Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2013 Special Issue

Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization

Andrew S. Cassidy[a,1], Julius Georgiou[b], Andreas G. Andreou[a,b,*]

[a] Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218, USA
[b] Department of Electrical and Computer Engineering, University of Cyprus, Nicosia 1678, Cyprus

ARTICLE INFO

Keywords:
Silicon brains
Neuromorphic engineering
Silicon neurons
Learning in silicon
FPGA neural arrays

ABSTRACT

We present a design framework for neuromorphic architectures in the nano-CMOS era. Our approach to the design of spiking neurons and STDP learning circuits relies on parallel computational structures where neurons are abstracted as digital arithmetic logic units and communication processors. Using this approach, we have developed arrays of silicon neurons that scale to millions of neurons in a single state-of-the-art Field Programmable Gate Array (FPGA). We demonstrate the validity of the design methodology through the implementation of cortical development in a circuit of spiking neurons, STDP synapses, and neural architecture optimization.

1. The computer and the brain

The brain is a massively parallel and efficient information processing system, with a radically different computational architecture from present day computers. Characteristics of neural computation include event based processing, fine-grained parallel computational units, robustness and redundancy, as well as adaptation and learning, all done under severe constraints of size, weight, and energy resources. This computational architecture excels at lower-level sensory information processing such as vision, and sensor–motor integration as well as cognitive tasks such as speech and language understanding.

Over the last half century computer scientists, architects and engineers have envisioned building computers that match the parallel processing capabilities of biological brains. Fifty years ago, the fathers of computer science Alan Turing (Turing, 1952) and John von-Neumann (Neumann, 1958) looked to the brain for inspiration in order to advance the science of computing.

Twenty-five years ago, the connectionist movement emerged as an alternative approach to artificial intelligence for solving the hard problems in perception and cognition. The central doctrine in the connectionist movement is that the cognitive abilities of the brain are a result of a highly interconnected network of simple processing units. These simple non-linear computational units abstract the function of neurons while synapses abstract the connections between neurons. The strength of the synaptic connections in networks of such units is determined through a learning algorithm. A two volume edited book-set by the "Parallel Distributed Research Group" (McClelland, Rumelhardt, & Group, 1987; Rumelhart, McClelland, & Group, 1987) defined the research agenda in the field of connectionist architectures and neural networks in the decades that followed. At about the same time, Carver Mead's book "Analog VLSI and Neural Systems" (Mead, 1989) inspired a new generation of scientists and engineers to explore hardware implementation of neural models in state-of-the-art silicon integrated circuit technology. The book had a dual objective: (i) to create a new design discipline for collective computational systems using analog VLSI subthreshold CMOS integrated circuit technology and (ii) to promote a synthetic approach in the understanding of biology and the human brain. This was the birth of neuromorphic design as an engineering discipline.

1.1. *Neuromorphic engineering: the formative years*

"Neuromorphic" electronic systems, a term coined by Carver Mead in the late 1980s, describes systems that perform artificial computation based on the principles of neurobiological circuits. In the following two decades, inspired by Mead's pioneering work (Mead, 1990) and colleagues at Caltech, a large number of CMOS neuromorphic chip designs have been reported in the literature.

These spanned a wide range of designs from analog VLSI models of neurons (Arthur & Boahen, 2010; Hsin, Saighi, Buhry, & Renaud, 2010; Saighi, Bornat, Tomas, Le Masson, & Renaud, 2010; Yu, Sejnowski, & Cauwenberghs, 2011) to silicon retina architectures

* Corresponding author at: Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218, USA.
E-mail addresses: andrewca@us.ibm.com (A.S. Cassidy), julio@ucy.ac.cy (J. Georgiou), andreou@jhu.edu (A.G. Andreou).

[1] Now with IBM Research, Almaden, USA.

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.neunet.2013.05.011

(Boahen & Andreou, 1992; Mahowald, 1992), and retinomorphic vision systems (Boahen, 1996), to attention circuits (Horiuchi & Koch, 1999), and biomorphic imagers (Culurciello, Etienne-Cummings, & Boahen, 2003) that abstract biology at a lower level. Other mixed-mode designs (Andreou, Meitzler, Strohben, & Boahen, 1995; Pardo, Dierickx, & Scheffer, 1998) and (Etienne-Cummings, Kalayjian, & Donghui, 2001) have also implemented silicon retinas and focal plane processing architectures that include processing beyond gain control and spatio-temporal filtering, including polarization sensing (Andreou & Kalayjian, 2002; Wolff & Andreou, 1995). Most of the above bio-inspired sensors have limited programmability as they employ analog computational circuits at the focal plane.

The shortcomings of non-programmable analog architectures motivated the exploration of analog vision chip architectures with programmable functionality (Serrano-Gotarredona, Andreou, & Linares-Barranco, 1999; Serrano-Gotarredona et al., 2009). Programmable architectures for associative memory (Boahen, Pouliquen, Andreou, & Jenkins, 1989; Pouliquen, Andreou, & Strohben, 1997), pattern classification (Genov & Cauwenberghs, 2001; Karakiewicz, Genov, & Cauwenberghs, 2007) and audition (Kumar, Himmelbauer, Cauwenberghs, & Andreou, 1998; Stanacevic & Cauwenberghs, 2005) have also been reported in the literature.

Programmable architectures have also been advanced by the adoption of a standard interface between chips known as Address Event Representation or (AER) in short. The time-multiplexed AER bus (Boahen, 2000; Lin & Boahen, 2009; Mahowald, 1992; Sivilotti, 1991) is a popular interconnect method for neuromorphic systems. Spike events from multiple channels are time-multiplexed onto a digital AER bus, transmitted, and decoded at the destination onto individual channels. Throughout this proposal, we use the terms spikes, events, and spike events interchangeably. AER has been used by many analog and digital spiking neural arrays, as well as to communicate events from off-chip neuromorphic sensors and even in 3D CMOS technology (Harrison, Özgün, Lin, Andreou, & Etienne-Cummings, 2010). The European Union project CAVIAR (http://www2.imse-cnm.csic.es/caviar/) demonstrated a board-level vision system architecture communicating using the AER protocol (Serrano-Gotarredona et al., 2009). Variants of AER to improve the efficiency of the protocol have also been proposed (Georgiou & Andreou, 2006, 2007). A probabilistic approach to AER has been exploited to perform computations in the address domain (Goldberg, Cauwenberghs, & Andreou, 2001b).

Learning in silicon has also been pursued intensively in the analog VLSI neuromorphic community. The early work by Diorio and colleagues (Diorio, Hasler, Minch, & Mead, 1996, 1997), the Field Programmable Analog Arrays (Sivilotti, 1991) and the research program of Hasler (Hall, Twigg, Gray, Hasler, & Anderson, 2005) paved the way to floating gate MOS transistors in configurable learning chips. Other designs employ dynamic circuits for implementing learning in analog VLSI with excellent results on small systems (Bartolozzi & Indiveri, 2007; Indiveri, Chicca, & Douglas, 2004; Mahowald, 1992). This work has continued with encouraging results for hardware models that abstract higher-level functions such as stimulus specific adaptation (Mill, Sheik, Indiveri, & Denham, 2011) and working memory using attractor dynamics (Giulioni et al., 2011).

Abstracting biology at a higher level, the Cellular Non-linear/Neural Networks (CNN) approach (Chua & Yang, 1988) offered another paradigm for an analog visual processor with programming capabilities. In CNN architectures, information processing is implemented through the evolution of a continuous-time non-linear dynamical network with nearest neighborhood connectivity. The CNN–UM (Universal Machine) is one of the earliest systems (Roska & Chua, 1993) that implemented CNN programmable functionality on a chip. Another example of CNN hardware implementation merges a CNN–UM type processor and an imager (Carmona et al., 1998; Dominguez-Castro et al., 1997). This system, while analog internally, has a digital interface with on-chip 7-bit A/D and D/A converters, improving the programmability and simplifying the interface to digital computers (Cembrano et al., 2004).

Programmable analog VLSI circuits and systems aimed at large-scale model simulation have also been under development in the last decade. The Neurogrid architecture in Kwabena's group (Arthur & Boahen, 2010; Choudhary et al., 2012; Silver, Boahen, Grillner, Kopell, & Olsen, 2007), the IFAT architecture (Goldberg, Cauwenberghs, & Andreou, 2001a; Vogelstein, Mallik, Culurciello, Cauwenberghs, & Etienne-Cummings, 2007), the PAX platform (Renaud et al., 2010) and the FACETS wafer-scale computational infrastructure (Bruederle et al., 2011) are notable projects in this direction.

1.2. Neuromorphic engineering: the nano-CMOS Era

In 1986, Mead's group at Caltech was employing bulk CMOS technology with λ between 2.5 micron and 0.7 micron (p. 59 of Mead, 1989). A quick review of our own publications and laboratory notebooks from that period, reveals that we were fabricating chips in 4 micron Silicon On Sapphire (SOS)–CMOS technology and in 3 micron *p*-well bulk CMOS. Alas! Twenty five years later, with foundry CMOS technologies at the 45 nm and 22 nm nodes, the neuromorphic engineering community at large has not been able to capitalize on the benefits of the ($\times$10 000) improvements in digital MOS transistor area density to engineer brain like structures and cognitive machines that match the effectiveness and energetic efficiency of the human brain. With the exception of the event-based, asynchronous vision sensors (Lichtsteiner, Posch, & Delbruck, 2008) and subsequent design (Posch, Matolin, & Wohlgenannt, 2011), the goals of endowing modern computer systems with industrial-strength robust bio-inspired sensoria or tackling the challenge of silicon cognition have been unrealized. And even though our lack of knowledge about the inner workings of brain function and behavior has contributed to this chasm and is limiting us today, matching the information processing capabilities of biological neural structures in state-of-the-art silicon technology has remained an elusive dream despite the stunning advances in microelectronics.

Even more elusive has been our quest to understand how to achieve the energy efficiency seen in biological brains. One would have thought that the research activities in the last two decades would have brought us closer to both a deeper understanding of brain function as well as to commercially-viable brain-inspired information technology at the scale. However, this is not the case. Many of the analog VLSI neuromorphic systems rely on analog devices and as such, scaling the density of these components (mostly MOS transistors and capacitors) did not follow Moore's law. Furthermore, the majority of neuromorphic hardware was based on traditional "analog" circuit models of neurons and synapses, a technology that does not offer flexibility in component models, nor in their level of description; an aspect which impedes rapid advances.

Mead advocated using analog transistor physics to perform neural computation, directly mimicking the currents in neuron ion channels (Mead, 1990), and speculated that an energy savings of approximately 10^4 could be gained over comparable traditional digital approaches. However the power dissipation of neuromorphic systems did not benefit from technology scaling either and our best circuits today hover between 10 and 100 nW per computational cell. Each cell has typically one or two single pole circuits with two or three current branches biased in the nano-ampere current level. Even though one could argue the power dissipation is manageable locally, the energy cost to send the digital representation of the state from one chip to another on the same board is high. It takes less than a femto-Joule of energy to move one bit worth of charge through the source to the drain of an MOS transistor in deep sub-micron CMOS technology, one pico-Joule to move one bit of information across a 1 cm die, and almost one hundred pico-Joules to move it from one die to another! (Cassidy & Andreou, 2012). This is a poor utilization of energy and a direct result of the limitations of two dimensional integration and the use of macroscopic components to interconnect chips. Optical interconnects, while efficient at distances measured in kilometers, have not been very helpful at short distances from a power dissipation point of view; this may change in the years to come with advances in silicon photonics and CMOS integration. In a two dimensional array of cells that is typical in neuromorphic electronic systems (feature maps), additional energy costs are accrued in going from the two dimensional representation of data to a one dimensional stream on the periphery of the die for inter-chip transmission.

Without exception, all of the analog VLSI systems reported in the literature, small or large, rely on substantial digital communication infrastructures for functionality, ranging from single chip micro-controllers (Goldberg et al., 2001a,b), to arrays of FPGAs (Bruederle et al., 2011; Choudhary et al., 2012; Renaud et al., 2010; Vogelstein et al., 2007) to even single board computers (Fasnacht & Indiveri, 2011).

Unlike biological nervous systems, constrained by the limitations of 2D CMOS technology, networks of electronic components such as switches, capacitors and short wires in VLSI integrated circuits are only weakly connected to each other i.e. each component is connected to only a few other components, often in a pipeline structure. Architectures of modern processors and memories are designed specifically to meet the constraints of limited connectivity in modern two dimensional integrated circuits, which feature a dozen or so metallization layers.

We are now at a juncture point: an era where digital transistors are nearly "free", and billion-transistor integrated Systems on a Chip (SoC) are now commonplace. Capitalizing on the dramatic advances in the scaling of the MOS transistor, analog silicon retinas (Boahen & Andreou, 1992) and mixed-signal sensors (Lichtsteiner et al., 2008; Posch et al., 2011) are being displaced by all digital architectures. Sophisticated digital sensor arrays and computational sensors are currently being developed to extract and quantify the subtle and intricate information from natural scenes in visible and infra-red wavelengths (Lin, Pouliquen, Andreou, Goldberg, & Rizk, 2012; Lin, Pouliquen, Goldberg, Rizk, & Andreou, 2011). Cellular neural network architectures and computational imagers have been reported in standard digital CMOS (Federico, Mandolesi, Julian, & Andreou, 2008; Mandolesi, Julian, & Andreou, 2004) as well as experimental 3D CMOS technologies (Mandolesi, Julian, & Andreou, 2006). Stream processor architectures for convolutional neural networks (ConvNets) have also been recently reported in the literature (Camuñas-Mesa et al., 2012; Pham et al., 2012). Digital FPGA based bio-inspired architectures have also been reported in the literature, for example recent work by Kestur et al., and references therein (Kestur et al., 2012).

In this nano-CMOS era, the engineering of large-scale neuromorphic systems aimed at silicon cognition must also be carried on at a different level of abstraction. Instead of using analog transistors to emulate the biophysics of neurons, we must move to a higher level of abstraction, using digital transistors to perform the arithmetic equivalent to the behavior of a neuron. Combined with high-density digital memories and high-speed digital communications interconnects, this paradigm will enable the implementation of large-scale, flexible silicon neural arrays.

State-of-the-art Field Programmable Gate Arrays (FPGAs) are often at the forefront of technological advances and the ability to rapidly prototype digital systems makes them an attractive platform for bio-inspired systems exploration in the nano-CMOS era. The design of small systems and applications of digital Spiking Neural Networks (SNNs) have already been reported in FPGAs (Belhadj, Tomas, & Bornat, 2009; Cardoso, Diniz, & Weinhardt, 2010; Cassidy, Denham, Kanold, & Andreou, 2007; Koickal, Gouveia, & Hamilton, 2009; La Rosa et al., 2005; Pearson et al., 2007; Rice, Bhuiyan, Taha, & Smith, 2009).

Research towards the engineering of custom large-scale digital bio-inspired integrated circuits has also begun with encouraging results. SpiNNaker is a System on a Chip (SoC), a massively-parallel digital neuromorphic computing architecture (Khan et al., 2008) based on an 18 core symmetric chip-multiprocessor where each core is an ARM968. A SpiNNaker computer will consist of a million microprocessor cores interconnected via a switching network fabric. APIs and software have already been developed for the SpiNNaker system (APT-Group, 2011a,b,c), hence an excellent platform to explore bio-inspired algorithms and architectures for cognitive computing. Another digital bio-inspired system architecture for energy-aware cognitive computing is currently being developed by IBM under the SyNAPSE project (Arthur et al., 2012; Merolla et al., 2011; Modha et al., 2011).

Complementary to advances in large-scale hardware architectures have been the advances in large-scale software simulation and modeling environments such as Brian (Goodman & Brette, 2009), Nengo (Anderson & Eliasmith, 2004; Eliasmith & Stewart, 2011; Eliasmith et al., 2012), and Compass (Modha et al., 2011; Preissl et al., 2012). The close coupling of these software environments to the SpiNNaker architecture (Galluppi, Davies, Furber, Stewart, & Eliasmith, 2012), Neurogrid (Choudhary et al., 2012), FACETS wafer-scale system (Bruederle et al., 2011), and SyNAPSE project (Modha et al., 2011) is likely to facilitate the widespread acceptance of custom architectures for large-scale simulations (Silver et al., 2007) as an alternative to high performance computing (de Garis, Shuo, Goertzel, & Ruiting, 2010; Markram, 2011).

In this paper we address the challenging task of engineering silicon brains in the nano-CMOS era, with billions of transistors and thousands of processors on a chip. We take a systematic approach, revisiting Marr's three levels of description and argue that the computational theory of parallel processing under physical constraints provides the theoretical underpinnings for both understanding the brain and a new era of neuromorphic engineering. We present a scalable neural architecture with computationally efficient structures for spiking silicon neurons and learning synapses. Finally we employ a cost function based formulation of parallel processing under the physical constraints of speed, energy, and area to optimize the neural architecture. Experimental verification of the computational structures for the optimization of the neural array, spiking neurons and Spike Timing Dependent Plasticity (STDP) learning is done using state-of-the-art field programmable gate arrays (FPGAs).[2]

2. Revisiting Marr's vision: employing new eyes

In the previous section, we have argued that our progress towards *synthetic brain* like systems with cognitive capability matching human performance and *energetic efficiency* is seriously hampered by difficulties in expressing brain-related functional system-level models and algorithmic constructs at an appropriate

[2] The Introduction in this paper is not meant to be a comprehensive review of hardware implementations of neural systems over the last quarter century. The interested reader is referred to a recent survey for a comprehensive overview (Misra & Saha, 2010).

