A system design perspective on neuromorphic computer processors

Garrett S Rose, Mst Shamim Ara Shawkat, Adam Z Foshie, John J Murray VI, and Md Musabbir Adnan

1 Min H Kao Department of Electrical Engineering & Computer Science, The University of Tennessee, Knoxville, TN 37996, United States of America
2 Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33174, United States of America

* Author to whom any correspondence should be addressed.
E-mail: garose@utk.edu

Keywords: design, perspective, neuromorphic, computers, processors

Abstract
Neuromorphic computing has become an attractive candidate for emerging computing platforms. It requires an architectural perspective, meaning the topology or hyperparameters of a neural network is key to realizing sound accuracy and performance in neural networks. However, these network architectures must be executed on some form of computer processor. For machine learning, this is often done with conventional computer processing units, graphics processor units, or some combination thereof. A neuromorphic computer processor or neuroprocessor, in the context of this paper, is a hardware system that has been designed and optimized for executing neural networks of one flavor or another. Here, we review the history of neuromorphic computing and consider various spiking neuroprocessor designs that have emerged over the years. The aim of this paper is to identify emerging trends and techniques in the design of such brain-inspired neuroprocessor computer systems.

1. Introduction
For as long as engineers have been developing calculating or computing machines, researchers have also sought to unlock the secrets of the brain and develop machines that think. Many artificial intelligence or machine learning systems today are far from true thinking machines. Even so, machine learning has responded in recent years to real needs for researchers and society to better handle challenges associated with big data processing. Unlike classic computer systems, brain-inspired architectures are able to discern useful information from imperfect or noisy data. Deep neural networks (DNNs) and other machine learning approaches have proven incredibly effective at classifying features in complex data and image recognition. So much so that some machine learning systems are now better than human beings at things such as recognizing sloppily written numeric digits. While still not thinking machines, machine learning has provided very powerful tools for modern society.

Neuromorphic computing has also emerged over the past few decades as another approach to constructing some form of thinking machines. Where machine learning approaches are bio-inspired to some extent, DNNs and convolutional neural networks (CNNs) place little emphasis on bio-mimicry and instead focus on more optimized signal processing. Neuromorphic systems, on the other hand, are typically more detailed in their representation of brain-like behavior, often emulating many complex behaviors observed in biological neural systems. Of course, the extent to which neuromorphic designs mimic biology varies system to system. A common feature, as we see it, is that neuromorphic systems process data in the form of streams of electric pulses or spikes. This spiking representation of data is based entirely on how information is represented in the brain. Thus, at least in how data is represented, neuromorphic systems are generally aimed toward more brain-like processing, even brain-mimicry, as compared to more conventional machine learning approaches.

There are expected advantages and applications to computing with spikes. For starters, by more directly emulating brain behavior, some neuromorphic computing systems are used to help neuroscientists learn more
about biological brains. Unlocking the secrets of brain behavior can in turn be important for developing more efficient computing systems in the future. For example, it is well known that the human brain consumes around 20 W of power to perform tasks for which a more conventional machine learning system would require kilowatts of power. The brain, which does process with spiking information, is incredibly energy-efficient. Thus, as neuromorphic circuits and architectures have emerged, so have more generalized neuromorphic processors or neuroprocessors.

Here, we consider a neuroprocessor as a neuromorphic system that is in some way reconfigurable or programmable. This reconfigurability provides the ability for the neuroprocessor to execute a range of neuromorphic applications, where the system is reconfigured to represent specific neural networks of interest. This paper provides a computer engineering perspective to neuromorphic computing, specifically focusing on techniques and considerations for the design of neuromorphic processors that implement spiking neural networks (SNNs).

While analog approaches do remain popular in neuromorphic system design, the term ‘neuromorphic’ has broadened to include digital or at least mixed-signal implementations. In fact, the definition of neuromorphic is often hard to pin down as it has certainly evolved from the early concepts introduced by Carver Mead. Bio-realism is a concept, but it could be bio-realism in the behavior, representing intrinsic mimicry of what biology does even if the tools used are inherently digital. This is different from Mead’s plea that we use the analog devices we have as they are. Furthermore, digital and even many mixed-signal implementations operate at accelerated time scales, not biologically plausible timescales (ms or s). From this perspective, we could categorize these more digital neuroprocessors as ‘neuromorphic emulators,’ in the sense that digital circuits, in some cases full processors, are used to emulate what can be achieved by exploiting device physics at a lower level. Our take is that any neuroprocessor is a sort of emulator in the sense that it emulates biology. Thus, we will draw distinctions between different architectures based on characteristics such as speed (accelerated vs biologically plausible timing), complexity, and mode of operation (analog, mixed-signal, digital).

The remainder of this paper is organized as follows. We first briefly review machine learning accelerators (MLAs), which we see as closely related to neuroprocessors but distinct. In section 3, we provide some history of neuromorphic computing with emphasis on the key components typically used to implement neuroprocessors. Section 4 provides discussions for several example neuromorphic processor designs, ranging from asynchronous, analog implementations to synchronous, digital systems. Interconnect between programmable or reconfigurable neuromorphic cores is a critical and yet often overlooked aspect of neuroprocessor design. As such, we provide some discussion of various approaches to neuroprocessor connectivity in section 5. Finally, we conclude this work with some further discussion on the field, including key research opportunities, in section 7. Here we also provide some comparisons across these various example designs as a way of identifying key trends.

