characteristics.

In this context we describe how to map the principles of the Neural Engineering Framework (NEF) [10], a unified approach for implementing complex neuro-dynamical systems and mapping control-theoretic algorithms with the neural connections between a highly heterogeneous population of spiking neurons, to the SpiNNaker System [13], a massively parallel programmable architecture oriented to the simulation of large scale models of spiking neural networks.

The paper describes the approach taken to encode and decode values directly on-chip, taking advantage of the programmability of the SpiNNaker system and exploiting the fast on-chip spike-based interconnect for communication between neural populations.

We show how a variety of networks can be built using encoding/decoding methods. In short, the approach presents the basis for testing large-scale neural models built with the NEF integrating SpiNNaker as the computational back-end in the existing framework and tools.

The rest of the paper is structured as follows: we introduce the Neural Engineering Framework and the SpiNNaker System in the first two sections. We then present the approach used to port the NEF on SpiNNaker, and present results obtained with the approach in sections IV and V respectively. Finally, discussion about how to expand the work and conclusions are drawn in the last two sections.

## II. NEURAL ENGINEERING FRAMEWORK

The Neural Engineering Framework [10] describes how biologically relevant variables can be encoded and processed in the dynamic neural activity of recurrently connected networks. This approach can be used to introduce complex control theoretic models into spiking neural networks, including standard attractor network models [8]. The NEF is captured by three principles:

- 1) **Representation** in neurons is defined by the combination of nonlinear encoding (exemplified by neuron tuning curves) and weighted linear decoding.

- 2) **Transformations** of neural representations are functions of variables that are represented by neural populations. Transformations are determined using an alternately weighted linear decoding which describes the transformation.
- 3) **Neural dynamics** are characterized by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analysed using control theory.

Nonlinear encoding is obtained by translating an analog value to a spike train for each neuron  $i$  in the encoding population as follows:

$$\delta(t - t_{in}) = G_i[\alpha_i \langle x \cdot e_i \rangle + J_i^{bias}] \quad (1)$$

where the spiking activity  $\delta(t - t_{in})$  is given by  $G_i$ , a nonlinear function describing the neural model, a gain factor  $\alpha_i$  (which in the scalar case is either 1 or -1), the value  $x$  to be encoded, the encoder  $e$  and a bias current  $J_i^{bias}$ . Encoding can be viewed as capturing the characteristic response (tuning curve) of a neuron to a specific stimulus space, demonstrated, for example, by tuning curves found in the visual system for orientation [6]. Tuning curves express the relation between the value of the stimulus and the spike response of a neuron, according to its preference (tuning) to the stimulus value.

An example of encoding is illustrated in fig. 1, which depicts the response of two example neurons out of a large population to an input stimulus  $x$ : the first neuron (blue tuning curve on the left) responds to negative values of  $x$ , by increasing its firing rate as the input tends to -1; the second neuron (green tuning curve on the right) responds to positive values of the input  $x$  instead. The response of the two neurons to a step input is shown in figure: the first neuron fires steadily when the input is negative, while it goes silent when the input is positive, as encoded by the second neuron which starts firing steadily. The original stimulus vector can be estimated by using decoding vectors  $d$  which can be found by a least square method [10], and coupled with a simple model of the post-synaptic current  $h(t)$  (a decaying exponential). Together, these give the following decoding scheme:

$$\hat{x} = \sum_{i,n} h(t - t_n) d_i \quad (2)$$

where  $h(t - t_n)$  is the convolution of the original spike train with the post-synaptic current (PSC) [29].

Encoding and decoding together define the neural population code. Conveniently, we can use encoding and decoding to compute neural connection weights. Specifically, we can calculate connection weights knowing the decoders for the source population and the encoders of the target population. Suppose we want to compute the function  $y = x$ , and we define representations for  $x$  and  $y$  as above. We can then determine connection weights to compute this function in the