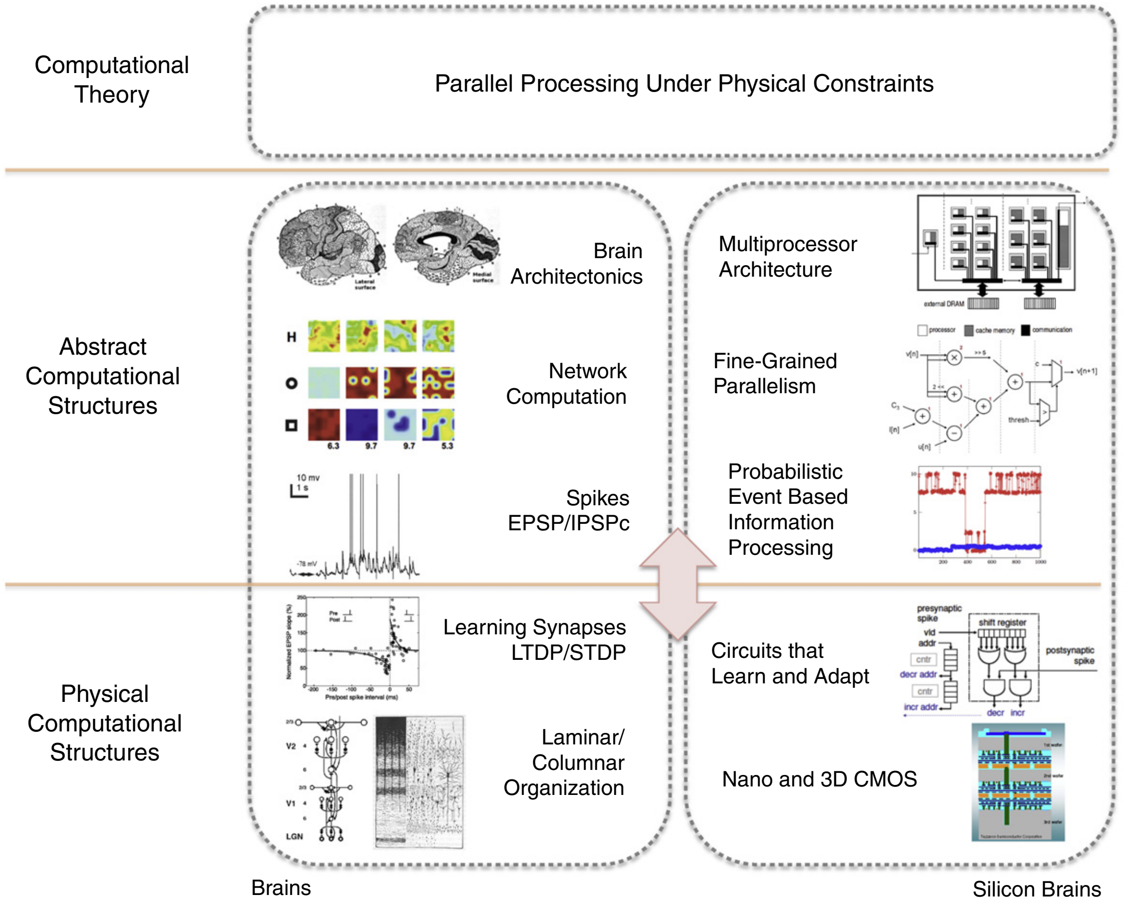

Computational Theory
Parallel Processing Under Physical Constraints
Abstract Computational Structures
Brain Architectonics
Network Computation
Spikes EPSP/IPSPc
Multiprocessor Architecture
Fine-Grained Parallelism
Probabilistic Event Based Information Processing
Physical Computational Structures
Learning Synapses LTDP/STDP
Laminar/ Columnar Organization
Circuits that Learn and Adapt
Nano and 3D CMOS
Brains
Silicon Brains

Fig. 1. Biological and silicon brains: layered levels of abstraction and information processing structures. Note that even though the abstract and physical layers are shown in a hierarchy, they are done so only for the sake of clarity. In particular, the layered description for the physical computational structures in the brain are simplified in this diagram. Churchland and Sejnowski offer a more accurate diagram for layers for the different physical scales in their seminal paper (Churchland & Sejnowski, 1988).

level of abstraction that can hide the details of their implementation. What we need is a multi-scale framework where information processing structures at different layers in the two levels of abstraction have different interpretations of the processing element's type, capability, and complexity. Any approach that relies on a single monolithic level of abstraction is bound to be slave to the underlying limitations imposed by that level. For example, on analog neuromorphic chips, models of higher-level cognitive function can only be expressed in terms of networks of analog neurons, with any other level of description is required to be implemented ad hoc on the host digital computer or a micro-controller. Thus the prevailing research approaches in neuromorphic engineering offer neither an effective basis for exploring system-level models of brain function, nor a practical foundation for future brain-inspired cognitive computing technology.

To address the challenges of engineering large-scale silicon brains, we need a fresh perspective, a new view, one that systematically allows to abstract brain computation into synthetic structures. Quoting Marcel Proust, *"Fundamental discoveries do not necessarily rely on exploring new landscapes, but on employing new eyes"*.

Conventional wisdom and the prevailing viewpoint in the scientific community suggest that complex information processing systems, natural or synthetic, are best considered at multiple levels of organization. In the broadly defined task of "engineering silicon brains", we must have a consistent framework to address scientific challenges and exploit technological opportunities at the different levels of description (see Fig. 1). David Marr in the preface of his seminal book "Vision: A Computational Investigation into the Human Representation and Processing of Visual Information" proposes a framework that has three levels of description to help us understand visual processing in biological systems and to engineer machines that see.

The three levels of organization as proposed by Marr are:

- *Computational theory:* What is the goal of the computation and the logical strategy needed to carry it out?
- *Representation and algorithm:* How can the computation be implemented and what input/output representations are needed?
- *Hardware implementation:* What is the physical realization of the algorithm and the architecture?

We paraphrase Proust and argue that what we need is "new vision" and Marr's *vision* on levels of organization (Marr, 1982), provides the foundation for a fresh perspective on parallel processing, "the computer and the brain".

At the *level of computational theory*, we suggest that the general theme of *parallel processing under physical constraints* could provide the theoretical foundation for understanding information processing in both the brain and in massively parallel computing systems. Thus a principled approach that links the algorithmic aspects and costs of parallel processing to the constraints imposed by the physical hardware in the physical structure could provide a fundamental foundation for further understanding the brain and a powerful principle for engineering and architecting future brain-inspired computing machinery. This will be further elaborated in following sections of this paper (Sections 3 and 6).

At the *level of representation*, in neural computation, the temporal dynamics of spiking neurons encode information. Spikes in biology or digital events in silicon systems can encode graded (analog) signals in time while at the same time employing the robustness of binary signaling. At this level we design abstract computational structures optimized for minimum energy that exploit spike-event based representations to compute likelihoods in graphical probabilistic models of inference and cognition. Such a probabilistic event based approach that was first proposed and used in engineered systems in (Goldberg et al., 2001a,b) provides a principled description of event generation rules that maximize the information transfer, while limiting the number of energy expensive events (spikes) (Laughlin & Sejnowski, 2003; Schreiber, Machens, Herz, & Laughlin, 2002), that need to be communicated between successive layers in a neural architecture. Using the digital abstraction, neural computation can readily take on a variety of computational models including the Leaky Integrate and Fire and dynamical Izhikevich neural models, as elaborated in Section 4. In this work, models of learning take on the STDP representation, a Hebbian learning rule, described in Section 5.

At the *level of implementation,* the physics and chemistry imposed by the underlying substrate define the constraints for the micro-architectural elements, whether real neurons in biological tissue or silicon neurons in nano-CMOS technologies. It is at this level of description that the information processing elements of biological tissue are likely to differ substantially in form and function from those implemented in nano-CMOS technology. At this level, attempts to draw analogies in the physical implementations are at best superficial. So long as we are aware of the fundamental differences in the underlying substrates, we are assured to stay away from the perilous paths that impeded our progress towards the engineering of large-scale silicon brains and cognitive machines over the last two decades.

In Sections 4 and 5, we detail multiplexed digital circuits, storing the neural state in dense local memory and multiplexing neural computation on shared arithmetic logic units. This approach uses the high frequency (relative to biological computation) of silicon circuits to efficiently implement the massive parallelism required for implementing large-scale neural computational architectures. The proposed holistic approach towards the engineering of silicon brains forms the foundation for a new research direction in brain-inspired architectonics. It relies on a combination of multi-scale abstraction from algorithms to representation and the architecture. At each level, the computational structures rely on consistent contracts between the levels of abstraction and the layers of description.

3. Parallel processing under physical constraints

Biological information processing systems employ dynamic matter and learning at all levels in an amazing network of complex structures of different scales, from the nano to the micro and macro. In the human brain, a physical structure with approximately 100 billion neurons and 100 trillion synapses, parallel processing must be the norm rather than the exception. Indeed parallel distributed processing is found at all levels of brain function from molecules to networks to social behavior.

Some insights into the brain's organization and parallel processing function can be gained by considering the way the visual representation of the natural world is organized through cortical maps. Early work by Hubel and Wiesel, suggested stimulus modalities are mapped in orthogonal dimensions: the ice-cube model for stimulus representation in $V1$ (Hubel & Wiesel, 1977). However, this simple and elegant idea does not scale beyond two stimulus features and into natural visual environments that involve a plethora of stimulus features. What is needed is a process by means of which multiple features such as orientation, spatial frequency, ocular dominance, and so on, can be mapped into the two and delta (2D+delta) dimensional patches on the surface of the cortex ($V1$) (Das, 2000). An important insight into this challenging problem emerged out of experimental work in the visual cortex of the cat where it was shown that maps in the visual cortex are optimized for uniform coverage (Stepanyants, Hof, & Chklovskii, 2002; Swindale, Shoham, Grinvald, Bonhoeffer, & Hübener, 2000). Through a process of development and self-organization, brains find an optimal solution in the presence of two conflicting requirements: spatial coverage and stimulus representation. More specifically, to maximize coverage and parallel processing, every location in the physical space must be mapped to all possible combinations of stimulus features. At the same time, this must be done under constraints imposed by the physical structure; i.e. the wiring length of axons must be kept as short as possible to minimize both the metabolic costs (energy) and the time to respond (delay). The result is a smoothness of mapping or locality of reference; i.e. neurons with similar stimulus response properties lie in close proximity on the cortical surface, with local discontinuities arising from the multi-feature mapping. Similar self-organizing principles are found in the structural and morphological organization of neurons in the brain, taking into account both the functional and the underlying physical and chemical constraints of the computational machinery (Chklovskii, Mel, & Svoboda, 2004; Varshney, Sjöström, & Chklovskii, 2006; Wen & Chklovskii, 2008; Wen, Stepanyants, Elston, Grosberg, & Chklovskii, 2009).

3.1. From brain architectonics to silicon neural architectures

Parallel processing silicon neural arrays must be optimized with respect to two performance objectives: speed (inverse of delay) and energy. Motivated by the need for a design methodology to address architectural space exploration in multi-scale parallel architectures, we have derived a simple objective function (Cassidy & Andreou, 2012) to link parallel processing in an information processing system of N units, to the costs of energy and delay, the traditional metrics in integrated circuits (Mead & Conway, 1979). The foundation of our model is a cost function formulation of Amdahl's law (Amdahl, 1967), that employs parameterized models of computation and communication to represent the characteristics of processors, memories, and communication networks. The interaction of these micro-architectural elements within a parallel processing framework defines global system performance in terms of energy–delay cost.

Starting from Amdahl's original intuitive argument, we can derive a generalized cost function that combines a delay cost function and an energy cost function according to the energy–delay product, to obtain a generalized objective function J_{ED} that links the gains from parallel processing to delay and energy costs (Cassidy & Andreou, 2012).

The equation below is an extension of the work in Cassidy and Andreou (2012) to address parallel processing in an asymmetric multiprocessor architecture such as the one necessary for brain-inspired multiprocessor systems. Using a summation over the processor types p of different performance, the objective function J_{ED} can be written as:

$$J_{ED} = \left[\sum_{j=0}^{K-1} \frac{F_j}{\sum_{p=0}^{P-1} N_{jp} \left(\sum_{i=0}^{M-1} G_{ijp} D_{ijp} \right)^{-1}} \right] \times \left[\sum_{j=0}^{K-1} \frac{F_j}{N_{jA}} \sum_{h \in \{A,I\}} \sum_{p=0}^{P-1} N_{jhp} \sum_{i=0}^{M-1} G_{ijhp} E_{ijhp} \right]^{\gamma}. \quad (1)$$

In the inner summations, each of the algorithm fractions F_j, are subdivided into constituent cost components. G_{ijp} is the fraction of F_j that incurs the ith cost component D_{ijp} or E_{ijh}. F_j and G_{ij} are fractions, such that $\sum_{j=0}^{K-1} F_j$ must equal 1 and $\sum_{i=0}^{M-1} G_{ijp}$ must equal 1. The ijth delay is D_{ij} and the ijth energy cost is E_{ijh} for the jth fraction of the algorithm and the active or idle unit, $h \in \{A, I\}$. The fraction F_j is divided by N_{jA} units in duration, but multiplied by $N_{jA}+N_{jI}$ computational units operating in parallel. N_{jp} is the number of units of type P assigned to the jth fraction of the algorithm. The innermost summation in the energy term is the average Energy-Per-operation of a computational unit $h \in \{\text{Active or Idle}\}$: $EPI_{ph} = \sum_{i=0}^{M-1} G_{ihp}E_{ihp}$.

In the outer summation of the delay term, N_j in the denominator reflects the speedup in delay obtained by parallelizing the algorithm over N units. In the middle summation of the energy term, N_{jh} is the number of active or idle units during the jth phase of the algorithm. However, while delay is reduced by a factor of N, energy expended is increased by a factor of N, since there are now N computational units running in parallel. This results in the N_j in the inner summation.

Adding an exponential weighting parameter γ to the energy side of the equation allows energy and delay to be unequally weighted. In the realm of energy efficient design, two metrics are typically used for design evaluation: the energy–delay product ED and energy–delay squared ED^2. The energy–delay product equally weights the contribution of delay and energy, while energy–delay squared doubly weights the contribution of delay in order to emphasize performance over energy savings. In our model, using $\gamma = 1$ results in the standard energy–delay product, while with $\gamma = 0.5$, the contribution of delay is twice as large as the contribution of energy, analogous to the energy–delay squared metric.

Efficient parallel computational architectures are developed by minimizing this cost function in order to maximize the performance of the architecture. The cost function is minimized by: maximizing system parallelism and minimizing the energy and delay costs of operations within the constituent cores. In Section 6 we will use the theoretical framework described in this section to explore the architectural space of spiking neural arrays implemented in FPGAs. However, prior to that, we must define the organization of the spiking neural array as well as the basic computational structures for spiking neurons, the interconnect fabric and local learning machinery.

4. Spiking neural arrays

In this section we present an architecture for implementing large-scale arrays of digital neurons. Although our architecture targets both FPGAs and standard cell ASICs in the current work, we report results from commonly available high-end FPGAs that have the advantage of reprogrammability. ASICs on the other hand, have higher density and operational performance as well as low power operation, with allocation of resources (RAM and logic) custom to the task at hand. However, FPGAs are amenable to reconfiguration hence suitable for the experimental work on architecture exploration that is discussed in the next section. Furthermore, advances in software tools allow the compilation of hardware design using high-level design languages (Cardoso et al., 2010).

4.1. Spiking neural array architecture

A system-level block diagram of the spiking neuron array and its interfaces is depicted in Fig. 2. Spike events enter and exit the system from the left side of the diagram. Events are communicated using the AER (Address Event Representation) protocol, a means of multiplexing data from multiple sources onto a single shared bus. Events traverse the architecture in Fig. 2 in a clockwise direction. Each incoming event looks up a synaptic weight value. That weight is sent to the appropriate neuron in the array based on the event address. When a neuron generates an event, it is tagged with the address of the generating neuron. The re-mapper translates the generating address into a set of destination addresses, which are sent back into the neuron array.

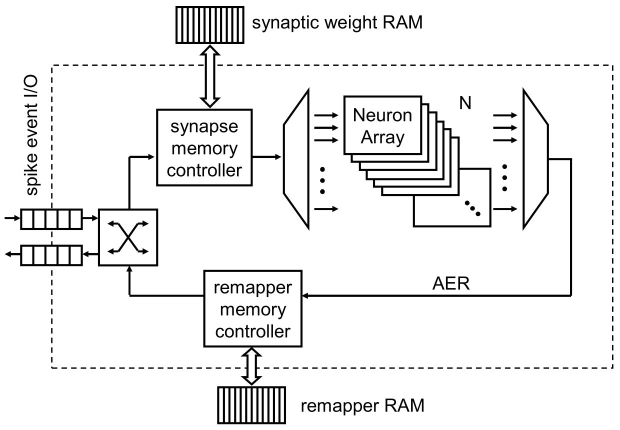


Fig. 2. Spiking neural array architecture.

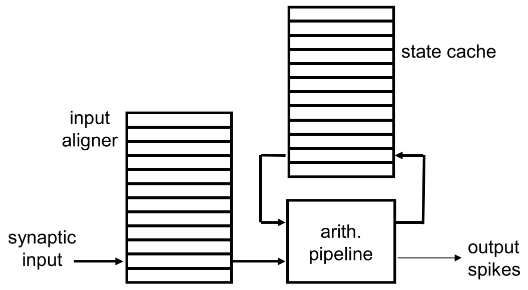


Fig. 3. Multiplexed spiking neuron block diagram.

The high-level architecture is derived from our earlier work (Cassidy & Andreou, 2008; Cassidy et al., 2007). One of the differences in the architecture presented in this paper is that the synaptic weight RAM and the re-mapper RAM are both implemented in SRAM external to the FPGA. The latter modification allows utilization of a much higher total RAM capacity as compared to internal to the FPGA. In addition, this leverages the increasing density benefits of commercial SRAM technology. The targeted system in this work is the Nallatech FSB accelerated computing platform (Nallatech, 2008), with an FSB expansion module based on a Xilinx Virtex 5 SX240T FPGA (Xilinx, 2011) and two GSI Technologies 36 Mb QDR SRAMs (GSI, 2011).

Each neuron in the array is a computational engine (or core) composed of local SRAM and an arithmetic pipeline for computation. One key feature of the neuron, introduced in our earlier work, is neural state multiplexing (Cassidy & Andreou, 2008). In this scheme, the computation for multiple neurons is multiplexed onto a single physical neuron. This requires two local memories, one to store the state for the neurons that are not being currently computed (the state cache), and one to align input events with the proper timeslot in the frame (the input aligner cache).

The multiplexed neuron block diagram is shown in Fig. 3. The state cache is a dual port RAM so that the pre-computation neuron state is read out of the cache at the same time as the post-computation neuron state is written back into the cache. The input aligner cache has two banks of dual port RAM. The two banks implement a ping-pong buffer to decouple writing new events

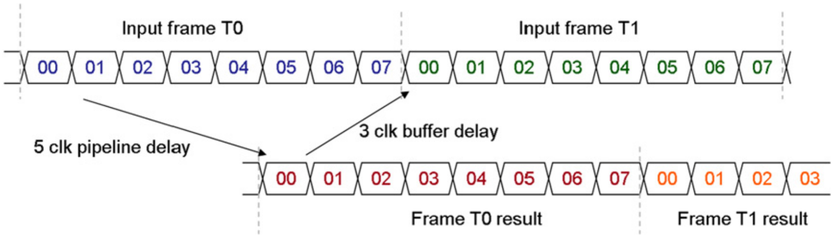


Fig. 4. Frame processing example.

from reading out current events. New events are written into one bank while current events are read out from the other. Writing new events is a read–modify–write operation using both ports of the RAM bank in order to prevent events in the buffer from being overwritten. Reading out current events also uses both ports of the RAM bank. The current value is read out of one port, while the other port clears the RAM location (one clock cycle delayed) removing it from the circular buffer.

Frame-based processing of multiplexed neurons effectively time multiplexes several neuron state computations onto a single neural arithmetic pipeline. Every physical neuron instantiated in the array computes the independent state of M multiplexed neurons, where M is the frame size. Fig. 4 illustrates the frame processing paradigm for $M = 8$ multiplexed neurons.