2. On machine learning accelerators

An important advance in computer architectures that helped accelerate the recent machine learning revolution was the use of the graphics processing unit (GPU) in CNNs, often as part of DNNs [1, 2]. Given the power of GPUs, they have also been integrated into several modern high-performance computing systems. GPUs are specifically optimized for matrix algebra computations, common in graphics processing but also many machine learning approaches. For example, a fully connected neural network layer in a DNN can be viewed as a dense matrix of synaptic weights to be multiplied with input vectors. The popular training approach known as backpropagation is also an algorithm consisting of several matrix and vector operations [3]. Thus, it is no surprise that the emergence of the GPU has played a central role in ushering in the machine learning revolution.

In addition to leveraging GPUs, MLAs have also emerged which are specifically tailored to the needs of deep learning and other machine learning approaches. For example, the tensor processing unit (TPU) was developed by Google as a system-on-chip architecture, specifically designed to accelerate neural network style machine learning [4, 5]. It is also worth noting that a TPU based system can easily be trained using TensorFlow software, also from Google.

MLAs built from field-programmable gate arrays (FPGAs) have also emerged in recent years [6–8]. An FPGA provides bit-level configurability of digital operators that can be implemented with significant parallelism. Such capability provides an opportunity to implement application-specific neural network architectures at the logic level, leaving out any hardware systems not necessary for that application. Such flexibility and performance advantages are certainly attractive. However, reconfigurable computing based approaches, such as FPGA-based acceleration for machine learning, often prove difficult from a programming perspective.

Emerging device technologies, such as phase change memory (PCM) and memristors, have also been explored as options for dense, energy-efficient matrix–matrix or matrix–vector multipliers. For example, in-memory computing approaches built from memristor-based resistive RAM (RRAM) array architectures
have been explored for their ability to implement DNNs in a high density, energy-efficient package. One implementation was presented in [9] where an RRAM-based architecture was successfully implemented and validated using the ImageNet dataset [2]. A pipelined implementation of an RRAM-based DNN accelerator was presented in [10] that showed potential for significant speedup and energy-efficiency relative to GPU based MLAs. Luo and Yu recently considered approaches for accelerating deep learning using non-volatile and volatile memristive elements [11]. In related work, approaches to pruning memristor-based DNN accelerators were considered as a way to further improve the density and energy-efficiency of such systems [12].

From the perspective of neuromorphic computing, MLAs occupy a class of their own. In fact, an MLA architecture is as distinct from a neuromorphic system architecture as machine learning is from neuromorphic computing. The two are related in that both are bio-inspired in some fashion. However, machine learning operations based on matrix algebra are not particularly bio-realistic. The mammalian brain computes with spikes in a dense network of both feed-forward and recurrent connections. Neuromorphic computing typically aims to be more bio-realistic from the perspective that computations are performed on information encoded as spikes and often includes neuronal features that more directly mimic behavior observed in biological systems.

3. Elements of a neuromorphic processor

The remainder of this paper is primarily concerned with the architectural and microarchitectural components of neuromorphic systems which are designed to more closely adhere to neurobiology than their MLA counterparts. However, our understanding of neurobiology is not perfect and as such our artificial neuromorphic systems cannot be perfect representations of biology. Moreover, even if we did have a more complete understanding of the brain, we likely still would not mimic every aspect of the biological brain in our engineered systems. With this perspective, neuromorphic computing could be said to draw inspiration from biology in instances where some functions are thought to improve computational performance. Note that this is not entirely different from machine learning, which indeed does draw inspiration from biology. However, neuromorphic computing is often still distinct in the insistence of holding fast to more bio-mimicry on some fronts, such as computing with spiking information. With that, it is important to take a quick look into some of the details of biological neural networks, at least as experts understand them today.

3.1. Biological neural networks as we understand them

Neural networks in biology are primarily networks of neurons (figure 1), tiny cells that respond to spiking electrochemical stimuli by generating their own spiking outputs. Between connected neurons is something known as the synaptic cleft, often simply referred to as a synapse. These synapses are responsible for regulating the impact an incoming signal will have on the post-synaptic neuron. This regulation is often modeled as a weight that either allows spiking signals to pass through virtually unobstructed or provides some resistance that essentially dulls the signal strength. In addition to the neuron and synapse, the connection between two neurons can be further compartmentalized by dendrites, small tentacles that receive information from the synaptic cleft and into a neuron, and axons, longer tentacles from the neurons that drive information through synapses and into other neurons. The length of an axon makes it analogous to a transmission line that will provide some delay and attenuation of signals as they propagate from one neuron to another. While axonal delay may or may not be modeled in an artificial system, any neuromorphic architecture is expected to at least be comprised of neuron and synapse models.