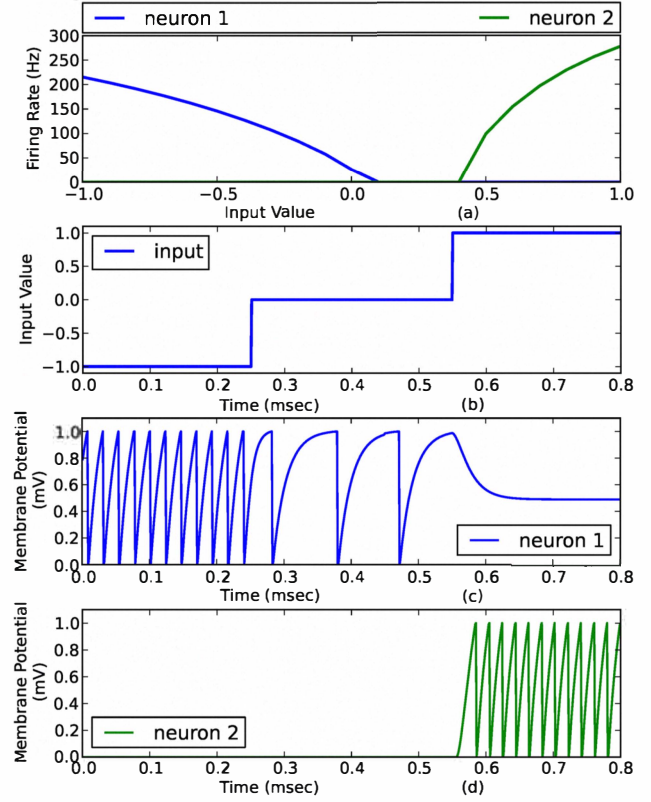


Fig. 1: (a) Two example neurons out of a heterogenous encoding population to an input stimulus  $x$ : the first neuron (blue tuning curve) responds to negative values of  $x$ , by increasing its firing rate as the input tends to -1; the second neuron (green tuning curve) responds to positive values of  $x$  by increasing its firing rate as the input tends to +1 (b) input stimulation (c, d) The sub-threshold voltage response of the two neurons to a step input is shown in figure: the first neuron fires steadily when the input is negative, while it goes silent when the input is positive, as encoded by the second neuron which starts firing steadily

$y$  population as:

$$\begin{aligned} \delta(t - t_{jm}) &= G_j[\alpha_j \langle (y = x) \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \langle \hat{x} \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \langle \sum_{i,n} h(t - t_n) d_i \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \sum_{i,n} \omega_{ij} h(t - t_n) + J_j^{bias}] \end{aligned}$$

So, in a fully spiking network with connection weights  $\omega_{ij} = \langle d_i \cdot e_j \rangle$ , we can decode the output of the  $y$  population to determine the input values  $x$  (i.e., it is computing  $y = x$ ). Critically, decoders can also be estimated to compute an arbitrary function  $f(x)$  other than identity, thus permitting a broad class of computations through transformational decoders  $d_i^{f(x)}$ . All such computations will be feedforward, however, and it should be noted that the accuracy of the computation is

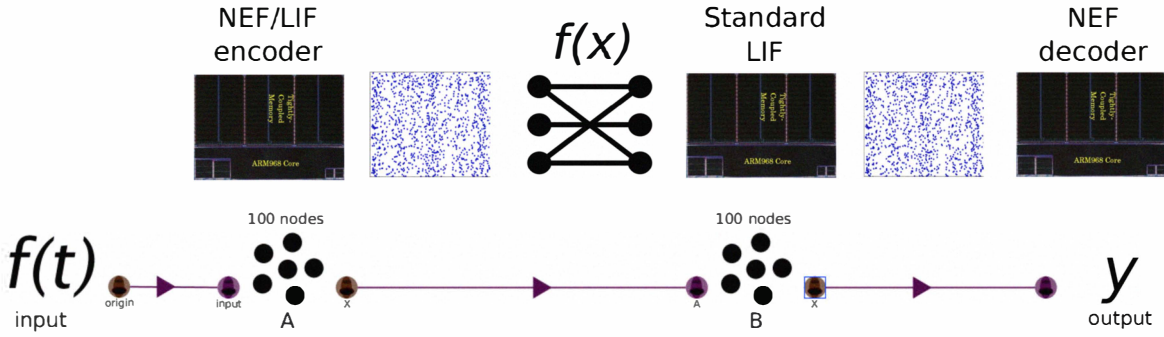


Fig. 2: **Approach:** Encoding and decoding processes happen directly on the SpiNNaker chip, through the use of two *ad-hoc* populations: the *LIF/NEF-encoder* and the *NEF-decoder* populations. The former translates values into spike trains for neurons in the population accordingly to eq. (1), while the latter collects spikes and estimates the output value using the decoders for the population connected to it, following eq. (2). On the rest of the chip, spikes travel in the neural space (consisting of standard LIF neurons) where the weights implement a function  $f(x)$  and are calculated starting from equation (4)

dependent on the number of neurons in the two populations and their neural properties. In particular, mean squared error is proportional to  $1/N$ , where  $N$  is the number of neurons in the neural population [8].

Introduction of neural dynamics allows variables represented by a neural population to be control theoretic state variables and hence apply modern control theory [23] methods to engineer and analyse time dynamics of network models, thus integrating standard attractor models and complex control [8]. Notably, the post synaptic currents dominate the dynamics of the neural system, so we can determine a general mapping between any dynamical system and a recurrently connected network that accounts for the difference in dynamics between our PSC model and ideal integration. For the simple case of an exponential PSC and a linear time-invariant (LTI) system, recurrent and feedforward weights can be computed from the input and dynamics matrices of the LTI system.

Specifically, a widely used model for the PSC with a time constant  $\tau$  is the exponential function [29] in the form  $h(t) = e^{-t/\tau}/\tau$ , which has as a Laplace transform  $h'(s) = 1/(1 + s\tau)$ . We can then transform the input and dynamics matrices (respectively  $\mathbf{A}$  and  $\mathbf{B}$ ) of LTI systems to neural equivalents, obtaining

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I} \quad (3)$$

$$\mathbf{B}' = \tau\mathbf{B}$$

where  $\mathbf{A}'$  and  $\mathbf{B}'$  are transformation matrices describing the dynamics in the neural system. These can be included directly in the recurrent and feedforward connection weights derived above. For example, the recurrent weights would be  $\omega_{ij} = \langle d_i \mathbf{A}' \cdot e_j \rangle$ .