The input state for frame $T0$ is read out of the state cache and the input aligner simultaneously, and sent to the arithmetic pipeline. The results from the arithmetic pipeline are available after a few clock cycles and are written back into the state cache. All computations are fully pipelined, so that operations are performed every clock cycle and there are no stalls in the datapath. Computation on frame $T1$ begins exactly one cycle after frame $T0$ enters the pipeline. The maximum frame size is constrained only by the on-chip memory required to store the neural state variables. The frame size also affects the computational speedup of the system, in an inversely proportional relationship. If the frame size doubles, the speedup is halved. In order to maintain full-rate processing, the minimum frame size should be greater than the pipeline depth (five clock cycles). The neuron blocks are logically arranged in a linear array and the communication fabric is comprised of a tree arbiter which multiplexes events onto the shared AER bus.

4.2. *Dynamical spiking silicon neurons*

The design space of spiking neuron models spans many decades of research, beginning with Hodgkin–Huxley's pioneering work in 1952 (Hodgkin & Huxley, 1952). Their detailed approach models neuron behavior using four differential equations to represent the membrane dynamics and the non-linear conductances of three types of ion channels. While this detailed approach produces a biophysically accurate model, it is computationally intensive and requires the estimation of a large number of free parameters. Since then, numerous models have been made in order to reduce the complexity of the model. Typically however, reducing the model complexity also reduces the biophysical accuracy and the number of neural behaviors that can be reproduced by the model. The Leaky Integrate and Fire (LIF) neural model is popular because of its relative simplicity, however, it cannot reproduce many complex neural behaviors. Izhikevich reviewed ten models including Hodgkin–Huxley and LIF, comparing them with his own approach (Izhikevich, 2003, 2004). His simplified approach (hereafter denoted as ''IZH''), contains only two coupled differential equations and yet is able to reproduce many complex neural behaviors. We adopt the IZH model as a computationally efficient model for neurons requiring complex dynamical behavior.

4.2.1. *Izhikevich neuron model*

The dynamics of the Izhikevich spiking neuron model are defined by two coupled differential equations, and a reset condition. The variable v represents the voltage across the neural membrane and u is a slow variable, representing membrane recovery.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2}$$

$$u' = a(bv - u). \tag{3}$$

The synaptic input is I and a, b are parameters controlling the dynamical behavior of the neural model. The reset condition is defined by:

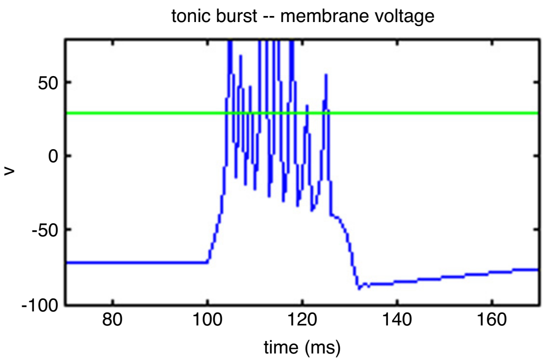
$$\text{if } v \geq +30 \text{ mV}, \quad \text{then } \begin{cases} v & \leftarrow c \\ u & \leftarrow u + d \end{cases} \tag{4}$$

where c, d are parameters controlling the neural reset behavior.

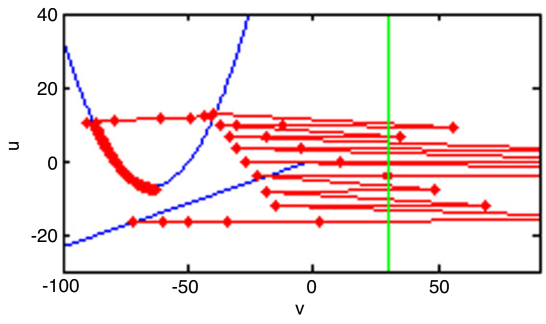
An example of the complex behavior achievable with the IZH model is tonic bursting. (Refer to Izhikevich, 2003, 2004 for 19 other neural behaviors produced by the IZH model.) Fig. 5(a) shows the membrane voltage ('v' parameter) for a burst of neural spikes. The spike firing threshold is 30 mV, depicted by the green line. The dynamics of the 'v' variable versus the 'u' variable is plotted in Fig. 5(b), showing the same burst of spikes in v–u phase space. Phase space is an effective method for visualizing the system dynamics. The nullclines are shown in blue ('v' is a parabola and 'u' a sloping line). The phase space diagram depicts the system starting at $t = 100$ with a step input and ending at $t = 200$. From the initial rest potential ($v = -75$ mV), the system moves in the positive 'v' direction until it crosses the threshold (green vertical line) and fires a spike. Upon firing a spike, the system is reset into an unstable region, again moving in the positive 'v' direction, firing another spike. As this bursting behavior continues, the 'u' variable increases until the system is finally reset into a region that moves the system away from the firing threshold (inside the parabola). The system will slowly move along the 'v' nullcline, until it is once again swept into the firing region, generating tonic bursting behavior.

The first implementation of the IZH model in an FPGA was reported in La Rosa et al. (2005). From the little detail reported in the publication describing the work, a single neuron was implemented, running at 1 MHz. This is far from sufficient for large-scale neural simulation acceleration or scalable neural array implementation. Another design uses the Izhikevich model for inspiration, implementing a similar dynamical neuron in analog VLSI (Wijekoon & Dudek, 2006). The fast and slow state variable paradigm is employed (analogous to 'v' and 'u' in IZH), creating a neuron that exhibits oscillatory and bursting behavior. However, this well principled design suffers from the disadvantages common to all analog VLSI approaches (see Section 1).

We implemented the IZH model using fixed point arithmetic. Numeric values were constrained to an 18-bit representation, based on the 18 × 18 fixed point multiplier cores in the FPGAs. We choose a 10.8 fixed point representation as a balance between dynamic range (10-bit integer portion) and precision (8-bit fractional portion).



(a) Tonic burst—membrane voltage.



(b) v–u phase space.

Fig. 5. Dynamical system description for the Izhikevich neuron model. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

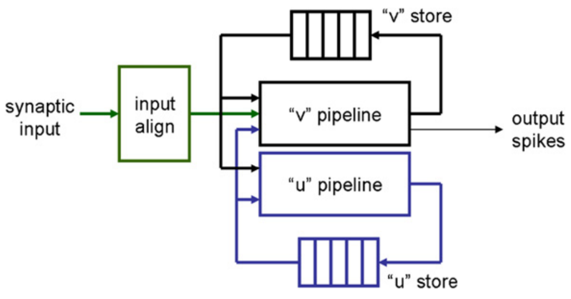


Fig. 6. IZH neuron block diagram.

In addition, significant implementation advantages can be gained if powers of two arithmetic can be used for multiplication and division. With this motivation, we modified Eq. (2) by multiplying the coefficients by 0.78125, for an approximate powers of two representation of two coefficients in the equation, ($0.78125 \times 0.04 = 1/32$ and $0.78125 \times 5 = 3.91 \approx 4$).

$$v[n+1] = v[n] + \frac{1}{32}v^2[n] + 4v[n] + 109.375 - u[n] + I[n] \quad (5)$$

$$u[n+1] = u[n] + a(bv[n] - u[n]). \quad (6)$$

These equations model the same system behavior as Eqs. (2) and (3) above, however, the units have effectively been modified. In addition, the constants a, b, c, d must also be slightly modified from the values of the original IZH model. Lastly, the differential equations are implemented in discrete time.

Using this approach, we created a multiplexed neuron as a base element for creating large-scale neural arrays. The physical neuron implements the arithmetic computations required for the IZH neural model, while local memories buffer the state of multiple neurons between operations, in a time-division multiplexed manner. A block diagram of an individual physical IZH neuron is shown in Fig. 6. The neuron state computations are performed in the 'v' and 'u' arithmetic pipelines. Frame-based processing of multiplexed neurons is supported by the two dual port memories, "v-store" and "u-store", as well as the input alignment block. The dual port memories buffer the multiplexed neuron state between frame iterations. The input alignment block aligns asynchronous input events with their proper timeslot in the frame.

The IZH neuron behavior described in Eqs. (4)–(6) is implemented using two parallel arithmetic pipelines, one for the 'v' dynamics, and one for the 'u' dynamics. These pipelines are shown in detail in Fig. 7. The arithmetic operations in Eqs. (5) and (6) are assigned to arithmetic functional units and arranged according to the standard algebraic order of operations. The data flows directly through the tree from input operands to resulting output. The arithmetic trees maximize the parallelism in time (pipelining) and space (parallel arithmetic units). The computations in each arithmetic tree are fully pipelined to support full-rate dataflow processing. The pipeline is kept full by the frame based processing of blocks of multiplexed neuron state.

Table 1
Device utilization: Xilinx Spartan XC3S1500.

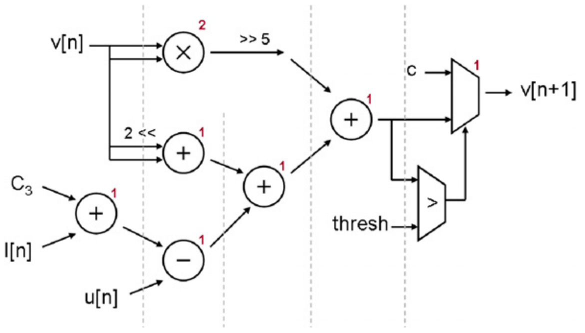
Resource	Percent utilization (%)	Total available
Slice FF's:	64	26,624
4-LUTs:	78	26,624
2 kB RAMs:	34	32
18×18 mults:	100	32

We made three specific optimizations for the algorithm. First, the constant coefficients 4 and $\frac{1}{32}$ in Eq. (5) are implemented as static shift operations (2's complement arithmetic). Second, since multipliers are a scarce resource in FPGAs, the multiplication of the parameter 'a' in Eq. (6) is implemented using a shift and add/subtract operation. This limits the resolution of the values that 'a' can take on, however, it is efficiently implemented in logic. Third, the multiplication operations in both pipelines (Eqs. (5) and (6)) share the same physical multiplier, via time multiplexing. As a result, the net throughput of the pipelines is halved (i.e. if the FPGA runs at 80 MHz, new results are generated by the pipelines every 40 MHz). This tradeoff is also made in order to optimize the utilization of the scarce multiplier resource.

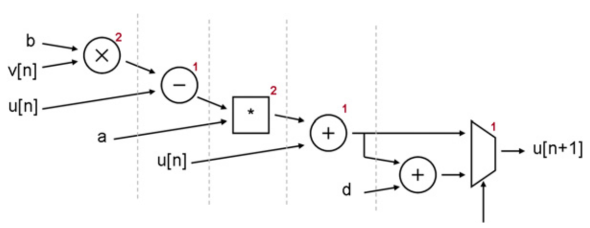
We implemented the silicon spiking neural array in a Xilinx Spartan XC3S1500 FPGA (Xilinx, 2011), hosted on an Opal Kelly XEM-3010 FPGA integration module (Opal-Kelly, 2012). The integration module has a USB 2.0 interface to a host PC. High-level control and interface to the design is through MatLab or Visual C++.

The device utilization is summarized in Table 1, and is roughly balanced for each resource (with the exception of 2 kB RAMs). Ultimately, the number of physical neurons that can be instantiated is limited by the number of 18×18 multiplier cores available on the device. The number of multiplexed neurons that can be multiplexed onto each physical neuron is limited by the amount of distributed memory available (LUT resource).

In the XC3S1500 FPGA, we have 32 physical neurons with 8 multiplexed neurons each, for a total of 256 neurons in the array. With a state of the art FPGA, the number of neurons can be increased by at least two orders of magnitude. A current generation Xilinx XC7VX980 FPGA has 3600 multipliers and 54 Mb of internal SRAM. Including multiplexed neurons, these devices enable on the order of one million independent dynamical neurons per chip (see Section 6).

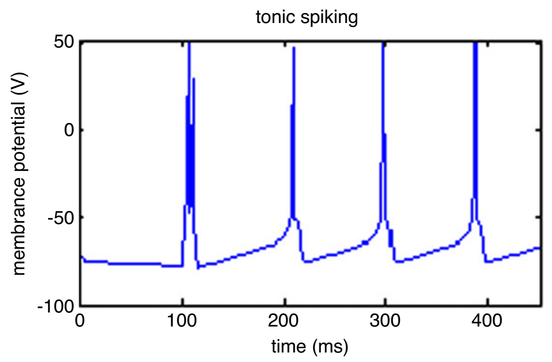


(a) ‘v’ pipeline.

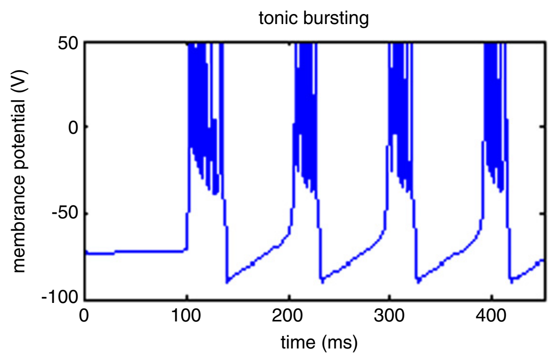


(b) ‘u’ pipeline.

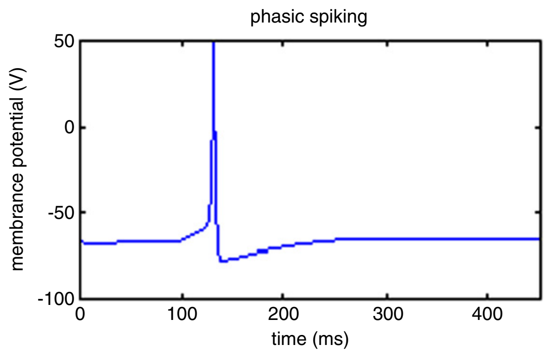
Fig. 7. IZH neuron arithmetic pipelines.



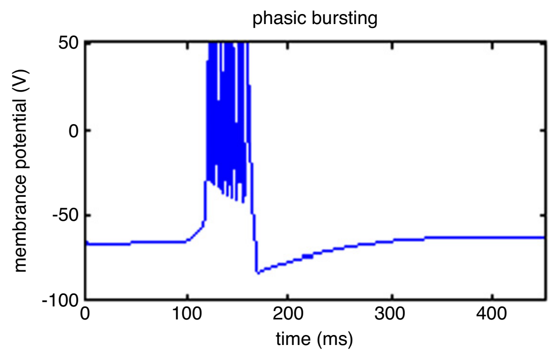
(a) Tonic spiking.



(b) Tonic bursting.



(c) Phasic spiking.



(d) Phasic bursting.

Fig. 8. Dynamical IZH neuron behavior.

Although the IZH neural array operates at 80 MHz in the Xilinx Spartan XC3S1500 FPGA, the multipliers are time multiplexed between the ‘v’ and ‘u’ arithmetic pipelines, so that the effective processing rate is 40 MHz. A 40 MHz clock period is 25 ns, far faster than the neurobiological timescale. Since multiplexed neurons are processed in frames, it takes 8 clock cycles or 200 ns to process one frame of 8 neurons. Then, assuming a biologically realistic simulation timescale of 1 ms per clock cycle, our silicon neural array simulates spiking neural networks 5000 times faster than biological real time.

The IZH neural array was designed and simulated in VHDL, prior to mapping onto the FPGA. Cycle and bit accurate simulations of the VHDL (ModelSim) capture the system functionality. Fig. 8 depicts the dynamical behavior of a single IZH neuron in four different test cases, tonic spiking, tonic bursting, phasic spiking, and phasic bursting. All four test cases are in response to a step input (at 100 ms). Tonic spiking and bursting have persistent activity for the duration of the step input, while phasic spiking and bursting generate activity only at the onset of the step input. The parameters for the four test cases are shown in Table 2. They were obtained from (Izhikevich, 2004) and modified for our fixed point implementation.

Table 2
Fixed point model parameters (TS = Tonic Spiking, TB = Tonic Bursting, PS = Phasic Spiking, PB = Phasic Bursting).

	a	b	c	d	I
TS:	$\frac{1}{64}$	0.156250	−50.508	6.2500	10.9375
TB:	$\frac{1}{64}$	0.234375	−39.063	3.9062	0.58594
PS:	$\frac{1}{64}$	0.273438	−50.508	6.2500	11.7188
PB:	$\frac{1}{64}$	0.273438	−42.969	1.1719	0.78125

An estimate of the computational power of the IZH neural array is as follows. Eqs. (4)–(6) are implemented with 16 fixed point arithmetic operations: 2 multiplications, 9 additions/subtractions, 1 compare operation, and 4 shift operations. All of the operations are fully pipelined and parallel. Thus running at 40 MHz, each pipeline pair computes: 16×40 MHz $= 640$ MOPS. There are 32 pipelines in the array, so the FPGA is performing: 32×640 M $=$ 20.48 GOPS. This is substantial performance for a medium sized FPGA. Also note that 20.48 GOPs is sustained performance—not maximum performance as is reported by many performance measures. The FPGA sustains 20.48 GOPS of computation, guaranteed by full data flow processing. This computational measure also only includes the arithmetic computations in the design. The routing, spike generation, and other array operations are performed entirely in parallel on the FPGA, but require additional computational cycles if run on a microprocessor.

4.2.2. Leaky integrate and fire neuron model (LIF)

The membrane voltage dynamics of the LIF neuron model is defined by the following equations. The equation for synaptic integration is:

$$v_s(n) = \sum_{i=1}^{N_s} w_i x_i(n) \tag{7}$$

where v_s is the synaptic input contribution to the membrane potential, N_s is the number of synapses, w_i are the synaptic weights, (positive w_i for excitatory synapses, and negative w_i for inhibitory synapses), and $x_i(n) \in \{0, 1\}$ denotes the arrival of a presynaptic spike on input i at time n. The neuron firing dynamics are defined by:

$$v(n) = \begin{cases} v_{\text{rst}} & \text{if } v_s(n) > v_{\text{th}} \\ & \text{or } t_o < t_{\text{abs\_r}} \\ v(n-1) + v_s(n) - v_L(\tau) & \text{otherwise} \end{cases} \tag{8}$$

where $v(n)$ is the membrane potential, $v(n-1)$ is the membrane potential at the previous time instant, v_{rst} is the reset potential, v_{th} is the threshold potential, $v_L(\tau)$ is the exponentially decreasing leak voltage with time constant τ, t_o is the time since the last output spike event, and $t_{\text{abs\_r}}$ is the absolute refractory period. The relative refractory function is defined as:

$$v_{\text{th}} = v_{\text{th}} + v_{\text{rel\_r}}(t_o) \tag{9}$$

where $v_{\text{rel\_r}}$ is an additional potential added to the threshold potential. This models the decreased probability of firing a spike shortly after an output spike has been generated. $v_{\text{rel\_r}}$ is set to a constant at the time of the output spike event and then decreases linearly with time until it reaches zero.