Biological neurons transmit signals through complex chemical processes in which neurotransmitters modulate their electrical potential [13]. A spiking neuron can be modeled as a comparator circuit that compares an input voltage to a preset threshold, and if the input voltage is over the threshold, a voltage pulse is produced. The circuit will continue to spike as long as the input voltage remains above the threshold. These spikes are usually generated at a frequency of a few milliseconds in biological systems. It is generally designed to mimic this firing rate as closely as possible, although some proposed circuits operate faster.

There are a variety of approaches to modeling neurons based on the theory that spiking is biologically accurate to various degrees. Some commonly used models are Hodgkin–Huxley [16], Izhikevich model [17], and the leaky integrate and fire (LIF) model [18]. Although Hodgkin–Huxley neuron models enable researchers to study brain functions with great detail, thus allowing them to replicate them with high precision in hardware, circuit implementations tend to be expensive in terms of power and chip area [19]. By compromising the biological functionality with simpler circuits and achieving better energy and area efficiency in hardware implementations, the Izhikevich model [17] is more straightforward to implement. Mead described the LIF model using an axon hillock circuit [14], illustrated in figure 2. The input current charges the neural circuit membrane capacitor, which determines when the switching threshold is reached and the output moves toward the power rail voltage in this axon hillock circuit. Once a spike has been generated, a feedback circuit prevents the membrane capacitor from accumulating any charge and causes the amplifier to revert back to ground.
Figure 1. An illustration of two biological neurons, showing some of the parts of the neuron. A pre-synaptic neuron will fire through the axon hillock, transmitting the spike(s) through the axon toward the synaptic cleft. At the synaptic cleft, neurotransmitters are released across the cleft and received into the corresponding dendrite of the post-synaptic neuron, where it is then accumulated at the soma of the same neuron.

Figure 2. Axon hillock neuron circuit, constructed from a capacitive integrator that carries the membrane potential $V_{\text{mem}}$ driving two cascaded inverters [14, 15].

An important point to make here is that the cells and chemistry that make our brains what they are tend to be quite complex. It is likely that this very complexity of the brain is part of what makes it so powerful. However, an artificial system cannot model everything and any engineered system must be abstract enough, thus simplified enough, that human designers can readily optimize the inner workings. All this means is that neuromorphic computing remains an infant field with much room to grow as we continually work to better understand the complexity of the brain and how best to harness that understanding.

3.2. Neuromorphic hardware: the early years

An early architecture for artificial intelligence was Rosenblatt’s single-layer neural network, dubbed the perceptron [20, 21]. The perceptron combined the thresholding activation function of the McCulloch–Pitts neuron [22] with Hebb’s findings that inputs are multiplied by weights which can in turn be adjusted or trained [23]. A hardware implementation of the perceptron algorithm was also realized in the form of the Mark I perceptron machine. The Mark I is notable as one of the earlier computer systems specifically implemented as a hardware based neural network [24].
The primary shortcoming of the perceptron model was that it was a single-layer network [25]. More specifically, as a single-layer network the perceptron was able to classify linearly separable objects but could not classify anything that was not linearly separable. The prime example of this shortcoming was the XOR function which produces logic ‘1’ outputs for non-adjacent input terms ‘01’ and ‘10’ (both bits must switch). A single perceptron fails any attempt at learning XOR behavior. However, multi-layer neural networks, originally termed multi-layer perceptrons (MLPs), are capable of classifying more complex datasets. Fast forward a few decades and this multi-layer, hierarchical approach to neural network construction is a critical feature of modern DNNs.

Despite shortcomings that led to the first AI winter, the perceptron was a significant milestone on the road to many approaches to machine learning common today, especially DNNs. Further, the basic building blocks of the perceptron, the McCulloch–Pitts neuron and the tunable synaptic weight, are often present in any neural network, including what we commonly refer to as neuromorphic systems. Notable in many of the arguments against the perceptron as a model whereby computers could be made to reason was the realization that the perceptron itself was likely too simple. The activation function, based on that McCulloch–Pitts neuron, was simply a thresholding neuron. The machine learning community has since utilized more complex nonlinear activation functions, including the sigmoid function, tanh, the softmax function, and the rectified linear unit (ReLU). Nonlinearity was found to be a key ingredient in improving neural network performance. However, that simple thresholding function remains attractive because it is simple. In other words, where a hardware implementation of a thresholding function is straightforward, more complex activation functions could be more costly. Thus, neuromorphic computing somewhat diverges from machine learning in where nonlinearity is injected into the system architecture.

As is well known, the term ‘neuromorphic’ was coined by Carver Mead in the 1980s [14]. Mead was an analog VLSI circuit designer who argued for two big opportunities for realizing brain-inspired, now neuromorphic computing: (1) more localized algorithms and (2) use the physics of the devices we have for computation [26]. In a very real way, neuromorphic began as a strong argument for a form of analog computing. Transistors are in fact analog devices which, important to neural network performance, exhibit nonlinearities in their current vs voltage responses. The exponential behavior of transistors were specifically presented as an opportunity for replicating the exponential behavior observed in the wide variety of neurotransmitters that exist at the synaptic cleft in biology. With this, neuromorphic computing injected nonlinearity into the synaptic response. Important in why this occurred was the observation that transistors could be used as they are, taking advantage of the device physics, to realize more efficient implementations of brain-inspired computing.