More in general weights are the result of the multiplication of 3 matrices:

$$\omega_{ij} = \mathbf{d}_i^{F\beta} \cdot \mathbf{M}^{\alpha\beta} \cdot \mathbf{e}_j^\alpha \quad (4)$$

where terms are: the *decoding matrix* from an input population  $\beta$  which can be written as  $d_i^{F\beta}$ , decoding some function  $F$ ; the *encoding matrix* for the target population  $\alpha$  written as  $e_j^\alpha$ ;

the *transformation matrix*  $M^{\alpha\beta}$  that defines the transformation between populations. In the case of feed-forward computations  $M^{\alpha\beta}$  is the identity matrix and the function is computed by the transformational decoders  $d_i^{F\beta}$ ; in the case of recurrent connections,  $\alpha$  and  $\beta$  index the same population of neurons; in the case of mono-dimensional representation encoders and decoders are scalar; in the case of multidimensional representation  $d_i^{F\beta}$  and  $e_j^\alpha$  are vectors. Functions and transformations are then defined only by weights in the neural space.

The Neural Engineering Framework offers a unified approach to building complex dynamical systems in the neural space, using only spiking neuron models, PSC models, and connection weights. These models can represent arbitrary functions, while considering biological characteristics (e.g. tuning curves) as constraints. All parameters are estimated directly from neural data, or using the representation method described above. In this sense no parameters need to be tuned, as they either are calculated using the framework or estimated from neurobiological data. The NEF has been successfully used in modelling a wide variety of neural systems including those involved in sensory processing [11], motor control [20], and cognitive functions [9] such as decision making [32], both matching experimental data (neural and behavioral), and making a variety of novel predictions [33].

### III. SPINNAKER SYSTEM

The SpiNNaker System [14] is an asynchronous, multi-chip, multi-core, massive, programmable parallel system oriented to the simulation of heterogeneous large-scale models of spiking neural networks [27]. Each SpiNNaker chip contains 18 ARM968 cores embedded in a programmable, packet based, network on chip [26], where spikes are encoded as source-based AER [22] event packets and transmitted through a Multicast Router. Every core is equipped with a local Tightly-Coupled Memory (TCM - 32Kb for instruction and 64Kb for data), while each chip has access to 1Gb SDRAM shared by the 18 ARM cores, containing all the post synaptic information needed locally and eliminating problems of memory sharing

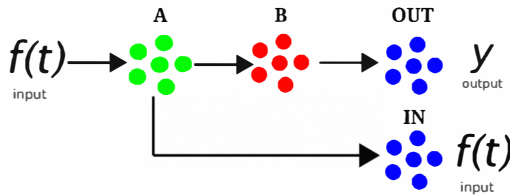


Fig. 3: Structure of the communication channel: the input value is encoded by population A (green, encoding population), which translates it into a population spike train. A is connected to B (standard LIF population) with an all to all connection. Weights between A and B are set to compute  $y = x$  in the communication channel experiment and  $y = x^2$  in the transformation experiment.

across the system. Each chip can be connected to 6 neighbour nodes in a toroidal mesh, supporting reconfigurable arbitrary connectivity through a multicast packet based routing system and 6 bi-directional asynchronous links [13]. From a computational and communication point of view each ARM core can be programmed to simulate in real time up to a 1000 simple neurons, receiving 1000 connections each and firing at a mean firing rate of 10 Hz, embedded in a configurable network based on the Multicast Router. In fact the number of neurons which can be modelled on a single core depends on factors including the activity of the neurons, the computational power needed to solve the neural dynamics equation in real time, the number of synapses and memory occupancy. Bigger populations are split across cores if needed, by using a hierarchical representation based on the Population abstraction [16]. The full system is designed to contain up to 65,536 chips and more than a million cores [15].

While not reaching the low power consumption [18] or speed [28] of dedicated analog hardware, due to its architecture the SpiNNaker System offers a compromise between performances of neuromorphic chips [17] and programmable, standard computing systems [2], within a low power budget of 1 Watt/SpiNNaker Chip [15]. Moreover it offers a custom packet switch network based on a Multicast Router [26], which is easily reconfigurable and less power consuming than a circuit-switched architecture [24] [5], and a memory system which is local to every chip, circumventing the challenges needed on GPUs to access memory [3] and maintain process coherency [25], while at the same time keeping power consumption lower than such systems. In this sense the SpiNNaker architecture is ideal for exploratory studies on large scale models which require programmability and fast reconfigurability within a tolerable power budget.

#### IV. INTEGRATING THE NEURAL ENGINEERING FRAMEWORK ON SPINNAKER

The NEF allows representation and computation of values and functions entirely in the neural space once the values are encoded/decoded using the framework. Hence it is possible to build a system that communicates only with spikes, by inputting and collecting them on a host machine which is

responsible for the encoding and decoding process. This approach however, while being very efficient for spike/AER based systems [22], does not scale up seamlessly as the size or firing rates (and consequently spikes needed to be sent from/to the system) of the encoding and decoding populations increase. Moreover the computational cost increases with the number of neuron: for example 3,000,000 neurons running on two Quad Core Intel Xeon E5540 processors with Hyper-Threading at 2.53GHz take 3 hours to produce 1 second worth of data; simulating it within a GPU environment leads to a 20x speedup (6-9 minutes per simulated second).