Using the same neural array architecture, we can substitute in different neuron models, changing the biophysical-level of emulation as well as the implementation complexity. The LIF neuron is a simplification over the IZH model. Its reduced complexity of arithmetic implementation and single state variable means that area to implement each neuron (physical and multiplexed) is minimized, and the dependence on the FPGA multiplier is also eliminated. This enables the overall neural array to scale to even greater neuron densities.

As shown in the LIF neuron block diagram (Fig. 9), there are three primary operations in the arithmetic pipeline, an exponential leak on the membrane potential, synaptic integration, and firing. The fire operation compares the computed value of the membrane potential with a threshold. If the value is above the threshold, a spike event is generated, and the membrane potential is reset to v_{rst}.

A block diagram of an individual physical LIF neuron is shown in Fig. 9. The membrane potential of the integrate and fire neuron is implemented as a 16-bit digital accumulator. When the accumulator exceeds a programmable threshold, a spike is output from the block and the accumulator is reset. The accumulator begins integrating again after the absolute refractory period.

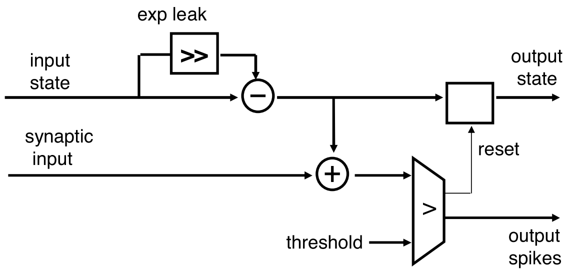


Fig. 9. LIF neuron block diagram.

Table 3
Device utilization: Xilinx Spartan XC3S1500.

Resource	Percent utilization (%)	Total available
Slice FF's:	28	26,624
4-LUTs:	44	26,624
2 kB RAMs:	34	32

The programmable relative refractory value increases the spike threshold for time $t_{\text{abs\_r}}$, decreasing the probability of firing another spike. Another programmable value sets the exponential decay of the accumulator, emulating a shunting leak current in the neural membrane.

We implemented the silicon spiking neural array in a Xilinx Spartan XC3S1500 FPGA (Xilinx, 2011), hosted on an Opal Kelly XEM-3010 FPGA integration module (Opal-Kelly, 2012). The initial design operates at 50 MHz. It was implemented in approximately 4000 lines of VHDL, requiring only 3 weeks of design and debug time. The device utilization is summarized in Table 3.

A modular approach has led to subsequent redesigns with improved clock frequency to 100 MHz with only a few days of work. The 100 MHz clock period is 10 ns, far faster than the biological neural spike timescale. If we assume a biological simulation timescale of 1 ms per clock, then we can simulate neural networks at 100,000 times faster than real time. Or, using an appropriate multiplexing scheme, the speedup could be used to simulate a greater number of neurons on a biological timescale. This tradeoff as well as the scaling of this architecture to truly large-scale neuromorphic systems is detailed extensively in Section 6.

To compare the performance of our architecture with a general purpose processor, we created a simple simulation modeling spatio-temporal receptive field convolutions in the auditory cortex using spiking neurons. The task consisted of 32 integrate and fire neurons with 16 synapses each, computing the convolutions over 100,000 timesteps. The average input spike rate per neuron was 70.4 spikes per second and average output spike rate of 11.1 spikes per second. We coded the simulation in low-level C for the processor, and we also ran the exact same task on our FPGA array of 32 neurons with the same parameters and input dataset.

The single core 2.13 GHz Pentium 4 processor ran this simulation in 45.5 ms (mean of 8 trials). The array of 32 FPGA integrate and fire neurons, running at 100 MHz, computes the same task in exactly 1.0 ms, a $45\times$ speedup over the single core, general purpose processor. This speedup is due to the parallelism in the neural array ($N = 32$) as well as the special purpose micro-architecture of the neural engines. Every operation is pipelined, including control, datapath arithmetic, and memory operations.

The neural architecture and analysis presented in this section also supports other neural models, such as the Izhikevich or Mihalas–Niebur (Mihalas & Niebur, 2009) neurons, by substituting into the arithmetic pipeline the appropriate computations. Without loss of generality, in the remainder of the architecture-level discussion, we use the LIF neural model.

5. Computational structures for STDP learning

Endowing large-scale neuromorphic systems with integrated learning enables them to adopt new behaviors or adapt to new stimuli. Furthermore, this opens possibilities for developmental approaches where the system evolves to optimally solve problems under relevant constraints. Thus, considerable research effort is directed towards developing learning capabilities for neuromorphic and other synthetic intelligent systems.

One learning method that has garnered substantial interest recently is Spike Timing Dependent Plasticity (STDP) (Bi & Poo, 1998; Markram, Gerstner, & Sjöström, 2011; Song & Abbott, 2000), a biologically-based, Hebbian reinforcement learning rule. This particular learning rule is attractive from a hardware perspective because it exploits the robustness of discrete value (digital) representations employed in long distance communication while preserving analog information in the time dimension. In this paradigm, inputs that contribute to a neuron firing are strengthened, while inputs that do not contribute are weakened. Contribution to firing is determined by the time of an incident (presynaptic) spike relative to the time of the neuron firing a (postsynaptic) spike. Indeed, many approaches have adopted STDP, using analog long-term storage (Bartolozzi & Indiveri, 2007; Bofill-i Petit & Murray, 2004; Chicca et al., 2003; Indiveri et al., 2004; Indiveri, Chicca, & Douglas, 2006; Koickal et al., 2009; Schemmel, Grubl, Meier, & Mueller, 2006) and floating gate (Liu & Mockel, 2008; Ramakrishnan, Hasler, & Gordon, 2012) circuits. The recent work by Bamford, Murray, and Willshaw (2012) provides an excellent overview in the state-of-the-art for analog hardware implementations. Digital implementation of the STDP rule has also been developed and reported in the literature (Belhadj et al., 2009; Cassidy, Andreou, & Georgiou, 2011; Cassidy et al., 2007).

A key characteristics to large-scale learning is the scalability of the learning circuit. The size and complexity of a single synaptic learning circuit is multiplied by the number of learning synapses in the system. In a system with 10^3–10^6 neurons and 2–3 orders of magnitude more synapses, the silicon area required to implement learning can be significant. We show that two key innovations enable scalable learning circuits: multiplexing of the learning circuit (Cassidy et al., 2007) and low complexity of the learning circuit (Cassidy, Andreou et al., 2011). Naturally, the best improvement in size and complexity over baseline performance is obtained by using both techniques together, as shown here.

5.1. The STDP learning rule

The STDP learning rule is an unsupervised method that updates synaptic weights based on the time of a presynaptic input spike relative to the time that an output spike event is generated. The concept is that synapses that contribute to the generation of an output spike event should be strengthened, while non-contributing synapses (i.e., those whose input spikes occur after the output spike is generated) should be weakened. The STDP modification function (Fig. 10), shows the change in synaptic weight based on the relative arrival time of a presynaptic input spike on a particular synapse. The output spike is generated at time zero in the plot. If a presynaptic spike arrives just *before* an output spike is generated, then the weight is increased by the amount Δw. If a presynaptic spike arrives just *after* an output spike is generated, then the weight is decreased.

5.2. Multiplexed learning circuits

The functional block diagram of a multiplexed implementation of digital STDP is shown in Fig. 11. In this approach, the presynaptic

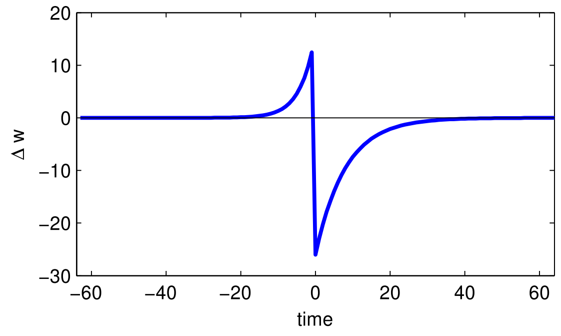


Fig. 10. General STDP modification function.

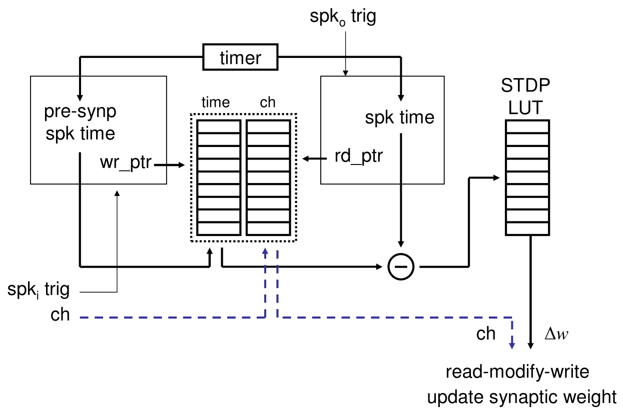


Fig. 11. Block diagram: baseline digital STDP.

spike times are stored in a circular buffer along with the synaptic index. When the neuron generates a postsynaptic spike, the time of the postsynaptic spike is compared with the stored presynaptic spike times. The time difference is used to address the look-up table (LUT) storing the STDP modification function (Fig. 10). The LUT outputs Δw, the value to modify the synaptic weight. A 'read–modify–write' block receives the Δw and modifies the appropriate synaptic weight based on the index stored in the circular buffer.

5.3. Low-complexity learning circuits

A low-complexity digital implementation of the STDP learning rule begins with the observation that weight update functions can be built from the convolution of digital signals. Fig. 12 shows the simplest case, a convolution of a rectangle function with a single pulse (the synchronous digital equivalent to an impulse function). By combining the output of multiple rectangle-impulse convolutions, the STDP function can take on many shapes of varying amplitude, width, and curvature, as shown for example in Fig. 14. The convolution of rectangle functions to create a triangular STDP function is shown in Fig. 16. In this case, the convolution of two rectangle functions creates a pyramid shape. The pyramid must be bisected in the center in order to create the negative weight update. This bisection is accomplished by selecting between the increment and decrement convolutions, depending on whether the presynaptic spike occurs prior to or following the postsynaptic spike.

The STDP learning rule updates synaptic weights based on the relative spike timing of presynaptic and postsynaptic spikes. Implementing this learning rule in silicon requires measurement of relative spike timing and a feedback path to modify the synaptic weights.

Our minimum complexity implementation encodes this update function using combinational digital logic. Fig. 13 gives an

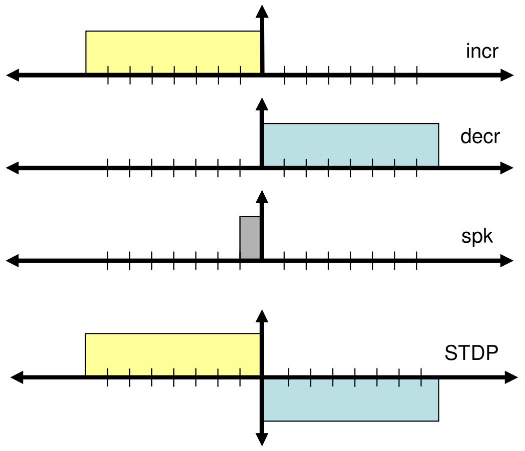


Fig. 12. Timing diagram: STDP modification function—I.

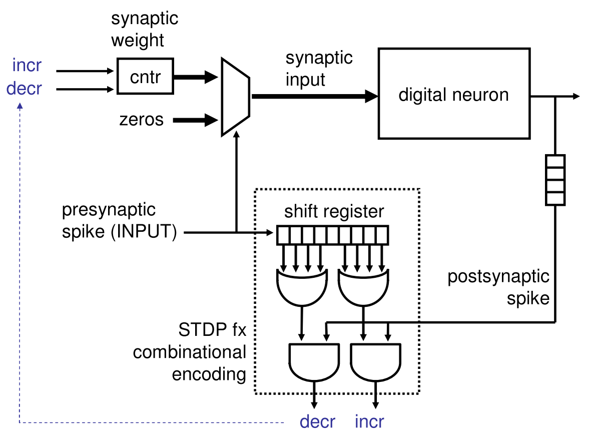


Fig. 13. Block diagram: minimum complexity digital STDP, encoding—I.

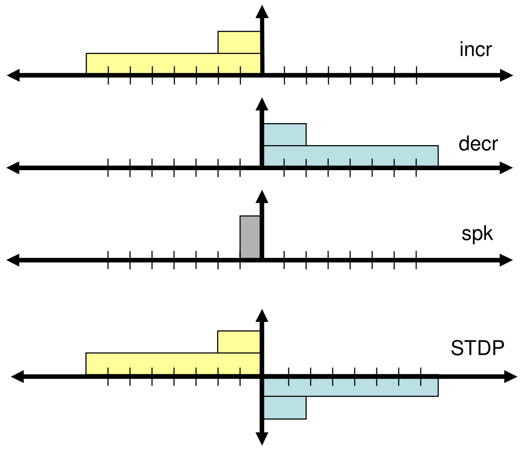


Fig. 14. Timing diagram: STDP modification function—II.

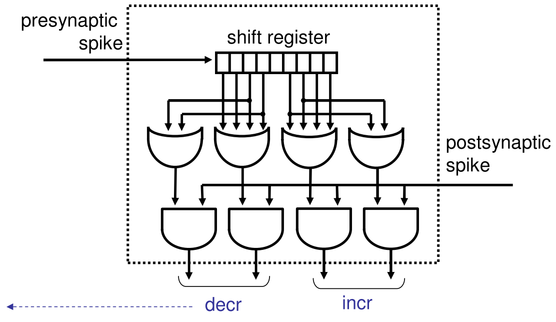


Fig. 15. Block diagram: STDP combinational encoding—II.

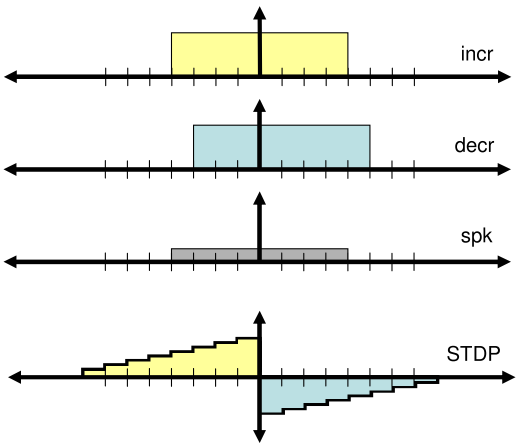


Fig. 16. Timing diagram: STDP modification function—III.

overview of a complete neuron including synapses and learning. In this implementation, synaptic weights are stored using a signed binary up/down counter capable of representing both excitatory and inhibitory synapses. (The learning circuit can also be used with more traditional RAM-based synapses by substituting a RAM with the synaptic weights and a read–modify–write operation instead of the up/down counter.) When a presynaptic spike arrives, the value of the synaptic weight is sent to the digital neuron. In addition, the presynaptic spike is also sent to a shift register in the STDP encoding block. The neuron integrates the synaptic input, and if it exceeds its firing threshold, it emits a postsynaptic spike. The postsynaptic spike is also sent to the STDP encoding block. Inside the STDP encoding block, a shift register and two `OR` gates create rectangle functions for increment and decrement. The postsynaptic spike is a single cycle pulse. The rectangle functions and the pulse are convolved using an `AND` gate, creating the STDP modification function shown in Fig. 12.

More complex STDP modifications, as depicted in Figs. 14 and 16, are created by changing the combinational logic and shift registers in the STDP encoding block. Fig. 15 shows the combinational encoding block for the STDP modification function shown in Fig. 14. This block directly replaces the block in Fig. 13 labeled "STDP fx combinational encoding". In this case, a second, smaller rectangle function is created by replicating the `OR` gates that aggregate the shift register values. We combine the two increment signals to form a pulse one or two cycles wide (depending on whether one or two rectangle convolutions are non-zero). Essentially we are pulse width encoding the amplitude of the STDP modification (Δw). The pulse width encoded signal is decoded by the up/down counter which increments (or decrements) once every clock cycle that the increment (or decrement) signal is asserted.

The third approach convolves rectangle functions as shown in Figs. 16 and 17. The rectangle function for the postsynaptic spike is also created using a shift register-`OR` gate combination. An additional function, the I/D sel block, selects between incrementing and decrementing the synaptic weight based on the binary decision of whether the presynaptic spike came before or after the postsynaptic spike. This decision is also encoded using basic combinational logic. We contrast these low-complexity STDP implementations with a baseline implementation reported earlier in Cassidy et al. (2007).

5.4. Extension to multiple synapses

The small, low-complexity STDP circuits presented thus far have been for a single synapse–neuron pair. With a modest amount

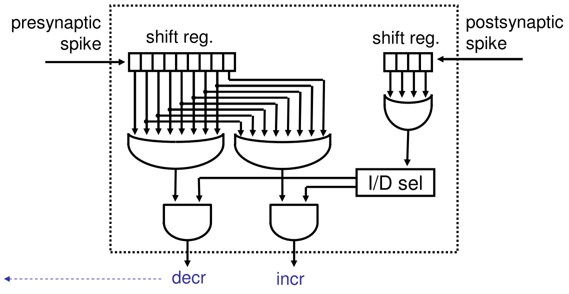


Fig. 17. Block diagram: STDP combinational encoding—III.

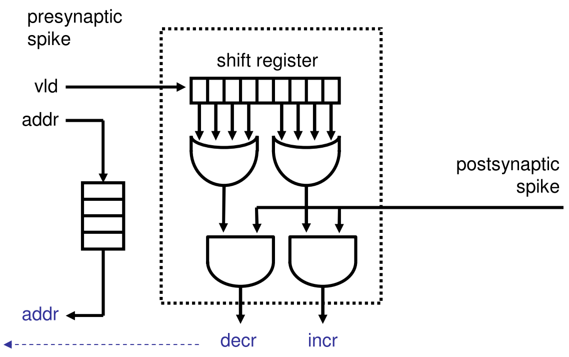


Fig. 18. Multiple synapse extension, first order approach.

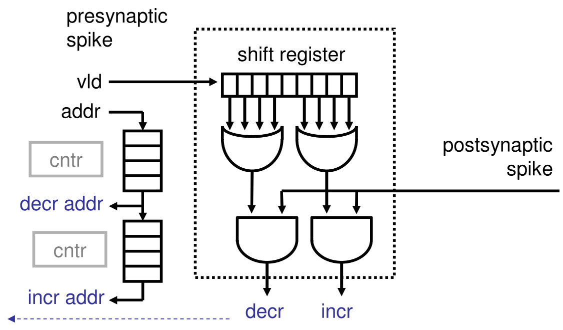


Fig. 19. Multiple synapse extension, second and third order approaches.

of additional circuitry, one STDP circuit can be used by all of the synapses in a neuron, by storing the synapse address of the presynaptic spike and then sending it out with the increment or decrement update signals. This first order approach using a single FIFO is shown in Fig. 18. However, this approach only updates one synapse for each postsynaptic event. By using two FIFOs, as shown in Fig. 19, up to two synapses per postsynaptic event can be updated, one increment and one decrement. By also adding counters that flush all of the addresses in the FIFOs, this can be extended to update an arbitrary number of synapses for each postsynaptic event.