Other key contributions from the work of Mead came in how inputs were summed and accumulated. The neuron component was realized as a CMOS integrator circuit, where weighted inputs that had been driven through synaptic circuits in turn charge up a capacitance. This capacitor based integrator, which often includes an amplification stage, could then be integrated with an analog comparator that produces a binary output based on the integrated potential being higher than some reference. The circuit is reminiscent of the McCulloch–Pitts neuron in that the activation function can often be modeled as a thresholding circuit.

The accumulation of weighted inputs is also simplified in a Mead style analog approach by leveraging Kirchoff’s current law to sum currents at a common circuit node. Here, voltage inputs can be driven into a set of synapses, where each voltage is an element in an input vector. The synapse works by leveraging the transistor device as it is, to the extent possible, and converting those voltages into currents. Further illustrating how this analog approach to using the devices as they are, Ohm’s law can actually be used to weigh the inputs, multiplying the voltage inputs by corresponding synaptic conductance values. With these weighted current inputs, a ‘zero-cost’ addition is performed using Kirchoff’s law [26].

### 3.3. Electronic devices for neuromorphic implementations

#### 3.3.1. Transistors: use them for what they are

When applied to biological channels, voltage on the membrane does not have a linear effect on the current, but rather causes the current to grow exponentially. The relationship cannot be achieved with a resistor. Since bipolar junction transistors and sub-threshold metal-oxide semiconductor field-effect transistors (MOSFETs) both exhibit an exponential relationship between voltage and current, they have shown great promise in representing biological synaptic channels. There are a number of reasons why MOSFET transistors have been used over the years. Their primary advantage is that, below the threshold of electric conductivity, they dissipate very little power. Furthermore, sub-threshold MOSFET transistors deliver current levels comparable to those found in biology. Neurology and sub-threshold MOSFET physics exhibit a great deal of similarity. Biological channels consist of two components: the original channel, the physical structure through which ions flow, and the gating mechanism which regulates whether the channel is open or closed. Sub-threshold MOSFETs have a similar concept, but there is one major difference. Whereas the biological molecular channel has a built-in activation/deactivation mechanism, the MOSFET’s activation/deactivation mechanism must be designed.
Moreover, stochasticity governs the current flowing through each biological channel and an MOSFET with a sufficiently small width can also achieve this. Due to all these analogies, transistors have commonly been used to model a population of biological channels [27].

3.3.2. Exploiting memristance for neuromorphic

The memristor was first proposed by Leon O Chua as one of four basic circuit components in [28], arguing this device acts as the missing link between electromagnetic flux and charge. Later, researchers at HP labs [29] demonstrated the existence of passive devices with properties reminiscent of Chua’s predictions. Memristors are typically implemented as two-terminal nanoscale devices that exhibit hysteretic switching resistance with memory and are non-volatile in nature. Memristors are self-regulating electronic devices in that the voltage applied across them modulates their resistance. Memristors can have a resistance that lies anywhere between two extreme values known as the low resistance state (LRS) and the high resistance state (HRS). The resistance level can be adjusted based on the magnitude of the voltage applied and the length of time that the voltage is applied. Switching from HRS to LRS and LRS to HRS could have different threshold voltages. For bipolar memristors, these are referred to as the positive threshold voltage ($V_{tp}$) and the negative threshold voltage ($V_{tn}$), respectively. Additionally, the switching times for HRS to LRS switching and LRS to HRS switching tend to be different, and these times are referred to as the positive switching time ($t_{swp}$) and the negative switching time ($t_{swn}$), respectively. Moreover, these values are affected by the type of device under consideration such as the material used and the switching mechanism.

The memristor, therefore, has the property of storing different values of resistance levels similar to artificial synapses in SNNs. Biological neurons include synapses that allow signals to pass between two neurons as well as weights the incoming signals [30]. Learning and storing information are based on synaptic plasticity, the changing of weighting factors in synapses. A memristor is useful as an optional component for neuromorphic circuits, since synaptic weight can be encoded as a value of memristance. Moreover, in terms of efficiency, the two terminal memristive synapse has the benefit of consuming less power, space, and cost [31].

3.4. Common neuromorphic components today

Many of the same circuit-level concepts pioneered by Mead and others are still in widespread use today by many in the neuromorphic field. The analog integrate-and-fire (I & F) neuron is a tried and true option for neuromorphic circuits. Leveraging Kirchhoff’s law to simplify addition is also an incredibly powerful method that helps reduce hardware overhead. However, computing with current does come with a cost in that even temporary static currents represent increases in power consumption. Thus, many in the field now turn to techniques such as sub-threshold operation to further reduce power. Since neuromorphic implementations are considered more bio-realistic, the reduced ‘biologically plausible’ speeds that come with sub-threshold operation are usually considered acceptable.