For this reason we exploit the programmability of the ARM968 cores constituting the computational heart of the SpiNNaker system [13] by implementing the encoding and decoding process directly on the SpiNNaker chip, through the use of two *ad-hoc* populations based on the leaky integrate-and-fire (LIF) neuron: the *NEF/LIF-encoder* and the *NEF-decoder* populations. The former translate values into spike trains for neurons in the population accordingly to principle 1, while the latter collects spikes and estimates the value using the decoders for the source population. In other words, the *NEF/LIF-encoder* population is implementing neurons that obey to equation (1) where the neural dynamics obey the standard LIF equation  $dV/dt = I/C - V/RC$ , while the *NEF-decoder* population is decoding values using equation (2). This kind of neural population is only used when the value represented needs to be explicitly represented, otherwise decoders are implicit in the connection weights, since communication between all other populations on the chip is done using spikes and standard LIF neurons. Those are weighted using NEF connection weights computed as described in equation (4). The approach is summarized in figure 2. Compared to a standard LIF model it adds a bias current proportional to the encoder and stimulus values.

Precision in encoders and decoders (and therefore in inter-connection weights) is crucial to avoid information corruption. Digital systems have the advantage of being programmable with a finite precision that can be evaluated. In this work we use a 32 bit fixed point implementation for neural state variables and parameters, including encoders and decoders, and 20 bit precision for the weights.

Spikes are therefore produced and collected only onboard, by taking advantage of the fast custom interconnect characterizing the SpiNNaker machine. This reduces the bandwidth and the load needed on a host, by sending and receiving only values to/from SpiNNaker. This approach tends to avoid difficulties and bottlenecks in translating and sending/receiving spikes directly from an host machine or from an FPGA by porting the encoding/decoding process onto SpiNNaker. Moreover it offers a "closed" spike system, where the interface communication consists in sending and receiving values for example from Nengo [34] (the software<sup>1</sup> that implements the NEF principles), or from a sensor or to a robotic arm while neural based computation is carried on board.

<sup>1</sup>available at <http://nengo.ca>

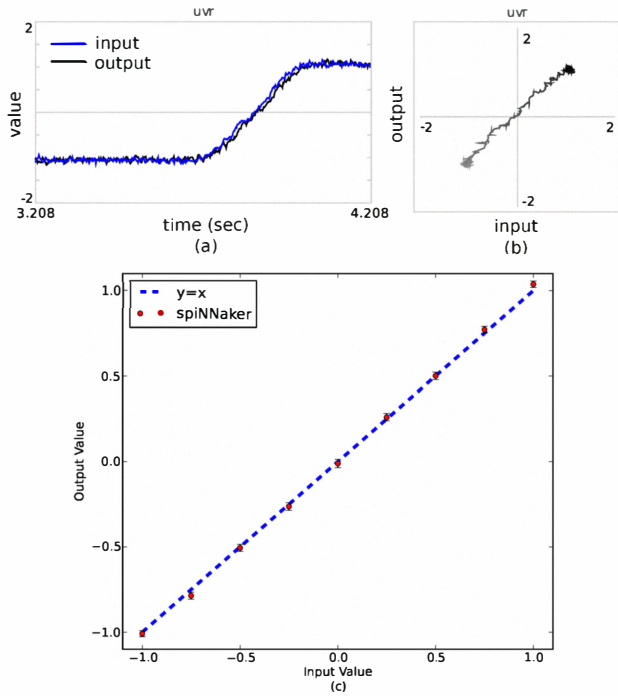


Fig. 4: **Representation Principle: Communication channel:** weights between A and B are set so that B represents the same value as A, representing the function  $Y = X$ . The value is then decoded by population OUT (decoding population). The value represented by A is decoded by decoding population IN to verify correct encoding. (b) Input value plotted (blue) against output (black) value using Nengo, showing the linear relation (c) Precision evaluation of the communication channel in the range of interest. Each value is sampled for 1 second and activity is averaged and standard deviation is showed.

## V. RESULTS

### A. Representation: Communication Channel

In order to test the representation of values in spike trains, we have implemented a communication channel with the structure illustrated in figure 3. The communication channel experiment shows how information can be represented using the NEF, as to be able to encode/decode information directly within the SpiNNaker System. This is done by a population which encodes a scalar value into neural activity and then decodes it with another population, equipped with *representational decoders* able to extract the original value. Such encoders/decoders are used to compute the function  $f(x)=x$  in the connection weights between the two populations (see fig. 2).

Population A comprises 150 *LIF-encoder* neurons. Population B is a standard LIF population comprises 150 neurons. *IN* and *OUT* are populations of 150 NEF-Decoder neurons each. Therefore the whole communication channel is then composed of 450 neurons firing in the 40-100 Hz range. Each Population lives on a dedicated core within the same chip. A value X is encoded by population A, the NEF-LIF encoder population.