We synthesized each of these STDP approaches, targeting a Xilinx Spartan XC3S1500 FPGA (Xilinx, 2011). For comparing area results, each approach uses a time window of 32 clock cycles, 16 prior and 16 following the postsynaptic spike. The resource utilization for the single synapse circuit, as well as the multiple synapse extension, are summarized in Table 4. The low-complexity STDP approaches are over 4× smaller than the baseline approach in flip-flops and over 20× smaller in terms of LUTs. The multiple synapse extension using FIFOs uses additional resources, but is still 2.8× smaller in flip-flops and 8.8× smaller in LUTs. Synthesizing the design for a 130 nm CMOS ASIC technology shows similar area

Table 4
Utilization: Xilinx Spartan XC3S1500.

Approach	Slice FF's	4-LUTs
Baseline	159	371
Comb. encoding I	35	14
Comb. encoding II	37	17
Comb. encoding III	39	16
Comb. encoding I—FIFO, 1	46	24
Comb. encoding I—FIFO, 2	48	26
Comb. encoding I—FIFO, 3	56	42

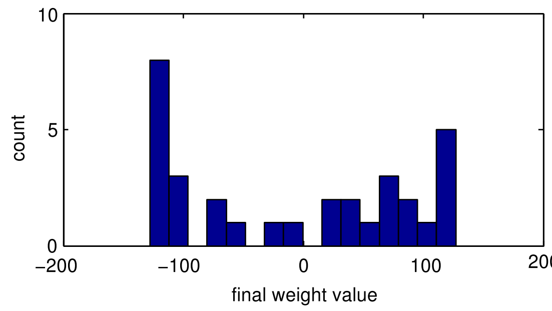


Fig. 20. Synaptic weight distribution histogram after STDP learning. Baseline system, 32 synapses. Single trial.

improvement. For additional comparison, the approaches given in Belhadj et al. (2009) require block RAMs, embedded multipliers, as well as 28× to 480× more LUTs than our largest approach (Comb. Encoding I—FIFO, 3).

5.5. Experiments and results

We tested the performance of the STDP learning algorithm by performing a balanced excitation experiment, based on the experiment run by Song et al. (see Figs. 2a and 2b in Song & Abbott, 2000). In this experiment, 32 synapses from a single neuron start with a uniform positive weight distribution. Each synapse is driven by an independent Poisson spike train input with the same average rate. When STDP is enabled, the synapses converge to a steady state condition with a bimodal distribution of excitatory and inhibitory weights.

The baseline implementation replicated the bimodal distribution as shown in Fig. 20. The results from the combinational encoding approaches are shown in Figs. 21–23. Each histogram plot shows the synaptic weight distribution after STDP learning. The plots are the aggregate results of eight trials using different stimuli for each trial. It is apparent that even the simple scheme (approach I) is capable of producing the expected bimodal distribution (Fig. 21). However, it is also apparent that approach III produces the best[3] shaped distribution, which is unsurprising given the best modification function shape shown in Fig. 16.

The STDP functions for this experiment are not symmetric, as is depicted in Figs. 12, 14, 16. Instead, the widths of the increment and decrement shift registers (rectangle functions in the convolution) are given in Table 5. These parameters were used to obtain the results shown for this experiment.

Finally, we use the STDP learning rule implementation to model the effects of ocular dominance during cortical column formation in the visual cortex. This experiment reproduces the results of Kanold and Shatz (2006) while employing silicon spiking neurons, synapses, and STDP learning implemented in the FPGA. Fig. 24

[3] "Best" refers to the distribution closest to Fig. 2b in Song and Abbott (2000), the experiment that we are replicating.

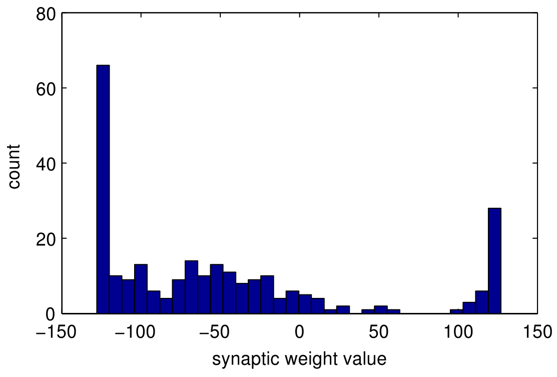


Fig. 21. Synaptic weight distribution histogram after STDP learning. Combinational encoding I system, 32 synapses.

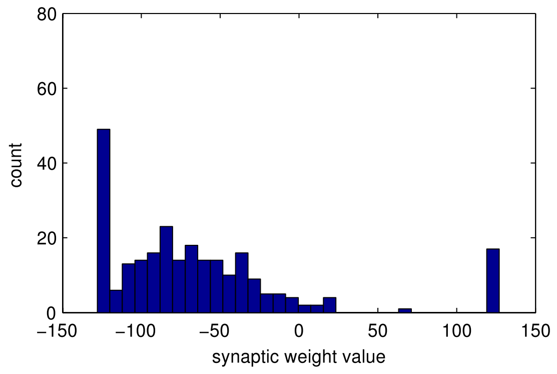


Fig. 22. Synaptic weight distribution histogram after STDP learning. Combinational encoding II system, 32 synapses.

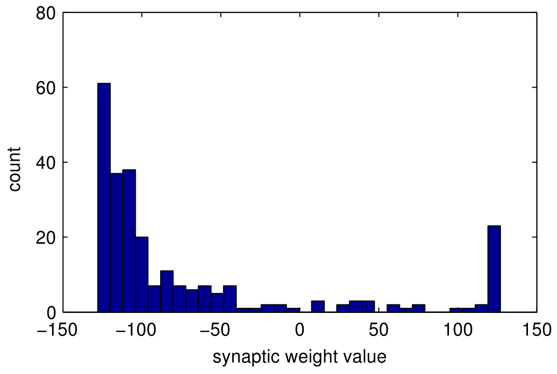


Fig. 23. Synaptic weight distribution histogram after STDP learning. Combinational encoding III system, 32 synapses.

Table 5
Balanced excitation experiment parameters.

Approach	Dec. width	Inc. width
Comb. encoding I	16	3
Comb. encoding II-L1	16	4
Comb. encoding II-L2	8	0
Comb. encoding III	16	8

depicts the cortical column formation scenario. There are two neurons, a layer 4 neuron N0 and a subplate neuron N1. There are two thalamic inputs th0 and th1, one originating from each eye. These inputs connect to the subplate neuron as well as the layer 4 neuron. In addition to the subplate neuron connection, the layer 4 neuron also receives spontaneous input. It is hypothesized that the subplate neuron plays an important role during cortical

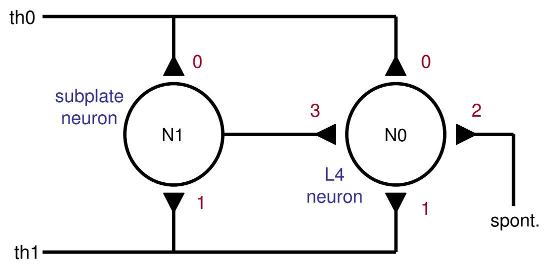


Fig. 24. Ocular dominance experiment: cortical column.

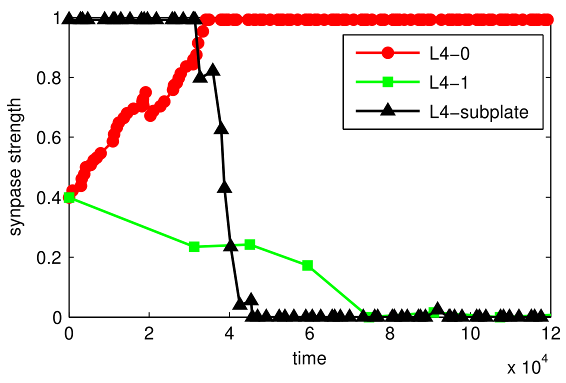


Fig. 25. Cortical column case 3.

column growth. Specifically, there are four cases for different input conditions:

1. No subplate, th0 $>$ th1: thalamic inputs (th0, th1) die away.
2. No subplate, th0 $=$ th1: thalamic inputs (th0, th1) die away.
3. Subplate, th0 $>$ th1: th0 strengthened, th1 and subplate die away.
4. Subplate, th0 $=$ th1: th0 or th1 strengthened, th1 or th0 and subplate die away.

Given a proper single set of system parameters (STDP τ, spike fire thresholds, leak current, and relative refractory period), all four cases can be met depending on the particular input conditions present. For example, results for case 3 are shown in Fig. 25. In this case, the input from th0 is stronger than th1, resulting in a strengthening of the L4-0 synaptic weight and a weakening of the L4-1 synaptic weight. As the L4-0 and L4-1 synapses differentiate, the subplate neuron synapse becomes less important and goes to zero. Results for the other three cases are similar.

6. Neural architecture optimization

Now given the preceding representations and implementations, we return to Marr's level of computational theory. Given the physical constraints of delay, energy, and area, how can we maximize the performance of these parallel computational architectures? In this section, we perform a constrained optimization in order to find optimal architectural parameters. Our goal is to find the optimal number of physical neurons (processors), as well as the allocation of processor and memory resources (expressed in units of area) given a fixed total area on the die and the finite communication bandwidths. Our analysis follows the approach we outlined in earlier work for traditional chip-multiprocessors (Cassidy & Andreou, 2009, 2012). In the present work, processors are the neuron engines and memory is the internal RAM (state cache and input aligner cache). Without loss of generality, throughout this work we employ the LIF neural model.

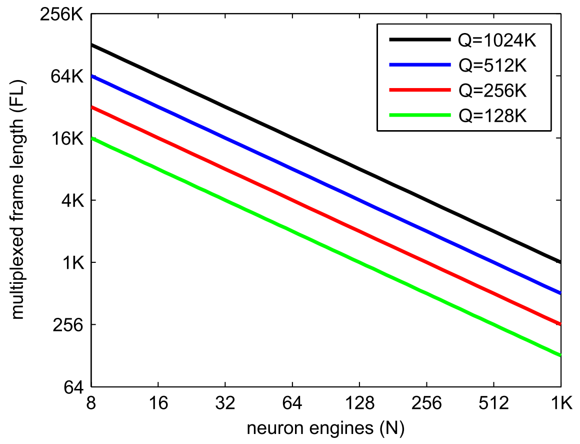


Fig. 26. Frame length FL vs. neuron engines N.

6.1. Physical constraints

Using the architecture outlined in Fig. 2, we begin the discussion of physical constraints, examining the tradeoffs between area and delay.

6.1.1. Physical vs. multiplexed neurons

The total number of neurons in the system (Q) is the product of the number of parallel physical neuron engines (N) and the frame length (FL) or number of neurons multiplexed onto a single physical neuron engine,

$$Q = N \cdot \mathrm{FL}. \tag{10}$$

Thus, for a fixed number of total neurons Q, there are many combinations of N and FL that will result in the desired total neuron count. This tradeoff is shown in Fig. 26. In the neural architecture presented here, there are two types of internal RAM, the state cache and the input aligner cache. The state RAM size is:

$$\mathrm{STATE\_RAM\,(Bytes)} = S \cdot \mathrm{FL} \tag{11}$$

where S is the size of one neuron state in Bytes. The input alignment RAM size is:

$$\mathrm{IN\_ALIGN\_RAM\,(Bytes)} = 2W \cdot \mathrm{FL} \tag{12}$$

where W is the size of one synaptic weight in Bytes and the factor of two accounts for the dual ping-pong banks of the input aligner. The total internal RAM is:

$$\mathrm{INT\_RAM\,(Bytes)} = N \cdot \mathrm{FL}(S + 2W). \tag{13}$$

Thus we will have a constant total internal RAM if $N \cdot \mathrm{FL}$ is held constant. For example, increasing the number of neuron engines N by two while decreasing the frame length FL by two, keeps the total number of neurons Q and total internal RAM constant. On the other hand, there is a distinct tradeoff between N and FL while holding Q constant. Larger values of N (and smaller FL) results in more parallel computational engines, which increases performance (decreases computational delay). However, larger values of FL (and smaller N) take up less area since the logic area (not RAM) is proportional to N and independent of FL. These area relationships will be revisited in Section 6.2.

6.1.2. Real-time speedup

Biological neural firing rates range from zero Hz to a few hundred Hz. A computational rate of 1 kHz is sufficient to capture the internal neural dynamics between firing events. In contrast, clock rates of standard digital FPGAs and standard cells ASICs are a few hundred MHz. In this work, we use a 200 MHz clock in a Xilinx Virtex 5 FPGA. The first-order speedup over real-time is:

$$\mathrm{xRT} = \frac{\mathrm{CLK}}{\mathrm{NCR}} = \frac{200\ \mathrm{MHz}}{1\ \mathrm{kHz}} \tag{14}$$

where NCR is the neural computation rate. NCR is the inverse of the neural simulation timestep (i.e. for a 1 ms simulation timestep, NCR = 1 kHz). By multiplexing neural state on physical neurons, we tradeoff speedup over real-time for system area. In the multiplexing case, the speedup over real-time is (using a multiplexed frame length of FL = 1024):

$$\mathrm{xRT} = \left(\frac{\mathrm{CLK}}{\mathrm{FL}}\right)\left(\frac{1}{\mathrm{NCR}}\right) = \left(\frac{200\ \mathrm{MHz}}{1024}\right)\left(\frac{1}{1\ \mathrm{kHz}}\right). \tag{15}$$

Thus, the maximum frame length FL in order to operate at or above real-time is 200,000 given a neural computation rate NCR of 1 kHz.

6.1.3. External RAM sizing

The size of the synapse RAM is:

$$\mathrm{SYNP\_RAM\,(Bytes)} = W \cdot P \cdot Q \tag{16}$$

where P number of synapses per neurons and W is the synaptic weight size defined above. This external RAM is the dominant factor in determining the total number of synapses in the system, as well as the number of synapses per neuron. The size of the re-mapper RAM is a function of fanout F_O (destinations per neuron):

$$\mathrm{REMAP\_RAM\,(Bytes)} = \frac{1}{8}\log_2(PQ) \cdot F_O \cdot Q. \tag{17}$$

Two factors determine the maximum fanout supported by the system. The first is the size of this external RAM, which determines the maximum number of destinations that can be stored. The second is the communication bandwidth which limits the number of destination events that can be replicated before saturating the system.

In the discussion above, we have assumed that both the synapse RAM and the re-mapper RAM assume a uniform number of synapses or destinations per neuron. This restriction can be removed with the addition of a second lookup stage. This additional lookup would return a pointer into synapse or re-mapper memory as well as a value specifying the number of locations to read beyond the pointer. This would allow an arbitrary number of synapses or destinations to be used on a per-neuron basis. The cost is the area of a second bank of RAM (of size proportional to Q) as well as the delay cost of the second table lookup.

6.1.4. Resource allocation analysis

We account for the resources in the FPGA by using the effective area of each resource. Modern integrated circuits, including FPGAs, are limited by circuits and wiring that exist in a two dimensional surface, the area of the die. Hence it is sensible to use the area as the basic constraint for optimization process.

We begin our analysis by examining the area breakdown of the system. The area of a neuron is:

$$A_{\mathrm{nrn}} = A_{\mathrm{RAM}} + A_{\mathrm{logic}}. \tag{18}$$

The logic area per neuron is a constant A_{logic}, while the RAM area is $A_{\mathrm{RAM}} = \mathrm{FL}(S + 2W)$, from (13). Using $A_{\mathrm{mem}} = (S + 2W)$, the area of the neural array is:

$$\begin{aligned} A_{\mathrm{array}} = NA_{\mathrm{nrn}} &= N(A_{\mathrm{RAM}} + A_{\mathrm{logic}}) \\ &= N(A_{\mathrm{mem}} \cdot \mathrm{FL} + A_{\mathrm{logic}}) \end{aligned} \tag{19}$$

where N is the number of physical neuron engines.

In the array, the input multiplexor scales proportional to N, while the output multiplexor is implemented as a binary tree.

There are $\log_2(N)$ levels in a binary tree, and a total of $N-1$ nodes. Thus, since both scale proportionally to N, they are incorporated into the A_{logic} term of the neuron engine.

The total available area on the silicon die A_{tot} is:

$$\begin{aligned} A_{\text{tot}} &= A_{\text{fix}} + A_{\text{array}} \\ &= A_{\text{fix}} + N(A_{\text{mem}} \cdot \text{FL} + A_{\text{logic}}) \end{aligned} \tag{20}$$

where A_{fix} is the fixed area dedicated to support functions (PLLs, debug circuitry, JTAG, control registers, etc.).

We need a common unit of area in order to perform operations on both A_{mem} and A_{logic}. We use memory byte equivalent units. For logic, this is the size (in Bytes) of an SRAM that occupies the same silicon area as the given block of logic. For memory this is simply the size of the RAM. For ASICs, this conversion is accomplished by determining the average mm^2 area of both SRAM and logic gates for a particular process. Then logic gates can be converted directly into SRAM Bytes, dropping the mm^2 in the process. In FPGAs, we accomplish this conversion by determining the area ratio between block RAMs and Configurable Logic Blocks (CLBs) or logic slices. For a Xilinx Spartan III FPGA, we estimate an 18 kB SRAM occupies the same area as 32 logic slices. Thus we use a conversion of 64 B per slice, subdivided as 32 B per slice LUT and 32 B per slice register.

In our first case, we consider the case where we are constrained not only by the fixed total silicon area, but also by requiring that the total number of neurons is held constant. The total area constraint is given in (20) and the total neuron constraint is given in (10) when Q is equal to a constant. In this case, the two constraints only intersect at a single point. Combining (20) and (10): we obtain:

$$A_{\text{tot}} = \frac{Q}{\text{FL}}(A_{\text{mem}}\text{FL} + A_{\text{logic}}) + A_{\text{fix}} \tag{21}$$

$$\text{FL} = \frac{QA_{\text{logic}}}{A_{\text{tot}} - A_{\text{fix}} - A_{\text{mem}}Q}. \tag{22}$$

Given values of $Q, A_{\text{tot}}, A_{\text{fix}}, A_{\text{logic}}$, and A_{mem}, we can directly determine the value of FL that satisfies both constraints. And the value of N at that point is given by (10).

6.2. Simplified cost function for symmetric multiprocessing

Here we apply this objective function to our symmetric array of parallel neural processing engines. The asymmetric cost function given by Eq. (1) discussed in Section 3.1 simplifies for symmetric architectures (Cassidy & Andreou, 2012) to:

$$J_{ED} = \left[\sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{i=0}^{M-1} G_{ij}D_{ij}\right] \times \left[\sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{h\in\{A,I\}} N_{jh} \sum_{i=0}^{M-1} G_{ijh}E_{ijh}\right]^{\gamma} \tag{23}$$

where F_j is the fraction of the algorithm that has parallelism of N_j. Each fraction F_j is divided into M cost components of the architecture. G_{ij} is the fraction of F_j that has the ijth cost component of D_{ij} or E_{ij}. The ijth delay is D_{ij} and the ijth energy cost is E_{ijh} for the jth fraction of the algorithm and the active or idle processors, $h \in \{A, I\}$. The weighting parameter γ scales the relative influence of delay and energy on the cost of the architecture. Since F_j is a fraction, $\sum_{j=0}^{K-1} F_j$ must equal 1 and since G_{ij} is also a fraction, $\sum_{i=0}^{M-1} G_{ij}$ must equal 1.