I & F neurons have been implemented in a variety of ways. In [32], the conductance-based silicon neuron was proposed, which is implemented as a current-mode conductance-based neuron with plasticity. In this case, the output current varies with the injected spikes like the I & F mechanism. This silicon neuron is therefore a good representation of the I & F. Neurons in [33] are another example of an analog I & F neuron containing low-power op amps that operate in two asynchronous phases, the integration phase followed by the firing phase. The op amp acts as a leaky integrator with a preferred leak rate and charges a capacitor based on the spikes from the input in this I & F neuron design. Indiveri introduced a low-power adaptive I & F neuron that was shown to reduce the power consumption relative to that of most axon hillock designs [34]. This design not only reduced the power of the spiking neuron, but also provided spike frequency adaptation and an adjustable refractory period.

Research has also focused on implementation of digital neurons to adopt the advantages such as simplicity, high signal-to-noise ratio, scalability, and affordability. As an example, Muthuramalingam implemented a single neuron using an FPGA including serial versus parallel implementations of computational blocks, bit precision and use of look-up tables in [35]. A piece-wise linear neuron is described in Hikawa [36], and it is implemented using an FPGA for its activation function. A digital stochastic bit stream neuron has been used to generate linear and sigmoid activation functions in studies by Daalen et al [37, 38]. Skrbek presents an architecture of shift-add neural arithmetic blocks implemented at a neuronal level, enabling the implementation of multiplication, square root, logarithm, and nonlinear arbitration functions [39]. In addition, a generic mixed-mode neuron was proposed in [40] whose accumulation rate can be tuned on-chip to avoid the need to adapt analog I & F neurons to the new types of devices that will interface with them.

The majority of the neurons available are CMOS silicon neurons. Nevertheless, other emerging materials are used because of their efficient energy consumption and optimized area for designing neurons. Memristors, for example, are used to define stochastic nature of neurons and describe complex spiking techniques [41, 42].
PCM devices [43, 44] are utilized in neuron designs as well. Finally, two-terminal volatile insulator–metal-transition (IMT) devices have been considered in combination with non-volatile memristors. The use of IMT devices is seen as an opportunity to realize very compact integrators with built-in spike shaping due to the Joule heating effects of the IMT device itself [45–47].

A number of synapse circuits have been developed with features inspired by biology. Some synapse models draw inspiration from ion pumps found in nature [48] while some researchers focus on modeling ion channels [49]. The spike time dependent plasticity (STDP) model has also been found to be successful for localized on-line learning in synapses, which is one of the popular learning algorithms for SNNs [50]. Non-spiking networks are typically trained using backpropagation [51] or least mean squares [52]. Synaptic hardware can be designed using a variety of devices, ranging from static CMOS to emerging technology. CMOS synapses reported in [53] were implemented in a 0.8 μm CMOS process and demonstrated both short and long term plasticity. CMOS was also used in the design of synapses in [54–56]. For example, a fully analog synapse using 0.6 μm CMOS process was presented in [55] that includes two functional transconductance amplifiers that reproduce synaptic weights, and also using on-chip STDP learning.

Several synapse designs have been proposed based on memristors to make them more area and energy-efficient. The neuromorphic synapses in [57, 58] use crossbar memristors to achieve high density synapses (figure 3(a)). A 1-transistor, 1-memristor (1T1R) crossbar array, also illustrated in figure 3(b), offers added benefits due to the use of a transistor in each RRAM memory cell that mitigates the sneak current path and programming disturbance associated with resistive (i.e., 1R) crossbar arrays [59]. The memristor bridge synapse has also been proposed for expressing negative and positive weights in other architectures, such as [60]. [40] demonstrated a bi-memristor synapse that can implement both positive and negative weights without additional local circuitry for the synapse, reducing the complexity associated with the memristor bridge synapse. Apart from memristors, there are a few other interesting emerging materials used to design synaptic components, including floating gate (FG) transistors, spin devices, and PCM cells. In general, FG transistors are used for storing synaptic weights [61] and the STDP mechanism [62]. PCM cells [63] and spintronic devices [64] are used for their high density and implementation of learning behaviors.

### 4. Reconfigurable neuromorphic system implementations

Many in the neuromorphic community have concentrated on custom VLSI circuits that either implement one class of applications, as a very application specific analog/mixed-signal implementation, or are used to explore neuroscience ideas and theory. This work is largely representative of where neuromorphic computing has existed since the late 1980s: exploratory in the sense that there remains much for us to learn about the brain and how we can emulate such behavior in the design of future computer processors [26]. Thus, the community has made great strides exploring various neurobiological behavior through carefully constructed analog/mixed-signal circuits that emulate that functionality. At the same time, this work has begun to evolve...
more toward system implementations of full-scale neural networks. It is this evolution toward full-scale network realization that has broadened the definition of neuromorphic computing into something that includes more digital implementations, including systems emulated using general purpose processors.

One apparent goal is the implementation of neuromorphic computer systems that are reconfigurable, even programmable in some manner. In a very real sense, this goal could be defined as a more general purpose approach to neuromorphic system design, where the system is built for neural network processing but can be reconfigured to execute a range of possible neuromorphic applications. The same hardware, in this case, could be used to execute image processing or a range of control applications.