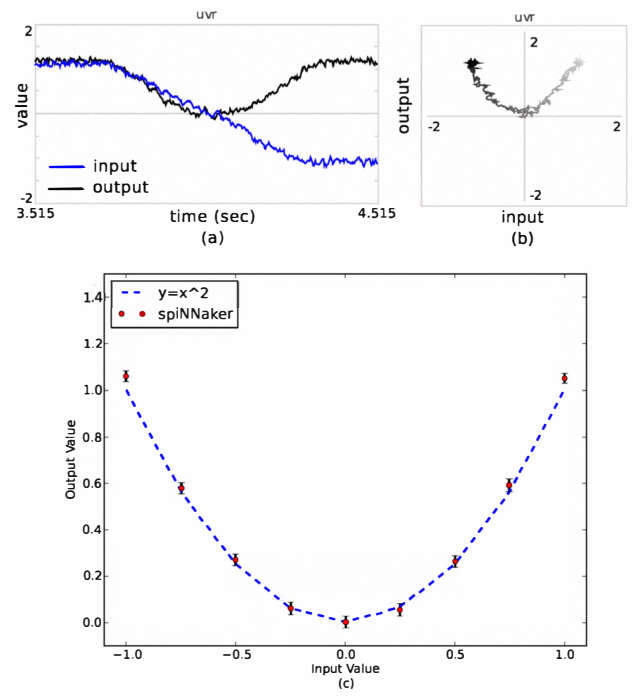


Fig. 5: **Transformation Principle: Computing the Square** (a) The blue line represents the direct decoding from the input sent to SpiNNaker and represents the ability of SpiNNaker to encode and decode input. The black line is the decoded result of the square operation implemented in the weights from A to B (b) The quadratic relation can also be observed when the input is plotted against the output within Nengo (c) Precision evaluation of the square computation in the range of interest. Each value is sampled for 1 second and activity is averaged and standard deviation is showed.

This population encodes values in spike trains according to the tuning curves of its neurons. Spikes travel to population B through an all to all connection with weights implementing the communication channel function  $y=x$ . Spike trains are then passed to the OUT population that converts them back into a value Y and outputs it to the external world. In particular it is possible to integrate SpiNNaker as a back-end simulator in Nengo by communicating with ethernet attached chips. Results are shown in figure 4 where precision of the encoding is evaluated and integration with Nengo is shown. All the experiments presented run in real time.

### B. Transformation: Computing the Square

In order to show computation we implemented the function  $y = x^2$  in the NEF, using the same network structure as that for the communication channel. Only the decoders of (and subsequently the connection weights to) population B are changed, implementing the function to be computed. Such decoders are called *transformational decoders* in the sense that they compute a (feedforward) transformation, as opposed to the *representational decoders* used for the communication channel experiment.

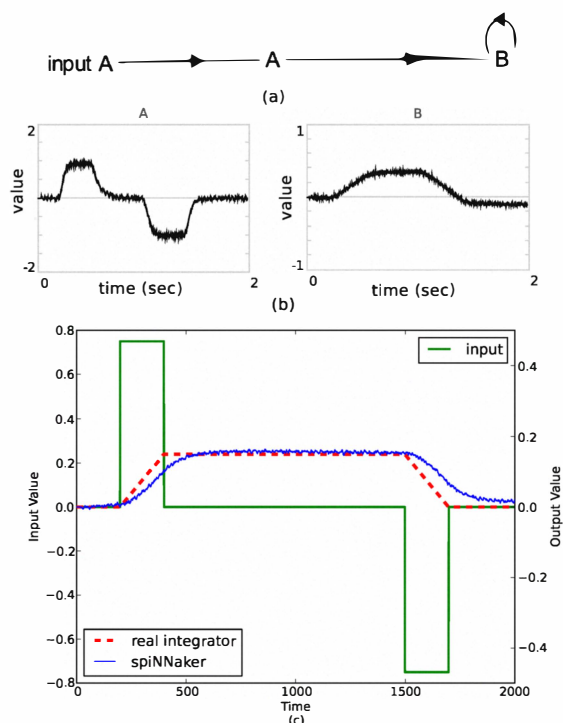


Fig. 6: **Dynamics: Neural Integrator** (a) Structure of the integrator network: An input is fed into population A, which travels through a communication channel to population B. Population B then computes the integral of the input received by A by means of recurring connection. (b) Integration in Nengo (c) Integration of the input value compared to an ideal integrator.

Results are shown in figure 5: the blue line represents the direct decoding from the input sent to SpiNNaker. The black line is the decoded result of the operation implemented in the weights from A to B. When the input is 0 (leftmost plot) the output from SpiNNaker is 0 as well (black line). When input is shifted to 1 both input (blue) and output go to 1. When input is shifted to -1 the result of squaring stays at 1. The quadratic relation is particularly evident when the input is plotted against the output, as done in figure 5(b) and 5(c). The network consists of 500 neurons as in the Communication Channel example (to be clear, it is the same network with the weights between A and B changed so as to compute the square, by estimating transformational decoders for B).