Analysis of the symmetric parallel neural architecture proceeds as follows. Optimizing for delay (and neglecting energy, $\gamma = 0$), we have simply:

$$J_D = \frac{1}{N}(G_0D_0 + G_1D_1) \tag{24}$$

where N is the number of neuron engines, G_0 is the average fraction of time the neuron does not spike, D_0 is the time to compute one frame of the neural state for the whole system (Q neurons), G_1 is the average fraction of time the neuron spikes and D_1 is the time to compute neural state plus the communication delay. Note that the "algorithm" is perfectly parallelizable. That is to say that the computation of neural state and firing can be distributed across the parallel neural array without serialization. Thus there is only one level of parallelism ($K = 1$) and $F_0 = 1$.

The output of the neural array is multiplexed onto a shared AER bus. If more than one event is generated in the array during the same clock cycle, one of the events will have to wait in a buffer for an open bus cycle. This contention for the shared communications resource can be modeled using queueing theory. Here we use an M/M/1 queue model, where the server rate μ is a fixed constant, based on the clock frequency of the bus arbiter. The arrival rate λ is a function of the average neuron firing (communication) rate and the number of parallel neuron engines: $\lambda = G_1N$. Using these parameters, the expected communication contention delay for an M/M/1 queue is:

$$\bar{t} = \frac{1}{\mu - \lambda} = \frac{1}{\mu - G_1N}. \tag{25}$$

Given an average neural firing rate of 100 Hz $= 1/100$ s $= 0.01$ s, a system clock rate of 200 MHz, and real-time operation, the queueing parameters are: $\mu = 200$ MHz and $\lambda = N100$ Hz. The computation and communication fractions are estimated as follows: $g_0 = 200$ M, $g_1 = 100$, $g_{\text{tot}} = g_0 + g_1 = 200{,}000{,}100$, and thus: $G_0 = \frac{g_0}{g_{\text{tot}}}$ and $G_1 = \frac{g_1}{g_{\text{tot}}}$. Note that if we operate at a timescale faster than real-time, then $\mu = 200$ MHz$/xRT$ and $g_0 = 200$ M$/xRT$.

We solve our constrained optimization problem using the method of Lagrange multipliers. The Lagrangian is formed by combining the cost function and the area constraint:

$$\begin{aligned} L(N, \text{FL}, \Lambda) &= \frac{1}{N}(G_0D_0 + G_1D_1) \\ &\quad + \Lambda\left[N(A_{\text{mem}}\text{FL} + A_{\text{logic}}) + A_{\text{fix}} - A_{\text{tot}}\right]. \end{aligned} \tag{26}$$

Substituting in, using $D_0 = Q$ and $D_1 = Q + \alpha + \bar{t}$:

$$\begin{aligned} L &= G_0\text{FL} + G_1\left(\text{FL} + \frac{\alpha}{N} + \frac{1}{N(\mu - G_1N)}\right) \\ &\quad + \Lambda\left[N(A_{\text{mem}}\text{FL} + A_{\text{logic}}) + A_{\text{fix}} - A_{\text{tot}}\right]. \end{aligned} \tag{27}$$

Differentiating the Lagrangian with respect to the three variables (N, FL, Λ):

$$\begin{aligned} \frac{\partial L}{\partial N} &= G_1\left[\frac{-1}{N^2(\mu - G_1N)} + \frac{G_1}{N(\mu - G_1N)^{-2}}\right] \\ &\quad + \frac{-G_1\alpha}{N^2} + \Lambda\left(A_{\text{mem}}\text{FL} + A_{\text{logic}}\right) = 0 \end{aligned} \tag{28}$$

$$\frac{\partial L}{\partial \text{FL}} = (G_0 + G_1) + \Lambda\,(NA_{\text{mem}}) = 0 \tag{29}$$

$$\frac{\partial L}{\partial \Lambda} = \left[N(A_{\text{mem}}\text{FL} + A_{\text{logic}}) + A_{\text{fix}} - A_{\text{tot}}\right]. \tag{30}$$

Given these three equations and three unknowns, we can solve for the optimal architecture using standard numerical methods. We can also view the constrained cost function graphically. The cost function J_D, constrained by (20), is plotted in Fig. 27. (Note the logarithmic scale on both axes.) Since the constrained cost function is monotonically decreasing, the optimum architecture (minimum J_D) occurs with a maximum number of parallel physical neuron engines (maximum N given the area constraint) and no multiplexing (frame length FL = 1).

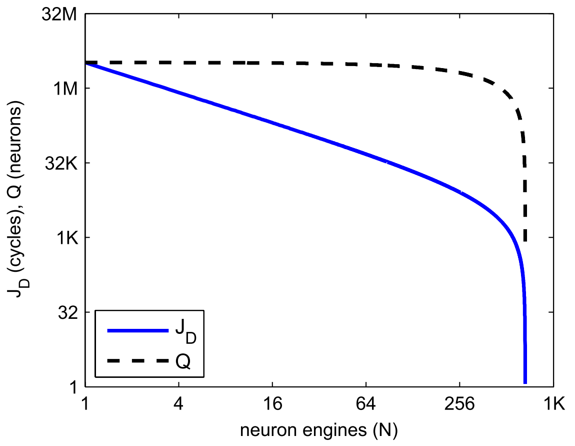


Fig. 27. Cost function J_D in units of cycles and total number of neurons Q versus number of physical neuron engines.

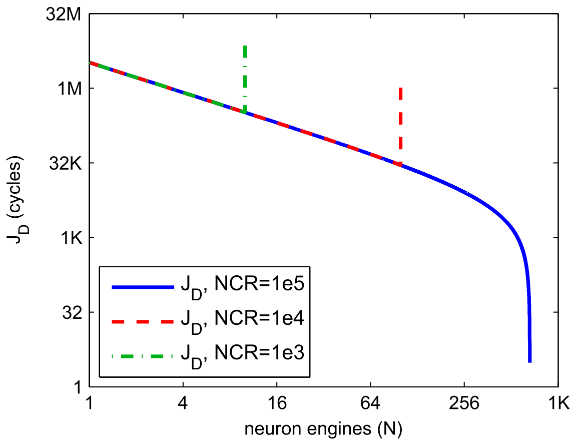


Fig. 28. Communication limited cost function J_D in units of cycles versus number of physical neural engines. (NCR = neural computation rate.)

From this figure, we can see four different interesting regimes. The first one is when minimizing the cost of the system in terms of delay J_D. In this regime, we maximize parallelism, filling the area with physical neurons and not using any multiplexing, thus FL = 1. This architecture is entirely composed of logic and no state RAM. A second interesting regime occurs when maximizing the total number of neurons in the system Q, then we build a system with one physical neuron $N = 1$ and maximum FL. This results in an architecture overwhelmingly composed of RAM. This is intuitive since the marginal increase in area for a multiplexed neuron is 2 B, while the marginal increase in area for a physical neuron is 10 kB. A third case is to maximize performance, while still meeting a specified total number of neurons Q. In this case, we find the desired value of Q on the vertical axis, find the intersection with the Q curve (black dashed), and then find the corresponding value of N at that point. Finally, if we wish to find a "sweet spot", balancing the total neuron density and the performance, we could choose the architecture at the knee of the curve. In Fig. 27, this corresponds to approximately 256 physical neurons, a delay cost of 8 thousand, and 2 million total neurons.

In the cost function example curve in Fig. 27, the minimum occurs with the maximum number of parallel neuron engines and FL = 1. The performance is bound by the limit that FL cannot go below 1. Communication constraints imposes another possible limit to the number of parallel engines that could be implemented. As the number of parallel engines increases, the communication traffic on the shared AER bus increases. If the amount of traffic exceeds the available bandwidth, the system breaks down. As the amount of traffic approaches this limit, the communication delay increases. By varying the neural computation rate (NCR) while holding the average neural firing rate constant, we alter the amount of traffic per simulation timestep, effectively varying the communication traffic generated in the system. Note that we increase NCR at the expense of decreasing xRT, according to Eq. (14). As we decrease the NCR, increasing the effective traffic rate, the system performance will become communication limited. This is shown in Fig. 28. For three values of NCR, we see the effect on the system performance in terms of delay cost J_D. For smaller values of NCR, the performance hits an asymptote, and the optimal architecture is prior to the asymptote.

This architectural exploration assumes an ASIC or custom VLSI design substrate. However to test our architecture and our analysis, we used FPGAs as a surrogate design platform. FPGAs are advantageous for rapid prototyping and reprogrammability, however, they have fixed resources and organization (memory sizes and ports, interconnect overhead, etc.). This places limits on the full range of architectures that can be implemented as compared to an ASIC. The next section presents our results using FPGAs, including discussion on limitations arising from the FPGA organization.

Table 6
Area implementation results: Xilinx V5SX240.

Q (k)	N	FL	REG (%)	LUT (%)	Used RAM (%)	Used BRAMs (%)
512	8	64 k	1.3	0.8	49.6	49.6
512	16	32 k	2.6	1.5	49.6	49.6
512	32	16 k	4.9	2.9	49.6	49.6
512	64	8 k	9.4	5.5	49.6	49.6
512	128	4 k	17.8	10.5	49.6	50.2
256	8	32 k	1.3	0.8	24.8	24.8
256	16	16 k	2.4	1.4	24.8	24.8
256	32	8 k	4.7	2.7	24.8	24.8
256	64	4 k	8.8	5.2	24.8	26.0
256	128	2 k	16.6	9.9	24.8	49.2
128	8	16 k	1.2	0.7	12.4	12.4
128	16	8 k	2.3	1.4	12.4	12.4
128	32	4 k	4.4	2.6	12.4	12.6
128	64	2 k	8.3	4.9	12.4	25.0
128	128	1 k	15.5	9.4	12.4	46.3
128	256	512	32.2	49.2	12.4	34.3
128	512	256	59.6	67.7	12.4	56.6

6.3. Implementation results

We implemented our neural array architecture, targeting Xilinx FPGAs in order to build a working system as well as to verify the analytical model. By using parameterized VHDL, we varied the architectural parameters and implemented different array configurations, thus exploring the design space. We simultaneously varied the number of physical neurons N and the multiplexed frame length FL by factors of 2, holding the total number of neurons Q constant. We repeated this process for four different values of Q: 128 thousand, 256 thousand, 512 thousand, 1 million. For the three smaller values of Q, we targeted a Xilinx Virtex 5 SX240 FPGA, while for the one million neuron case, we targeted a Xilinx Virtex 6 SX475 FPGA. The area implementation results for the V5SX240 are shown in Table 6 while the results for the V6SX475 are shown in Table 7.

We can make several interesting observations based on the data in these tables. First we see that the overall utilization of the FPGA resources is rather low. Block RAM utilization is approximately 50% for 512 thousand neurons and even lower in all of the other cases. Logic utilization ranges from less than a percent ($N = 8$) to approximately 10% for $N = 128$. This highlights the high overhead of using FPGAs with fixed resources. A blank substrate would allow a much higher utilization of the available silicon. Second, we see

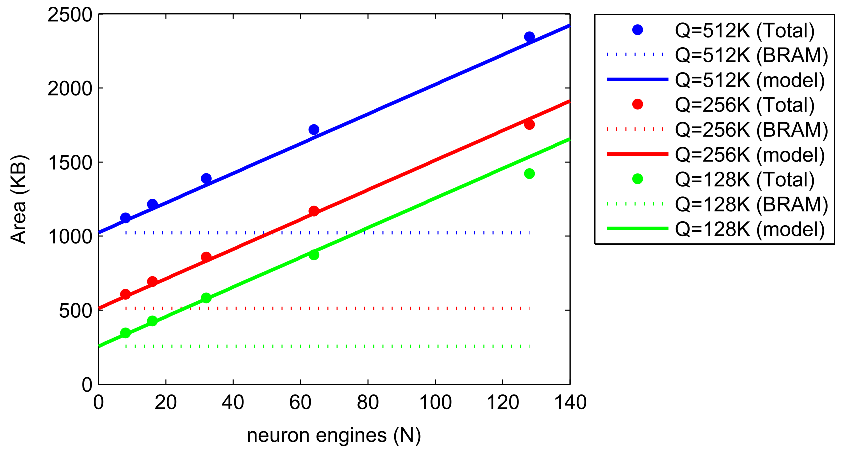


Fig. 29. Area vs. neuron engines.

Table 7
Area implementation results: Xilinx V6SX475.

Q (k)	N	FL (k)	REG (%)	LUT (%)	Used RAM (%)	Used BRAMs (%)
1024	16	64	0.7	1.1	36.1	36.1
1024	32	32	1.3	2.1	36.1	36.1
1024	64	16	2.4	3.9	36.1	36.1
1024	128	8	4.5	7.4	36.1	36.1
1024	256	4	8.6	14.3	36.1	36.1

that the architecture is dominated by RAM, as compared to logic. With fixed FPGA resources, the RAM is the limiting resource. Third, note that in several cases, the RAM is actually limited by the number of RAM ports, and hence number of BRAMs and not the actual amount of RAM actually used. With a constant Q, when N is varied, FL is varied by an equivalent amount in the opposite direction, thus the total RAM should remain constant, as given by (13). This holds true for 512 and 1024 thousand neurons (see the "BRAM blks" column). However, for 128 and 256 thousand neurons, the number of used block RAMs begins to increase even though the used RAM does not. This is to accommodate the number of ports needed by the architecture (# BRAMs $\approx$ RAM ports/2). Block RAMs come in a fixed size: 4 kB, and have two ports. Finally, notice the cases of $Q = 128$ thousand and $N = 256, 512$. These two design points deviate from the trends of linearly increasing LUTs and BRAMs increasing by the number of required ports. This deviation is due to a switch from storing internal state using block RAMs to distributed RAM.

6.4. Area model validation

Plotting the area data reveals a strong correlation between empirical results and our analytical model. According to (19) in our model, the array area is a linear function. Holding the total number of neurons constant $Q = N \cdot \mathrm{FL}$, the area allocated to internal RAM stays constant, while the area for the neural engines scales linearly with N. The empirical and analytical results are shown in Fig. 29. The solid circles are empirical data points, while the lines are the first order analytical model, using $A_{\mathrm{mem}} = 2$ and $A_{\mathrm{logic}} = 10$ kB. The analytical area model parameters and the minimum squared error fit lines for the empirical data points are summarized in Table 8. While the empirical data is very close to our first-order analytical model, we can see that the slope of the MSE lines change for the different values of Q, thus A_{logic} slightly varies as a function of FL. In addition, the fit lines have a small offset (approximately 32 kB) above the first-order model. Back annotating the model with the more detailed model parameters would improve the analytical predictions even more.

Table 8
Summary of parameters for area model and MSE fit line.

	Model 512 k	MSE 512 k	Model 256 k	MSE 256 k	Model 128 k	MSE 128 k
Slope	10	10.1	10	9.53	10	8.93
Offset	1024	1060	512	544	256	287

Using our model, we also plotted the dual constraints, constant total neurons (10) and constant total area (20), shown in Fig. 30. As noted earlier, due to the rigid FPGA organization, we were only able to use a fraction of the total FPGA area (generally less than 50%). Thus, assuming a fraction of 35% of the total area, the maximum number of neural engines predicted by the analytical model is no more than 128 (if constrained to be a power of 2). This analytical prediction matches the empirical results given in Table 6, where architectures with $N > 128$ failed to map for $Q = 512$ and 256 thousand. In the case of $Q = 128$ thousand, the FPGA utilization greatly increases due to the jump from using block RAM to distributed RAM. Thus $N = 256$ and $N = 512$ are able to map.

6.5. Cost model validation

We also experimentally verified the frame time (hence communication delay) of the neural array using a cycle-accurate simulator (ModelSim). The results are shown in Fig. 31 together with the delay cost of the architecture, as predicted by the analytical model (27). With a high neural computation rate (NCR), the communication overhead is negligible. Thus the frame time is the dominant term in the delay cost of the architecture. Once again the analytical model and the experimental results match very closely.

7. Discussion

In this paper we have revisited Marr's levels of representation, emphasized the need for *layered* levels, the physical and the abstract, *not in a hierarchy* but rather in a recurrent relationship. Computational theory sits on top and provides the foundations for information processing architectures in brains and modern computing systems. Furthermore we provide a concrete design methodology for architectural exploration and understanding through "constraints that deconstrain" (Kirschner & Gerhart, 1998) through the use of a cost function optimization and parametric models at different levels of description that link software layers of abstraction to hardware layers, as well as delay and energy constrained by physical space.

For example, we have demonstrated the architecture of functional components, i.e. dynamical neurons from the simple

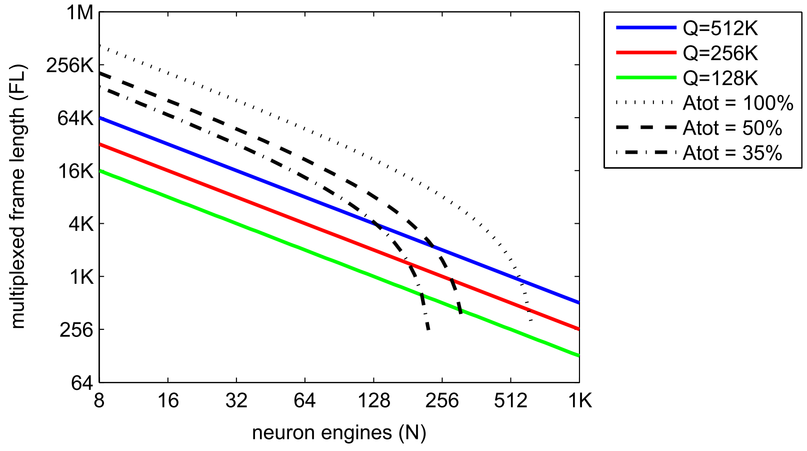


Fig. 30. Constant total neuron Q and constant total area A_{tot} curves.

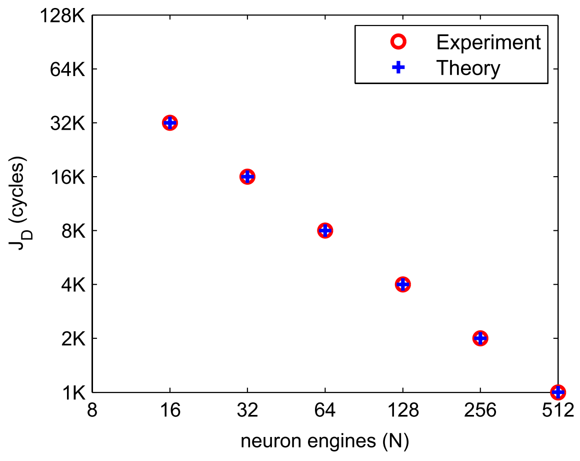


Fig. 31. J_D cost model validation: experiment vs. theory.