4.1. Field programmable analog array for neuromorphic systems

Building upon the latter portion of section 3.4, field programmable analog array (FPAA) devices provide the components and functionality necessary for a neuromorphic system, in a reconfigurable package. The use of analog designs can reduce area usage by two orders of magnitude, and energy consumption by three orders of magnitude [26, 65]. Analogous to FPGAs, FAAs contain configurable analog blocks (CABs) that contain a variety of analog devices, such as transistors, capacitors, amplifiers, multipliers, and other components. These CAB contents vary between designs to fit the intended architecture. Initial FAAs used SRAM based memory, as in FPGAs, however, this necessitated the use of a DAC for each individual configurable parameter [65]. More recently, FG memory has become the preferred storage method for parameters due to its accuracy, wide programming range, ability to store analog values, and non-volatility. One of the most notable benefits of a recent FPAA which uses FG devices is that the routing itself can perform computation operations by utilizing partial reconfigurability of the FG devices and some additional circuitry [66]. The operation performed can be a vector matrix multiplication, as in [66], but others are possible [67]. However, FG devices require high voltages, up to 12 V on a 350 nm process for the design in [66], which are generated using on-chip charge pumps. The design in [66], and other recent FP AAs, utilize direct interaction between analog and digital components, instead of requiring a plethora of converters [65].

One prime example of how FAAs can be utilized for neuromorphic systems is in [68], where the Hodgkin–Huxley neuron is implemented to obtain data from a silicon-based design. The resulting design, which was tunable and fully reconfigurable thanks to the FPAA, consumed less than 1 μW of power [68]. With the use of FAAs, through reconfigurable analog circuits, a multitude of neuromorphic applications become possible through much more compact implementations, which consume significantly less power than their digital-only counterparts. This can enable the design of neuromorphic systems which are small enough for embedded systems and IoT, while also being battery powered.

4.2. Reconfigurable on-line learning spiking neuromorphic processor

The reconfigurable on-line learning spiking neuromorphic processor (ROLLS) was presented in [69], implemented as a full-custom neuromorphic learning circuit with 128k analog synapses and 256 neurons. More specifically, the ROLLS neuroprocessor architecture includes a row of 256 neuron cells and two 256 × 256 synapse arrays. One synapse array is built from circuits designed for long-term plasticity mechanisms while the other array implements short-term plasticity mechanisms in each synaptic cell. Additionally, the architecture includes 2 rows of 256 ‘virtual synapses,’ implemented as linear integrator filters that provide excitatory and inhibitory synaptic behavior. The overall processor architecture is programmable, in the sense that a user can configure specific synapse properties, network topology, and neuron properties. While individual synapses and neurons are analog integrated circuits, implemented to be bio-realistic, the configuration logic that manages how the processor executes a neural network for a particular application is digital.

The synaptic and neuron circuits are all implemented with various configurable features, allowing the user some flexibility in how the neuroprocessor can be tailored to specific applications. The neurons, for example, are adaptive exponential I & F neurons that include several bio-realistic functions [15]. The neurons circuits can model the dynamics of N-methyl-D-aspartate receptors, leak conductance, spike-frequency adaptation, emulated sodium channels for producing spiking outputs, and emulated potassium channels for a refractory mechanism. These various features are controllable by the application of digital signals, illustrating neuroprocessor configurability [69].

ROLLS operates at biologically plausible timescales (milliseconds to seconds) and is built from components whose behavior is faithful to neurobiology, to the extent possible. If we were to further categorize this particular architecture, we could say it is a good example of a bio-realistic, real-time neuroprocessor. In addition to being designed with real-time operations in mind, ROLLS is also a low-power implementation due to the fact that the core analog circuits operate in subthreshold mode. This subthreshold mode of operation not only saves power but also helps in achieving the goal of biologically plausible timescales for neuroprocessor operation.

As is the case for many neuromorphic systems, communication in and out of the ROLLS neuroprocessor is accomplished through address event representation (AER) of spikes. In this particular implementation, spiking
events exiting the array of neurons are encoded as the destination address, basically row and column information for the synapse/neuron that will receive the spiking event. While the internal machinations of ROLLS are primarily analog, leveraging transistor devices for what they are to better realize bio-realistic operation, global communication is digital.

4.3. Dynap-SEL

Another neuromorphic processor from Giacomo Indiveri and collaborators is known as the dynamic neuromorphic asynchronous processor (DYNAP) [70]. This is a mixed-signal, multi-core neuroprocessor which combines efficient analog computational circuits with highly robust, asynchronous digital logic for communications. An improved version of DYNAP with additional features was more recently developed called DYNAP with scalable and learning devices (Dynap-SEL) [71]. The original DYNAP neuroprocessor was implemented in a 180 nm CMOS process while the improved Dynap-SEL chip uses a 28 nm FDSOI process.

The Dynap-SEL chip contains four neural processing cores that each hold 16 × 16 analog neurons along with 64 4 bit programmable synapses per neuron [71]. A fifth core is also present on the chip that contains 1 × 64 analog neuron circuits, 64 × 128 plastic synapses with on-line learning, and 64 × 64 programmable synapses [71]. The synaptic inputs for each of these cores are routed by asynchronous AER circuits. The neurons integrate synaptic input and translate output spikes into address events which are then routed to other synapses.