### C. Dynamics: Integrator

In order to show an implementation of neural dynamics within the NEF we implemented a neural integrator. Such a mechanism has been proposed as the neuronal basis of oculomotor control [12] where it is used as a velocity to position integrator: the input of the system represents the eye movement velocity and which is integrated to represent the final eye position. More generally the integrator can be considered a line attractor in the higher dimensional neural state space, letting the network maintain a value (representing

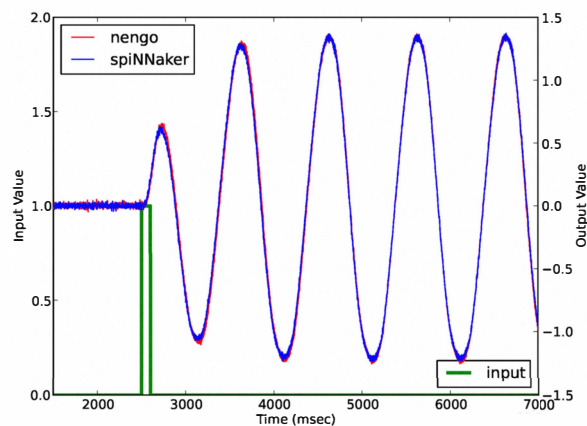


Fig. 7: **Cyclic Attractor: Oscillator** response of a neural oscillator to a perturbation; after the system is 'shocked' it starts oscillating at its characteristic frequency in Nengo and SpiNNaker. Only one dimension of the oscillating population is represented for clarity's sake.

for instance the eye position) through self-sustained activity in an abstract space over a period of time. The integrator has also been proposed to be the basis of working memory in neurons [30].

We can use the third principle as described in section II to translate the dynamics of an integrator defined by standard control theory input and dynamics matrices [10]. Using equations 3, by setting  $\mathbf{A} = 0$  and  $\mathbf{B} = 1$  (as in standard control theory  $\mathbf{A} = 0$  and  $\mathbf{B} = 1$  correspond to a linear attractor) we obtain  $\mathbf{A}' = 1$  and  $\mathbf{B}' = \tau$ , where  $\tau$  is the synaptic time constant. We can then compute the neural connection weights using eq. 4. The integrator structure is shown in figure 6: an input is fed into population A, and it travels through a communication channel to population B. Population B then computes the integral of the input received by A by means of recurrent connections. The Input population comprises 150 NEF-encoder neurons firing at 80-100 Hz. The integrator is composed of 200 neurons fully recurrently connected (40000 connections) firing at 80-150 Hz. Weights for this connections are computed using the encoders and decoders as described above. For simplicity, the weights are imported directly from Nengo and loaded on board. Population A is an encoding population as described in the section above.

Results from a simulation run, sending non-encoded input values to SpiNNaker and getting decoded output back is displayed in figure 6: population B integrates the positive pulse represented by input population A and holds the integrated value. Then a negative pulse input is received and integrated. Between the two pulses the integrator is able to hold the value with little drifting. The difference in the response of the neural integrator to the ideal one is due to the fact that the neural integrator has a PSC filter applied.

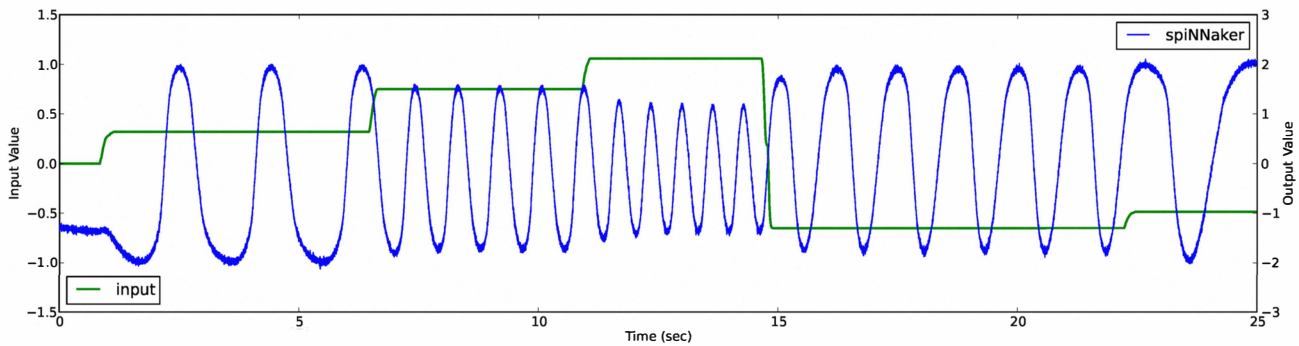


Fig. 8: **Non-linear system: Frequency Controlled Oscillator.** The oscillator has its orbit period (frequency of the oscillation) controlled by another input neural population. Only one dimension of the oscillating population is represented.

#### D. Cyclic attractors: Oscillator

All the models presented so far use a scalar representation for encoders, decoders and transformation. It is possible to extend the representation to vectors in an  $n$ -dimensional space by choosing neurons with preferred direction vector in the space, and hence employ  $n$ -dimensional encoders and decoders and transformational matrices for dynamical systems. It is then possible to evaluate neurobiological evidence to choose the appropriate kind of representation. By doing this it is possible to build another class of attractors: cyclic attractors. Rather than stabilising on a fixed point, line or plane, cyclic attractors settle on a periodic pattern of activity being dynamically stable. Such attractors can be used to explain repetitive behaviours like walking, flying, chewing or swimming; in particular a (more complex) cyclic attractor built accordingly to the NEF principles has been used as the basis to model swimming behaviour in the lamprey eel [10]. A cyclic attractor can be defined by the control equation  $\dot{x} = Ax + Bu$  where

$$A = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix}$$

implementing a cyclic attractor with an harmonic oscillator. The results are shown in figure 7: after the system is 'shocked' it starts oscillating at its characteristic frequency  $\omega$ .