LIF to the more complex Izhikevich model. While the LIF neural model has been dominant in neuromorphic systems, we have demonstrated that neurons with more complex behaviors can be added to neuromorphic systems with relative ease. This functionality has been enabled by shifting from a predominantly analog silicon neuron paradigm, to a digital silicon neuron paradigm abstracting functionality. The digital processing level of implementation is also advantageous for its stability and high precision properties. Third, in terms of computational theory, the architecture optimization work contributes a methodology for analytical optimization of parallel architectures subject to the physical constraints of delay, energy, and area. Our approach goes beyond what was recently discussed in a paper on architecture, constraints, and behavior by Doyle and Csete (2011), where the architecture of the brain is discussed in terms of the physical layers in clothing juxtaposed to layers of abstraction in biological systems. The end result of our work at each of these layers of abstraction is the advent of nano-CMOS digital neuromorphic systems which incorporate both computation and learning, that can scale into the range of millions of neurons per chip.

Our goal was to explore the design of canonical structures, and hence we have begun at the level of spiking neurons and proceeded to design a neural architecture that is capable of supporting multiple functions. In the work presented here, we have focused on "generic" design for a neural architecture without consideration of the problem at hand. Partly our motivation is a belief that the brain is comprised of canonical computational structures that have evolved to solve problems ranging from early sensory processing to high-level cognition. The cortical column/STDP experiment in Section 5.5 involves multiple timescales and layers of representation. At the base level is the neural computation timescale (1 ms time ticks). The learning occurs at a longer/slower timescale. Then we used simulated annealing to learn the free parameters of the system (highest/longest timescale.) The architecture was optimized for the continual processing at the 1 ms timescale. The learning events occur less frequently and are less optimized in the data flow architecture (a read–modify–write action on memory to update the synaptic weights.) At the highest level, the simulated annealing algorithm was run on the host PC (in software), updating parameters infrequently at the longest timescale. The recent paper by Douglas and Martin linking behavior to the architecture of the cortical sheet demonstrates beautifully the links of information processing and computation under physical constraints, adding the developmental dimension in biological tissue (Douglas & Martin, 2012).

From an engineering perspective however, we may want to optimize an architecture for specific tasks such as natural language processing or the functionality of a specific silicon nervous system for robotic applications. To do so, we must further develop the "bottom up" models in such a way so that they can be parameterized to be simple functions of an application domain, for instance inference using graphical models. This optimization approach was recently demonstrated for automatic speech recognition (Cassidy, Yu, Zhou, & Andreou, 2011) and can be generalized for asymmetric spiking neural multiprocessors, defining an exciting direction for future work. Rapid design time, low cost, flexibility, digital precision, and stability are characteristics that favor digital implementation as a promising alternative to analog VLSI based approaches for designing neuromorphic systems. High computational power as well as low size, weight, and power (SWAP) are advantages that digital architectures offer over software based neuromorphic systems.

Some applications will favor minimizing power dissipation as opposed to performance. This is achieved by trading off faster than real-time performance for lower clock frequency and a lower supply voltage. In terms of the cost function in Eq. (1), this will be equivalent to setting the weighting parameter γ to a value greater than one. The latter will be applicable to a full custom CMOS design with digital circuits operating in subthreshold (Martin, Pouliquen, Andreou, & Fraeman, 1996; Vittoz, 2005) at a few MHz (Imam et al., 2012; Merolla et al., 2011) or even in the hundreds of kHz at ultra low voltages (Lotze & Manoli, 2012). According to the formula $P = CV^2f$, this will lead to significant power savings. Preliminary work in this direction and analysis of communication architectures in this application regime, suggests that a switched mesh architecture may be advantageous over the AER scheme used in this paper (Cassidy, Murray, Andreou, & Georgiou, 2011). It should be pointed out that switched capacitor and charged based analog circuits (Bamford & Giulioni, 2010; Noack, Mayr, Partzsch, Schultz, & Schuffny, 2012; Stanacevic & Cauwenberghs, 2005) could scale in deep sub-micron CMOS and hence it is possible that one would see hybrid architectures involving analog state holding elements, in synapses and even sophisticated neuron circuits (Folowosele et al., 2009).

Finally, our architecture does not appear to be a truly single-chip solution because of the two banks of SRAM external to the FPGA. However, these external SRAM banks can be readily integrated into a single chip solution using a 3D VLSI process, such as provided by Tezzaron (Tezzaron, 2011). In this approach, three VLSI tiers map to one tier for the parallel computation layer, including state memory and logic, and two tiers of SRAM for the synapse weights and re-mapper RAM. It is interesting to note that in the early days of neural networks, there were substantial efforts to develop all digital bio-inspired systems (Hammerstrom, 1995, 1998; Hammerstrom & Lulich, 1996; Wawrzynek et al., 1996; Wawrzynek, Asanovic, & Morgan, 1993). While these efforts were partially successful and visionary at that time, they were limited by technology and namely the inability to integrate substantial amounts of memory on the die. It would be interesting to see if the emerging 3D-CMOS technology (Tezzaron, 2011) or the IBM embedded DRAM technologies (Iyer et al., 2005) will help alleviate this problem.

Also promising directions in the nano-CMOS era involve exploiting more unconventional memory technologies such as nano wires and switches (Avizienis et al., 2012), memristors (Strukov, Snider, Stewart, & Williams, 2008), nano and CMOL architectures (Likharev, 2008) and system concepts that rely on these technologies (Gao & Hammerstrom, 2007; Gao, Zaveri, & Hammerstrom, 2008; Hammerstrom & Zaveri, 2009). The early pulse coupled mixed system designs that exploited timing to represent analog state (Murray, Del Corso, & Tarassenko, 1991) may also see a re-emergence in the context of stochastic computation and probabilistic event-based systems. It also seems increasingly likely that process scalable charged-based (Anthony, Ma Kohler, Kurtze, Kushne, & Sollner, 2008) and switched-capacitor circuits (Bamford, Hogri et al., 2012; Folowosele et al., 2009; Georgiou, Andreou, & Pouliquen, 2006; Noack et al., 2012) may play a role yielding true mixed-signal architectures that exploit the best of continuous value and discrete value representations.

8. Conclusions

We conclude this paper with a summary of ideas and results that we believe are relevant to the design of future neuromorphic systems aimed at commodity information processing at the scale.

First, we demonstrated the advantages of the digital neuron abstraction instead of the more biophysically realistic analog neurons. Second, to emulate the massive parallelism and high neuron density found in biology, we multiplex neural state computation in order to take advantage of high clock frequencies and dense SRAMs found in nanometer silicon technology. Third, we have presented an energy–delay cost function that is the first step towards developmental and evolutionary design principles that we believe have shaped biological neural systems. A highly quantitative formulation, the cost function also presents a qualitative view of parallel processing subject to physical constraints in layered information processing architectures. Delay cost is minimized by maximizing parallelism in the system. Under a fixed physical size constraint—we have employed area in this paper—the core area (processor and local memory) must be minimized in order to increase parallelism, balanced by the simultaneous goal of maximizing processing density through multiplexing. Both energy and delay costs are reduced by eliminating the serial fraction of the algorithm, as well as by minimizing costly off-chip accesses by keeping state local to each neural engine/processing unit.

Finally, the dawn of neuromorphic systems engineering at the scale has arrived. We have presented a design methodology for large-scale digital neuromorphic architectures for the nano-CMOS era. Our approach to the design of spiking neurons and STDP learning circuits relies on layered parallel computational structures where neurons are abstracted as arithmetic logic units and communication processors. We demonstrated the validity of the design methodology through the implementation of cortical development using spiking neurons and STDP learning and neural architecture optimization in state-of-the-art Field Programmable Gate Arrays (FPGAs). The implementation of neural arrays, spiking neurons as well as STDP learning rules have been prototyped in FPGAs. The latter are an ideal platform for investigating digital silicon neuron architectures, due to the inherent flexibility and high-level design methodology and offer a stepping stone towards full large-scale ASIC designs. With this work, we have taken a significant step towards realizing the goals of a new class of artificial, high-performance, energy-efficient, parallel computational architectures inspired by the brain.

Acknowledgments

This work was supported by the EU ICT Grant (ICT-231168-SCANDLE) ''acoustic SCene ANalysis for Detecting Living Entities'' and by ONR MURI (N000141010278) ''Figure-Ground Processing, Saliency and Guided Attention for Analysis of Large Natural Scenes''.

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings AFIPS spring joint computer conference* (pp. 483–485).

Anderson, C. H., & Eliasmith, C. (2004). *Neural engineering: computation, representation, and dynamics in neurobiological systems*. The MIT Press.

Andreou, A. G., & Kalayjian, Z. K. (2002). Polarization imaging: principles and integrated polarimeters. *IEEE Sensors Journal*, *2*(6), 566–575.

Andreou, A. G., Meitzler, R. C., Strohben, K., & Boahen, K. A. (1995). Analog VLSI neuromorphic image acquisition and pre-processing systems. *Neural Networks*, *8*(7/8), 1323–1347.

Anthony, M., Kohler, E., Kurtze, J., Kushner, L., & Sollner, G. (2008). A process-scalable low-power charge-domain 13-bit pipeline ADC. In *VLSI Circuits, 2008 IEEE Symposium on*(pp. 222–223).

APT-Group. (2011a). SpiNNaker a chip multiprocessor for neural network simulation. Tech. rep., The University of Manchester. January.

APT-Group. (2011b). SpiNNaker application programming interface. Tech. rep. Version 0.0, University of Manchester. December.

APT-Group. (2011c). SpiNNaker software specification and design. Tech. rep. version 0.0, The University of Manchester. December.

Arthur, J. V., & Boahen, K. A. (2010). Silicon–neuron design: a dynamical systems approach. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *58*(5), 1034–1043.

Arthur, J. V., Merolla, P. A., Akopyan, F., Alvarez, R., Cassidy, A. S., & Chandra, S. et al. (2012). Building block of a programmable neuromorphic substrate: a digital neurosynaptic core. In *Proceedings of the 2012 international joint conference on neural networks*, IJCNN (pp. 1–8).

Avizienis, A. V., Sillin, H. O., Martin-Olmos, C., Shieh, H. H., Aono, M., Stieg, A. Z., et al. (2012). Neuromorphic atomic switch networks. *PLoS One*, *7*(8), e42772.

Bamford, S. A., & Giulioni, M. (2010). Intimate mixing of analogue and digital signals in a field-programmable mixed-signal array with lopsided logic. In *Proceedings of the 2010 IEEE biomedical circuits and systems conference*, BioCAS 2010 (pp. 234–237).

Bamford, S. A., Hogri, R., Giovannucci, A., Taub, A. H., Herreros, I., Verschure, P. F. M. J., et al. (2012). A VLSI field-programmable mixed-signal array to perform neural signal processing and neural modeling in a prosthetic system. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *20*(4), 455–467.

Bamford, S. A., Murray, A. F., & Willshaw, D. J. (2012). Spike-timing-dependent plasticity with weight dependence evoked from physical constraints. *IEEE Transactions on Biomedical Circuits and Systems*, *6*(4), 385–398.

Bartolozzi, C., & Indiveri, G. (2007). Synaptic dynamics in analog VLSI. *Neural Computation*, *19*(10), 2581–2603.

Belhadj, B., Tomas, J., & Bornat, Y. (2009). Digital mapping of a realistic spike timing plasticity model for real-time neural simulations. In *Proceedings of the XXIV conference on design of circuits and integrated systems*, DCIS (pp. 1–6).

Bi, G. Q., & Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, *18*(24), 10464–10472.

Boahen, K. A. (1996). Retinomorphic vision systems. In *Proceedings of the 5th international conference on microelectronics for neural networks*, MicroNeuro (pp. 2–14).

Boahen, K. A. (2000). Point-to-point connectivity between neuromorphic chips using address events. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *47*(5), 416–434.

Boahen, K. A., & Andreou, A. G. (1992). A contrast sensitive silicon retina with reciprocal synapses. *Advances in Neural Information Processing Systems*, *3*, 764–772.

Boahen, K. A., Pouliquen, P. O., Andreou, A. G., & Jenkins, R. E. (1989). A heteroassociative memory using current-mode MOS analog VLSI circuits. *IEEE Transactions on Circuits and Systems*, *36*(5), 747–755.

Bofill-i Petit, A., & Murray, A. F. (2004). Synchrony detection and amplification by silicon neurons with STDP synapses. *IEEE Transactions on Neural Networks*, *15*(5), 1296–1304.

Bruederle, D., Petrovici, M. A., Vogginger, B., Ehrlich, M., Pfeil, T., Millner, S., et al. (2011). A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological Cybernetics*, *104*(4–5), 263–296.

Camuñas-Mesa, L. A., Zamarreño-Ramos, C., Linares-Barranco, A., Acosta-Jimenez, A. J., Serrano-Gotarredona, T., & Linares-Barranco, B. (2012). An event-driven multi-kernel convolution processor module for event-driven vision sensors. *IEEE Journal of Solid-State Circuits*, *47*(2), 504–517.

Cardoso, J., Diniz, P., & Weinhardt, M. (2010). Compiling for reconfigurable computing: a survey. *ACM Computing Surveys*, *42*(4), 13:1–13:64.

Carmona, R., Espejo, S., Dominguez-Castro, R., Rodriguez-Vazquez, A., Roska, T., & Kozek, T. et al. (1998). A 0.5 μm CMOS CNN analog random access memory chip for massive image processing. In *Proceedings of the 5th IEEE international workshop on cellular neural networks and their applications*, CNAA (pp. 271–276).

Cassidy, A. S., & Andreou, A. G. (2008). Dynamical digital silicon neurons. In *Proceedings of the 2007 IEEE biomedical circuits and systems conference*, BioCAS 2007 (pp. 289–292).

Cassidy, A. S., & Andreou, A. G. (2009). Analytical methods for the design and optimization of chip-multiprocessor architectures. In *Proceedings of the 43rd annual conference on information sciences and systems*, CISS (pp. 482–487).

Cassidy, A. S., & Andreou, A. G. (2012). Beyond Amdahl's law: an objective function that links multiprocessor performance gains to delay and energy. *IEEE Transactions on Computers*, *61*(8), 1110–1126.

Cassidy, A. S., Andreou, A. G., & Georgiou, J. (2011). A combinational digital logic approach to STDP. In *Proceedings of the 2011 IEEE international symposium on circuits and systems*, ISCAS (pp. 673–676).

Cassidy, A. S., Denham, S. L., Kanold, P. O., & Andreou, A. G. (2007). FPGA based silicon spiking neural array. In *Proceedings of the 2007 IEEE biomedical circuits and systems conference*, BioCAS 2007 (pp. 75–78).

Cassidy, A. S., Murray, T., Andreou, A. G., & Georgiou, J. (2011). Evaluating on-chip interconnects for low operating frequency silicon neuron arrays. In *Proceedings of the 2011 IEEE international symposium on circuits and systems*, ISCAS (pp. 2437–2440).

Cassidy, A. S., Yu, K., Zhou, H., & Andreou, A. G. (2011). A high-level analytical model for application specific CMP design exploration. In *Proceedings of the 2011 conference on design automation & test in Europe*, DATE'11 (pp. 1–6).

Cembrano, G., Rodriguez-Vazquez, A., Galan, R., Jimenez-Garrido, F., Espejo, S., & Dominguez-Castro, R. (2004). A 1000 FPS at 128 × 128 vision processor with 8-bit digitized I/O. *IEEE Journal of Solid-State Circuits*, *39*(7), 1044–1055.

Chicca, E., Badoni, D., Dante, V., D'Andreagiovanni, M., Salina, G., Carota, L., et al. (2003). A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory. *IEEE Transactions on Neural Networks*, *14*(5), 1297–1307.

Chklovskii, D. B., Mel, B. W., & Svoboda, K. (2004). Cortical rewiring and information storage. *Nature*, *431*(7010), 782–788.

Choudhary, S., Sloan, S., Fok, S., Neckar, A., Trautmann, E., & Gao, P. et al. (2012). Silicon neurons that compute. In *Proceedings of the 2012 international joint conference on neural networks*, IJCNN (pp. 1–8).

Chua, L. O., & Yang, L. (1988). Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, *35*(10), 1257–1272.

Churchland, P. S., & Sejnowski, T. J. (1988). Perspectives on cognitive neuroscience. *Science*, *242*(4879), 741–745.

Culurciello, E., Etienne-Cummings, R., & Boahen, K. A. (2003). A biomorphic digital image sensor. *IEEE Journal of Solid-State Circuits*, *38*(2), 281–294.

Das, A. (2000). Optimizing coverage in the cortex. *Nature Neuroscience*, *3*(8), 750–752.

de Garis, H., Shuo, C., Goertzel, B., & Ruiting, L. (2010). A world survey of artificial brain projects, Part I: large-scale brain simulations. *Neurocomputing*, *74*(1–3).

Diorio, C., Hasler, P. E., Minch, B. A., & Mead, C. A. (1996). A single-transistor silicon synapse. *IEEE Transactions on Electron Devices*, *43*(11), 1972–1980.

Diorio, C., Hasler, P. E., Minch, B. A., & Mead, C. A. (1997). A floating-gate MOS learning array with locally computed weight updates. *IEEE Transactions on Electron Devices*, *44*(12), 2281–2289.

Dominguez-Castro, R., Espejo, S., Rodriguez-Vazquez, A., Carmona, R., Foldesy, P., Zarandy, A., et al. (1997). A 0.8 μm CMOS two-dimensional programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage. *IEEE Journal of Solid-State Circuits*, *32*(7), 1013–1026.

Douglas, R. J., & Martin, K. A. C. (2012). Behavioral architecture of the cortical sheet. *Current Biology*, *22*(24), R1033–R1038.

Doyle, J. C., & Csete, M. (2011). Architecture, constraints, and behavior. *Proceedings of the National Academy of Sciences of the United States of America*, *108*(Suppl. 3), 15624–15630.

Eliasmith, C., & Stewart, T. C. (2011). Nengo and the neural engineering framework: connecting cognitive theory to neuroscience. In *Proceedings of the 33rd annual meeting of the cognitive science society* (pp. 1–2).

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science*, *338*(6111), 1202–1205.

Etienne-Cummings, R., Kalayjian, Z. K., & Donghui, C. (2001). A programmable focal-plane MIMD image processor chip. *IEEE Journal of Solid-State Circuits*, *36*(1), 64–73.

Fasnacht, D. B., & Indiveri, G. (2011). A PCI based high-fanout AER mapper with 2 GB RAM look-up table, 0.8 us latency and 66 MHz output event-rate. In *Proceedings of the 45th annual conference on information sciences and systems*, CISS (pp. 1–6).

Federico, M. D., Mandolesi, P. S., Julian, P., & Andreou, A. G. (2008). Experimental results of simplicial CNN digital pixel processor. *IET Electronics Letters*, *44*(1), 27–29.