Dynap-SEL uses a multi-level, mixed mesh/hierarchical routing structure that makes use of both source routing and destination routing. This mixed routing scheme takes advantage of the low bandwidth usage of mesh routing along with the low latency of hierarchical routing [71]. Events that are generated by the neurons can be routed by one of three levels of routers. The R1 routers handle routing within the same core, R2 routers handle routing events to other cores on the same chip, and R3 routers handle routing events to cores on other chips. The R1 routers utilize source routing while the R2 and R3 routers use forms of destination routing. This routing method is supported by distributed SRAM and ternary content addressable memory throughout the architecture.

The highly flexible and optimized routing scheme used by Dynap-SEL makes for a highly scalable system and for resources among different chips to be merged. The asynchronous memory control present also allows for on-line reconfiguration which facilitates structural plasticity and on-line learning algorithms. Dynap-SEL also effectively eliminates the von Neumann bottleneck problem by making use of multiple physical circuits in parallel to carry out computation as well as by co-localizing the memory and computation via distributed memory modules throughout the architecture [71].

4.4. Neuroprocessors of the human brain project

Based in Europe, the human brain project (HBP) aims to ‘tame brain complexity’ by ‘building a research infrastructure to help advance neuroscience, medicine, computing and brain-inspired technologies’ [72]. In particular, the HBP aims to emulate and study the activity of the brain with great attention to detail. As part of the HBP, researchers at the University of Manchester have developed a large digital neuromorphic system known as SpiNNaker [73]. Similarly for the HBP, a separate research collaboration by the University of Heidelberg and the Technische Universität Dresden built a mixed-signal neuromorphic system at wafer-scale known as BrainScaleS [74, 75].

4.4.1. SpiNNaker

The SpiNNaker neuromorphic system makes use of more than one million parallel ARM processors to model and simulate one billion spiking neurons with biologically-realistic synaptic connections in real time [73]. SpiNNaker is designed as a two-dimensional toroidal mesh of chip multiprocessors (CMPs). Each of these CMPs contain 1 Gbit of mobile DDR SDRAM and an MPSoC that incorporates up to 20 ARM 968 processors interconnected by two self-timed network-on-chip (NoC) fabrics. The first self-timed NoC is a communication NoC which carries neural spike event packets between processors. The second self-timed NoC is used as a general-purpose interconnect that allows processors to access system resources.

SpiNNaker was designed with three main design principles: bounded asynchrony, virtualized topology, and energy frugality. Bounded asynchrony is inspired by biological systems which have no global synchronization but each rate of local change is still controlled by the same local physical effects. In SpiNNaker, bounded asynchrony is implemented via the use of real-time event-driven application code. To compute the neuron models in real-time, a 1 ms timer interrupt runs on each of the parallel processors which evaluates the neuronal differential equations and can generate output spike events. These timer interrupts running at the same rate throughout the system and a negligible communication delay create a system-wide synchrony despite each processor running asynchronously to each other. Adhering to a virtualized topology means that SpiNNaker is
not required to map neurons to processors in a way that matches their physical topology. This is because communication is effectively instantaneous on a biological timescale throughout the system allowing any neuron to be placed on any processor. Finally, the principle of energy frugality maintains that energy consumption is the real cost of computation. To help lower the energy consumption system-wide, SpiNNaker uses more efficient embedded processors in the CMPs.

4.4.2. BrainScaleS

BrainScaleS is a wafer-scale neuromorphic system designed for implementing upwards of 40 million synapses and 180k neurons [74, 75]. Its communication infrastructure uses a digital network chip IC along with FPGAs to employ high-speed, source-synchronous serial packet communication for spike event transmission. The communication infrastructure facilitates BrainScaleS to run at a speed-up factor of \(10^4\) as compared to biologically realistic speeds.

The BrainScaleS mixed-signal system is built out of analog network cores (ANCs) which unite an exponential I & F neuron model with its synapses into a common structure. Each ANC is then combined with an interconnect using the serial event protocol structure mentioned and support circuitry to create a chip known as a high input count analog neural network (HICANN). The HICANN ASIC forms the basic building block of BrainScaleS and its component structure is shown in figure 4. In this block diagram, the ANC is the largest structure and is split into an upper and lower half, each containing 256 \(\times\) 256 synapses and 256 membrane circuits. These ANC halves are split by 64 horizontal bus lanes and flanked by 128 vertical L1 bus lanes. At intersections between these bus lanes are crossbar switches. Connections to adjacent HICANN chips are also present on each half of the ANC. Digital and analog circuits for supporting STDP based on-line learning are also present in each half of the ANC. A passive sparse switch matrix is placed at the intersection of the synaptic driver inputs and the vertical L1 bus lanes.