Using the framework it is possible to manipulate the parameters that control attractor properties by modifying the matrix  $M^{\alpha\beta}$  using a control signal represented by another population: if the signal is a function of time the system described is linear time-varying system; if  $A$  is an input then the system becomes non linear. In this example we use a population to control the frequency of the oscillator, therefore controlling the speed of the cyclic attractor (eg. controlling the swimming speed in the zebra fish [21]) by increasing the dimensionality of the space encoded by the population to accommodate for a frequency control input. Results are shown in figure 8. In both experiments inputs and oscillating populations are composed by 150 neurons each firing in the 80-120 Hz range.

## VI. DISCUSSION

The work presented in this paper constitutes the basis for building real-time, large scale neural systems using the Neural

Engineering Framework on SpiNNaker. The programmability of the ARM cores makes the integration of SpiNNaker with the existing tools and software possible in a seamless way. It needs to be considered that the advantages of running large-scale models in real-time are strongly reduced if such models take a long time to be compiled and loaded on a computational back-end. In fact in the experiments have shown that the most computationally expensive task is to compute the weight connection matrices (described by eq. 4) and map them on a parallel system such as SpiNNaker. However it is possible to exploit the possibility to program SpiNNaker cores to parallelize this job and run it on board. Once the neural populations are mapped to specific cores, connections can be generated by sending them just encoders, decoders and transformational matrices, having each core doing the matrix multiplications, self-configuring and indexing its own local connections. We have shown how to map multidimensional encoders and decoders in the weight space so as to represent transformations in the neural space; it is however possible to implement  $n$ -dimensional *NEF/LIF-encoder* and *NEF-decoder* populations. Such high dimensional inputs can be used to manipulate complex, symbol-like structures such as language [31]. Such high dimensional spaces can be mapped to large scale populations of neurons so as to represent a large variety of symbols, having each neuron mapping a fraction of the high dimensional space. Running such models in real time makes possible to test them and embed them in real world, interactive scenarios, like decision making [33] or other cognitive functions as working memory, recognition and fluid reasoning [9].

## VII. CONCLUSIONS

We have successfully constructed neural circuits using the Neural Engineering Framework on the SpiNNaker hardware. We were able to encode and decode values using the NEF directly on board and implemented feed forward and recurrent dynamic computations. This approach takes advantages of the programmability of the ARM968 cores inside a SpiNNaker chip, letting it encode and decode spikes onboard. This reduces the bandwidth and the computational load needed on a

host machine, by sending and receiving only values to/from SpiNNaker and using it as a fast, scalable, configurable and power efficient computational back-end. This approach offers advantages when firing rates and dimension of input and output neurons increase, letting the system scale up seamlessly and be integrated in interactive real-time systems. It also presents the basis for building and testing large-scale neural models built with the Neural Engineering Framework on the SpiNNaker architecture.

#### ACKNOWLEDGMENTS

The work presented in this paper is largely based on experiments that were carried out at the 2011 Workshop on Neuromorphic Engineering in Telluride<sup>2</sup>. The authors would like to thank the organizers and the sponsors.

The SpiNNaker project is supported by the Engineering and Physical Science Research Council (EPSRC), grant EP/4015740/1, and also by ARM and Silistix. We appreciate the support of these sponsors and industrial partners.