Folowosele, F. O., Harrison, A., Cassidy, A. S., Andreou, A. G., Etienne-Cummings, R., & Mihalas, S. et al. (2009). A switched capacitor implementation of the generalized linear integrate-and-fire neuron. In *Proceedings of the 2009 IEEE international symposium on circuits and systems*, ISCAS (pp. 2149–2152).

Galluppi, F., Davies, S., Furber, S. B., Stewart, T. C., & Eliasmith, C. (2012). Real time on-chip implementation of dynamical systems with spiking neurons. In *Proceedings of the 2012 international joint conference on neural networks*, IJCNN (pp. 1–8).

Gao, C., & Hammerstrom, D. W. (2007). Cortical models onto CMOL and CMOS—architectures and performance/price. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *54*(11), 2502–2515.

Gao, C., Zaveri, M. S., & Hammerstrom, D. W. (2008). CMOS/CMOL architectures for spiking cortical column. In *Proceedings of the 2008 international joint conference on neural networks*, IJCNN (pp. 2441–2448).

Genov, R., & Cauwenberghs, G. (2001). Charge-mode parallel architecture for vector–matrix multiplication. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *48*(10), 930–936.

Georgiou, J., & Andreou, A. G. (2006). High-speed, address-encoding arbiter architecture. *IET Electronics Letters*, *42*(3), 170–171.

Georgiou, J., & Andreou, A. G. (2007). Address-data event representation for communication in multichip neuromorphic system architectures. *IET Electronics Letters*, *43*(14), 767.

Georgiou, J., Andreou, A. G., & Pouliquen, P. O. (2006). A mixed analog/digital asynchronous processor for cortical computations in 3D SOI-CMOS. In *Proceedings of the 2006 IEEE international symposium on circuits and systems*, ISCAS (pp. 4955–4958).

Giulioni, M., Camilleri, P., Mattia, M., Dante, V., Braun, J., & Del Giudice, P. (2011). Robust working memory in an asynchronously spiking neural network realized with neuromorphic VLSI. *Frontiers in Neuroscience*, *5*, 149.

Goldberg, D. H., Cauwenberghs, G., & Andreou, A. G. (2001a). Analog VLSI spiking neural network with address domain probabilistic synapses. In *Proceedings of the 2001 IEEE international symposium on circuits and systems*, ISCAS (pp. 241–244).

Goldberg, D. H., Cauwenberghs, G., & Andreou, A. G. (2001b). Probabilistic synaptic weighting in a reconfigurable network of VLSI integrate-and-fire neurons. *Neural Networks*, *14*, 781–793.

Goodman, D. F. M., & Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*, *3*(2), 192–197.

GSI, (2011). GSI technology home. www.gsitechnology.com. February.

Hall, T. S., Twigg, C. M., Gray, J. D., Hasler, P. E., & Anderson, D. V. (2005). Large-scale field-programmable analog arrays for analog signal processing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *52*(11), 2298–2307.

Hammerstrom, D. W. (1995). A digital VLSI architecture for real-world applications. In *An introduction to neural and electronic networks* (2nd ed.) (pp. 335–358). Academic Press.

Hammerstrom, D. W. (1998). Digital VLSI for neural networks. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 304–309). The MIT Press.

Hammerstrom, D. W., & Lulich, D. P. (1996). Image processing using one-dimensional processor arrays. *Proceedings of the IEEE*, *84*(7), 1005–1018.

Hammerstrom, D. W., & Zaveri, M. S. (2009). Prospects for building cortex-scale CMOL/CMOS circuits: a design space exploration. In *Proceedings of the 27th Norchip conference*, NORCHIP-2009 (pp. 1–8).

Harrison, A., Özgün, R., Lin, J., Andreou, A. G., & Etienne-Cummings, R. (2010). A spike based 3D imager chip using a mixed mode encoding readout. In *Proceedings of the 2010 IEEE biomedical circuits and systems conference*, BioCAS 2010 (pp. 190–193).

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology: London*, *117*(4), 500–544.

Horiuchi, T. K., & Koch, C. (1999). Analog VLSI-based modeling of the primate oculomotor system. *Neural Computation*, *11*(1), 243–265.

Hsin, C., Saighi, S., Buhry, L., & Renaud, S. (2010). Real-time simulation of biologically realistic stochastic neurons in VLSI. *IEEE Transactions on Neural Networks*, *21*(9), 1511–1517.

Hubel, D. H., & Wiesel, T. N. (1977). Ferrier lecture: functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society B: Biological Sciences*, *198*(1130), 1–59.

Imam, N., Akopyan, F., Arthur, J. V., Merolla, P. A., Manohar, R., & Modha, D. S. (2012). A digital neurosynaptic core using event-driven QDI circuits. In *Proceedings of the 18th IEEE international symposium on asynchronous circuits and systems*, ASYNC (pp. 25–32).

Indiveri, G., Chicca, E., & Douglas, R. J. (2004). A VLSI reconfigurable network of integrate-and-fire neurons with spike-based learning synapses. In *Proceedings of the 2004 European symposium on artificial neural networks*, ESANN (pp. 405–410).

Indiveri, G., Chicca, E., & Douglas, R. J. (2006). A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, *17*(1), 211–221.

Iyer, S. S., Barth, J. E., Jr., Parries, P. C., Norum, J. P., Rice, J. P., Logan, L. R., et al. (2005). Embedded DRAM: technology platform for the Blue Gene/L chip. *IBM Journal of Research and Development*, *49*(2), 333–350.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070.

Kanold, P. O., & Shatz, C. (2006). Subplate neurons regulate maturation of cortical inhibition and outcome of ocular dominance plasticity. *Neuron*, *51*(5), 627–638.

Karakiewicz, R., Genov, R., & Cauwenberghs, G. (2007). 480-GMACS/mW resonant adiabatic mixed-signal processor array for charge-based pattern recognition. *IEEE Journal of Solid-State Circuits*, *42*(11), 2573–2584.

Opal-Kelly, (2012). FPGA USB modules. www.opalkelly.com.

Kestur, S., Park, M. S., Sabarad, J., Dantara, D., Narayanan, V., & Chen, Y. et al. (2012). Emulating mammalian vision on reconfigurable hardware. In *2012 IEEE international symposium on field-programmable custom computing machines* (pp. 141–148).

Khan, M., Lester, D., Plana, L., Rast, A., Jin, X., & Painkras, E. et al. (2008). SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor. In *Proceedings of the 2008 international joint conference on neural networks*, IJCNN (pp. 2850–2857).

Kirschner, M., & Gerhart, J. (1998). Evolvability. *Proceedings of the National Academy of Sciences of the United States of America*, *95*(15), 8420–8427.

Koickal, T. J., Gouveia, L. C., & Hamilton, A. (2009). A programmable spike-timing based circuit block for reconfigurable neuromorphic computing. *Neurocomputing*, *72*, 3609–3616.

Kumar, N., Himmelbauer, W., Cauwenberghs, G., & Andreou, A. G. (1998). An analog VLSI chip with asynchronous interface for auditory feature extraction. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *45*(5), 600–606.

La Rosa, M., Caruso, E., Fortuna, L., Frasca, M., Occhipinti, L., & Rivoli, F. (2005). Neuronal dynamics on FPGA: Izhikevich's model. In *Proceedings of SPIE: bioengineered and bioinspired systems II* (pp. 87–94). SPIE.

Laughlin, S. B., & Sejnowski, T. J. (2003). Communication in neuronal networks. *Science*, *301*(5641), 1870–1874.

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128 × 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, *43*(2), 566–576.

Likharev, K. K. (2008). CMOL: second life for silicon? *Microelectronics Journal*, *39*(2), 177–183.

Lin, J. H., & Boahen, K. A. (2009). A delay-insensitive address-event link. In *15th IEEE symposium on asynchronous circuits and systems*, ASYNC (pp. 55–62).

Lin, J. H., Pouliquen, P. O., Andreou, A. G., Goldberg, A. C., & Rizk, C. G. (2012). Flexible readout and integration sensors (FRIS): a bio-inspired, system-on-chip, event based readout architecture. In *Proceedings of SPIE: infrared technology and applications XXXVIII conference* (pp. 8353–1N).

Lin, J. H., Pouliquen, P. O., Goldberg, A. C., Rizk, C. G., & Andreou, A. G. (2011). A bio-inspired event-driven architecture with pixel-level A/D conversion and non-uniformity correction. In *Proceedings of the 45th annual conference on information sciences and systems*, CISS (pp. 1–6).

Liu, S.-C., & Mockel, R. (2008). Temporally learning floating-gate VLSI synapses. In *Proceedings of the 2008 IEEE international symposium on circuits and systems*, ISCAS (pp. 2154–2157).

Lotze, N., & Manoli, Y. (2012). A 62 mV 0.13 μm CMOS standard-cell-based design technique using Schmitt-trigger logic. *IEEE Journal of Solid-State Circuits*, *47*(1), 47–60.

Mahowald, M. (1992). VLSI analogs of neuronal visual processing: a synthesis of form and function. Ph.D. Thesis, Ph.D. Dissertation, California Institute of Technology.

Mandolesi, P. S., Julian, P., & Andreou, A. G. (2004). A scalable and programmable simplicial CNN digital pixel processor architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *51*(5), 988–996.

Mandolesi, P. S., Julian, P., & Andreou, A. G. (2006). A simplicial CNN visual processor in 3D SOI-CMOS. In *Proceedings of the 2006 IEEE international symposium on circuits and systems*, ISCAS (pp. 1311–1314).

Markram, H. (2011). Human brain project. http://www.humanbrainproject.eu/.

Markram, H., Gerstner, W., & Sjöström, P. J. (2011). A history of spike-timing-dependent plasticity. *Frontiers in Synaptic Neuroscience*, *3*(4), 1–24.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. W.H. Freeman and Company.

Martin, M. N., Pouliquen, P. O., Andreou, A. G., & Fraeman, M. E. (1996). Current-mode differential logic circuits for low power digital systems. In *Proceedings of the 39th midwest symposium on circuits and systems*, MWSCAS (pp. 183–186).

McClelland, J. L., Rumelhardt, D. R., & Group, P. R. (1987). *Parallel distributed processing, psychological and biological models—Vol. 2*. The MIT Press.

Mead, C. A. (1989). *Analog VLSI and neural systems*. Addison-Wesley Publishers.

Mead, C. A. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636.

Mead, C. A., & Conway, L. (1979). *Introduction to VLSI systems*. Addison-Wesley Publishers.

Merolla, P. A., Arthur, J. V., Akopyan, F., Imam, N., Manohar, R., & Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm. In *2011 IEEE custom integrated circuits conference*, CICC 2011. (pp. 1–4). IEEE.

Mihalas, S., & Niebur, E. (2009). A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. *Neural Computation*, *21*(3), 704–718.

Mill, R., Sheik, S., Indiveri, G., & Denham, S. L. (2011). A model for stimulus-specific adaptation in neuromorphic analog VLSI. *IEEE Transactions on Biomedical Circuits and Systems*, 1–8.

Misra, J., & Saha, I. (2010). Artificial neural networks in hardware a survey of two decades of progress. *Neurocomputing*, *74*(1–3), 239–255.

Modha, D. S., Ananthanarayanan, R., Esser, S. K., Ndirango, A., Sherbondy, A. J., & Singh, R. (2011). Cognitive computing. *Communications of the ACM*, *54*(8), 62.

Murray, A. F., Del Corso, D., & Tarassenko, L. (1991). Pulse-stream VLSI neural networks mixing analog and digital techniques. *IEEE Transactions on Neural Networks*, *2*(2), 193–204.

Nallatech, (2008). FPGA accelerated computing. www.nallatech.com. December.

Neumann, J. v. (1958). *The computer and the brain*. Yale University Press.

Noack, M., Mayr, C., Partzsch, J., Schultz, M., & Schuffny, R. (2012). A switched-capacitor implementation of short-term synaptic dynamics. In *Proceedings of the 19th international conference on mixed design of integrated circuits and systems*, MIXDES 2012. (pp. 214–218). IEEE.

Pardo, F., Dierickx, B., & Scheffer, D. (1998). Space-variant nonorthogonal structure CMOS image sensor design. *IEEE Journal of Solid-State Circuits*, *33*(6), 842–849.

Pearson, M., Pipe, A., Mitchinson, B., Melhush, G., Gilhespy, I., & Nibouche, M. (2007). Implementing spiking neural networks for real-time signal-processing and control applications: a model-validated FPGA approach. *IEEE Transactions on Neural Networks*, *18*(5), 1472–1487.

Pham, P.-H., Jelaca, D., Farabet, C., Martini, B., LeCun, Y., & Culurciello, E. (2012). NeuFlow: dataflow vision processing system-on-a-chip. In *Proceedings of the 55th midwest symposium on circuits and systems*, MWSCAS (pp. 1044–1047).

Posch, C., Matolin, D., & Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, *46*(1), 259–275.

Pouliquen, P. O., Andreou, A. G., & Strohben, K. (1997). Winner-takes-all associative memory: a Hamming distance vector quantizer. *Analog Integrated Circuits and Signal Processing*, *13*(1–2).

Preissl, R., Wong, T. M., Datta, P., Flickner, M., Singh, R., Esser, S. K., et al. (2012). Compass: a scalable simulator for an architecture for cognitive computing. In *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'12. (pp. 1–11). IEEE Computer Society.

Ramakrishnan, S., Hasler, P. E., & Gordon, C. (2012). Floating gate synapses with spike-time-dependent plasticity. *IEEE Transactions on Biomedical Circuits and Systems*, *5*(3), 244–252.

Renaud, S., Tomas, J., Lewis, N., Bornat, Y., Daouzli, A., Rudolph, M., et al. (2010). PAX: a mixed hardware/software simulation platform for spiking neural networks. *Neural Networks*, *23*(7), 905–916.

Rice, K. L., Bhuiyan, M. A., Taha, T. M., Vutsinas, C. N., & Smith, M. C. (2009). FPGA implementation of Izhikevich spiking neurons for character recognition. In *Proceedings of the 2009 international conference on reconfigurable computing and FPGAs*, ReConFig'09. (pp. 451–456). IEEE.

Roska, T., & Chua, L. O. (1993). The CNN universal machine: an analogic array computer. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *40*(3), 163–173.

Rumelhart, D. E., McClelland, J. L., & Group, P. R. (1987). *Parallel distributed processing, foundations—Vol. 1*. The MIT Press.

Saighi, S., Bornat, Y., Tomas, J., Le Masson, G., & Renaud, S. (2010). A library of analog operators based on the Hodgkin–Huxley formalism for the design of tunable, real-time, silicon neurons. *IEEE Transactions on Biomedical Circuits and Systems*, *5*(1), 3–19.

Schemmel, J., Grubl, A., Meier, K., & Mueller, E. (2006). Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 international joint conference on neural networks*, IJCNN (pp. 1–6).

Schreiber, S., Machens, C. K., Herz, A. V. M., & Laughlin, S. B. (2002). Energy-efficient coding with discrete stochastic events. *Neural Computation*, *14*(6), 1323–1346.

Serrano-Gotarredona, T., Andreou, A. G., & Linares-Barranco, B. (1999). AER image filtering architecture for vision-processing systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, *46*(9), 1064–1071.

Serrano-Gotarredona, T., Oster, M., Lichtsteiner, P., Linares-Barranco, A., Paz-Vicente, R., Gomez-Rodriguez, F., et al. (2009). CAVIAR: A 45 k neuron, 5 M synapse, 12 G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking. *IEEE Transactions on Neural Networks*, *20*(9), 1417–1438.

Silver, R., Boahen, K. A., Grillner, S., Kopell, N., & Olsen, K. L. (2007). Neurotech for neuroscience: unifying concepts, organizing principles, and emerging tools. *The Journal of Neuroscience*, *27*(44), 11807–11819.

Sivilotti, M. A. (1991). Wiring considerations in analog VLSI systems, with application to field programmable networks. Ph.D. Thesis, Ph.D. Dissertation, California Institute of Technology.

Song, S., & Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*(9), 919–926.

Stanacevic, M., & Cauwenberghs, G. (2005). Micropower gradient flow acoustic localizer. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *52*(10), 2148–2157.

Stepanyants, A., Hof, P. R., & Chklovskii, D. B. (2002). Geometry and structural plasticity of synaptic connectivity. *Neuron*, *34*(2), 275–288.

Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. *Nature*, *453*(7191), 80–83.

Swindale, N. V., Shoham, D., Grinvald, A., Bonhoeffer, T., & Hübener, M. (2000). Visual cortex maps are optimized for uniform coverage. *Nature Neuroscience*, *3*(8), 822–826.

Tezzaron, (2011). Tezzaron semiconductor. www.tezzaron.com. February.

Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London: Series B, Biological Sciences*, *237*(641), 37–72.

Varshney, L. R., Sjöström, P. J., & Chklovskii, D. B. (2006). Optimal information storage in noisy synapses under resource constraints. *Neuron*, *52*(3), 409–423.

Vittoz, E. A. (2005). Weak inversion for ultimate low-power logic. In *Low-power CMOS circuits: technology, logic design and CAD tools* (pp. 1–18). CRC Press.

Vogelstein, R. J., Mallik, U., Culurciello, E., Cauwenberghs, G., & Etienne-Cummings, R. (2007). A multichip neuromorphic system for spike-based visual information processing. *Neural Computation*, *19*(9), 2281–2300.

Wawrzynek, J., Asanovic, K., Kingsbury, B., Johnson, D., Beck, J., & Morgan, N. (1996). Spert-II: a vector microprocessor system. *IEEE Computer*, *29*(3), 79–86.

Wawrzynek, J., Asanovic, K., & Morgan, N. (1993). The design of a neuro-microprocessor. *IEEE Transactions on Neural Networks*, *4*(3), 394–399.

Wen, Q., & Chklovskii, D. B. (2008). A cost-benefit analysis of neuronal morphology. *Journal of Neurophysiology*, *99*(5), 2320–2328.

Wen, Q., Stepanyants, A., Elston, G. N., Grosberg, A. Y., & Chklovskii, D. B. (2009). Maximization of the connectivity repertoire as a statistical principle governing the shapes of dendritic arbors. *Proceedings of the National Academy of Sciences of the United States of America*, *106*(30), 12536–12541.

Wijekoon, J. H. B., & Dudek, P. (2006). Simple analogue VLSI circuit of a cortical neuron. In *Proceedings of the 13th IEEE international conference on electronics, circuits, and systems*, ICECS (pp. 1344–1347).

Wolff, L. B., & Andreou, A. G. (1995). Polarization camera sensors. *Image and Vision Computing*, *13*, 497–510.

Xilinx, (2011). www.xilinx.com. February.

Yu, T., Sejnowski, T. J., & Cauwenberghs, G. (2011). Biophysical neural spiking, bursting, and excitability dynamics in reconfigurable analog VLSI. *IEEE Transactions on Biomedical Circuits and Systems*, *5*(5), 420–429.