#### REFERENCES

- [1] D.J. Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge Univ Pr, 1992.
- [2] Rajagopal Ananthanarayanan, Steven K Esser, Horst D Simon, and Dharmendra S Modha. The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*, pages 63:1—63:12, New York, NY, USA, 2009. ACM.
- [3] M.A. Bhuiyan, V.K. Pallipuram, and M.C. Smith. Acceleration of spiking neural networks in emerging multi-core and GPU architectures. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.
- [4] Tom Binzegger, Rodney J Douglas, and Kevan A C Martin. A quantitative map of the circuit of cat primary visual cortex. *J Neurosci*, 24(39):8441–8453, September 2004.
- [5] A. Cassidy, A.G. Andreou, and J. Georgiou. Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis. In *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*, pages 1–6. IEEE, 2011.
- [6] S Celebrini, S Thorpe, Y Trotter, and M Imbert. Dynamics of orientation coding in area V1 of the awake primate. *Vis Neurosci*, 10(5):811–825, 1993.
- [7] S Dehaene, M Kerszberg, and J P Changeux. A neuronal model of a global workspace in effortful cognitive tasks. *Proceedings of the National Academy of Sciences of the United States of America*, 95(24):14529–14534, 1998.
- [8] C Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural computation*, 7(6):1276–1314, 2005.
- [9] C Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University (in press), 2012.
- [10] C Eliasmith and C H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, Cambridge, MA, 2003.
- [11] Chris Eliasmith, M Brandon Westover, and Charles H Anderson. A general framework for neurobiological modeling: an application to the vestibular system. *Neurocomputing*, pages 1071–1076, 2002.
- [12] K. Fukushima, CR Kaneko, A.F. Fuchs, and Others. The neuronal substrate of integration in the oculomotor system. *Progress in neurobiology*, 39(6):609, 1992.
- [13] S B Furber, S Temple, and A D Brown. High-Performance Computing for Systems of Spiking Neurons. *The AISB06 workshop on GC5: Architecture of Brain and Mind*, 2006.
- [14] S B Furber, S Temple, and A D Brown. On-chip and Inter-Chip Networks for Modelling Large-Scale Neural Systems, 2006.
- [15] Stephen Furber and Andrew Brown. Biologically-Inspired Massively-Parallel Architectures - Computing Beyond a Million Processors. In *Proceedings of the 2009 Ninth International Conference on Application of Concurrency to System Design, ACS'D '09*, pages 3–12, Washington, DC, USA, 2009. IEEE Computer Society.
- [16] F. Galluppi, S. Davies, A. D. Rast, T. Sharp, L.A. Plana, and S. Furber. A Hierarchical Configuration System for a Massively Parallel Neural Hardware Platform. In *Proceedings of the 9th ACM international conference on Computing frontiers*. ACM, 2012.
- [17] K M Hynna and K Boahen. Neuronal Ion-Channel Dynamics in Silicon. In *Proc. 2006 Int'l Symp. Circuits and Systems (ISCAS 2006)*, pages 3614–3617, 2006.
- [18] G Indiveri, E Chicca, and R Douglas. A VLSI array of low-power spiking neurons and bistable synapses with spiketime dependent plasticity. *IEEE Transactions on Neural Networks*, 17:211–221, 2006.
- [19] E M Izhikevich and G M Edelman. Large-scale model of mammalian thalamocortical systems. *Proc. National Academy of Sciences of the USA*, 105(9):3593–3598, March 2008.
- [20] Dethier J, Gilja V, Nuyujukian P, Ellassaad S, Shenoy KV, and Boahen K. Spiking neural network decoder for brain-machine interfaces. In *Proc. of the 5th International IEEE EMBS Conference on Neural Engineering, Cancun, Mexico*, pages 396–399, 2011.
- [21] P Dwight Kuo and Chris Eliasmith. Integrating behavioral and neural data in a model of zebrafish network interaction. *Biological Cybernetics*, 93(3):178–187, 2005.
- [22] J Lazzaro, J Wawrzynek, M Mahowald, M Silviotti, and D Gillespie. Silicon Auditory Processors as Computer Peripherals. *IEEE Transactions on Neural Networks*, 4(3):523–528, May 1993.
- [23] F.L. Lewis and EW Kamen. Applied optimal control and estimation. *IEEE Transactions on Automatic Control*, 39(8):1773–1773, 1994.
- [24] L P 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*, 71, December 2007.
- [25] J M Nageswaran, N Dutt, J L Krichmar, and A Nicolau. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5–6), 2007.
- [26] L Plana, S Furber, S Temple, M Khan, Y Shi, J Wu, and S Yang. A GALS Infrastructure for a Massively Parallel Multiprocessor. *IEEE Design & Test of Computers*, 24(5):454–463, 2007.
- [27] Alexander Rast, Francesco Galluppi, Sergio Davies, Luis Plana, Cameron Patterson, Thomas Sharp, David Lester, and Steve Furber. Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware. *Neural Networks*, (0):In Press, 2011.
- [28] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proc. 2010 Int'l Symp. Circuits and Systems (ISCAS 2010)*, pages 1947–1950, 2010.
- [29] Erik De Schutter. *Computational Modeling Methods for Neuroscientists*. The MIT Press, 1st edition, 2009.
- [30] Ray Singh and Chris Eliasmith. Higher-dimensional neurons explain the tuning and dynamics of working memory cells. *Journal of Neuroscience*, 26(14):3667–3678, 2006.
- [31] TC Stewart, T Bekolay, and C Eliasmith. Neural representations of compositional structures: representing and manipulating vector spaces with spiking neurons. *Connection Science*, 23(2):145–153, 2011.
- [32] TC Stewart, C Eliasmith, and L Carlson. Neural Cognitive Modelling: A Biologically Constrained Spiking Neuron model of the Tower of Hanoi Task. In *33rd Annual Conference of the Cognitive Science Society*, Austin, TX, USA, 2011.
- [33] Terrence C Stewart and Chris Eliasmith. Neural Symbolic Decision Making: A Scalable and Realistic Foundation for Cognitive Architectures. In *BICA'10*, pages 147–152, 2010.
- [34] Terrence C Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the Nengo simulator. *Frontiers in Neuroinformatics*, 3(0), 2009.
- [35] AM Thomson and C Lamy. Functional maps of neocortical local circuitry. *Frontiers in neuroscience*, 2007.
- [36] J H B Wijekoon and P Dudek. Integrated Circuit Implementation of a Cortical Neuron. In *Proc. 2008 Int'l Symp. Circuits and Systems (ISCAS 2008)*, pages 1784–1787, 2008.
- [37] T.J. Wills, C. Lever, F. Cacucci, N. Burgess, and J. O'Keefe. Attractor dynamics in the hippocampal representation of the local environment. *Science*, 308(5723):873, 2005.

<sup>2</sup><https://neuromorphs.net/nm/wiki/ng11/results/Spinnaker>