The current BrainScaleS wafer system consists of 352 HICANN ASICs fabricated onto a 20 cm wafer using a 180 nm CMOS process. A newer, second generation BrainScaleS system was revealed to be in development that will use a smaller, 65 nm CMOS process [76]. This new version aims to improve portions of the neuron circuit design to enable nonlinear dendrites as well as perceptron mode operation. A high-speed analog-to-digital converter will also be included for membrane voltage readout.

4.5. TrueNorth

The TrueNorth architecture was introduced in [77] implemented as a non-von Neumann, low-power, highly parallel, scalable, and defect-tolerant invention. Neurosynaptics core was used as the building block for the architecture. More specifically, TrueNorth includes 4096 neuro-synaptic cores for a total of 1 million digital neurons and 256 million synapses. The chip’s event-driven routing infrastructure enables neural networks connected by many millions of synapses to communicate with each other. A key strength of this architecture is the novel hybrid asynchronous–synchronous model that interfaces both asynchronous and synchronous
elements, as well as tools to support the design and verification. Moreover, it represents an event-driven, low-power, real-time neuromorphic processor that consumes about 65 mW of power. A variety of cognitive and sensory perception applications can also be customized using the TrueNorth chip’s connectivity and neural parameters.

Spiking neurons are implemented in the TrueNorth architecture by using a network that connects them. The chip is programmed by defining the behavior of neurons and the connectivity between them. Data transmitted using spikes can be encoded using their frequency, time, and spatial distribution. An array of neuronal synaptic cores and peripheral logic are included in a 64 × 64 chip. A scheduler, token controller, core SRAM, neuron, and router constitute a single core. A synaptic crossbar connects each core’s synapses. Neurons transmit their outputs to the input buffer of the axon that they communicate with. Communication can happen across the routing network or over the same core of the communicating neuron. Two-dimensional mesh networks are created by communicating in the west, east, north, and south directions among the router and the four neighbors. The spike packet contains the dx, dy address of the destination core, axon index and tick for integration, as well as several flags to assist debugging. As soon as a spike arrives at the router of the destination core, it is passed on to the scheduler.

4.6. Loihi

Loihi was presented in [78], implemented as a neuromorphic manycore processor with on-chip learning to advance the development of SNNs in silicon. It incorporates a variety of novel features, such as hierarchical connectivity, dendritic compartments, synaptic delays, and, most importantly, programmable synaptic learning rules. More specifically, there are 128 neuromorphic cores in Loihi, three × 86 processor cores embedded on the chip, and four communication interfaces that extend the mesh in four directions to other chips. The networks-on-chip (NoCs) perform all communication asynchronously between cores by sending packetized messages. In addition to × 86-to- × 86 messaging, the NoC supports write, read and response messages to work with core management. It also supports spike message for SNN computation, as well as barrier messages to time synchronize between cores. The host CPU can source all types of messages externally or the × 86 cores can source them internally, and any on-chip core can receive them. A second-level network may allow messages to be hierarchically encapsulated to facilitate off-chip communication. With hierarchical addressing, the mesh protocol can support up to 16 384 chips and 4096 on-chip cores.

There are 1024 primitive spiking neural units (compartments) in each neuromorphic core, which are grouped into sets of neuronal trees. A total of ten architectural memories store configuration and state variables for each compartment, as well as their fan-in and fan-out connectivity. A pipelined, time-multiplexed approach is used to update their state variables every algorithmic time-step. A neuron generates a spike message when its activation level reaches a certain level. The spike message is sent to a set of fan-out compartments, which reside in one or more destination cores.

Loihi offers several features that ease the constraints that can sometimes be burdensome for programmers imposed by other neuromorphic designs. As an example, Loihi provides three sparse matrix compression models that compute neuron indices by storing state variables in synapse state variables as well as the common dense matrix connectivity models. A neuron also has the flexibility to send a single spike to any number of destination cores, depending on the network connections. Moreover, the precision of Loihi weights ranges from one to nine bits, signed or unsigned, and weight precisions can be mixed even within a single neuron’s fanout distribution among all the neurons in the network. The definition of connectivity templates can be used as a generalized weight sharing mechanism that can be used to support different types of CNNs. The feature can greatly reduce the amount of connectivity resources required by networks.

A Loihi core features a programmable learning engine that can continuously update synaptic state variables to reflect past spike activity in real time. The learning engine employs filtered spike traces to support the widest possible range of rules. A rich set of input terms and synaptic target variables is proposed for the learning rules that can be programmed in microcode. The learning profiles are bound to these specific sets of rules that are relevant to each synapse to be modified. An individual profile is defined by some combination of presynaptic neurons, postsynaptic neurons, or synapse types. Simple pairwise STDP rules, as well as complex rules based on both spike-timing and rate averages, are supported, as are triplet STDP rules and reinforcement learning. This chip includes only digital logic, is functionally deterministic, and is designed with asynchronous bundled data in mind. In that way, it enables event-driven spike generation, routing, and consumption, which reduces idle time. SNNs that are fundamentally highly sparse in both space and time are ideal candidates for implementing this implementation style.

4.7. A family of dynamic adaptive neural network architectures

Starting in 2014, the TENNLab neuromorphic research group at the University of Tennessee began developing neural processor architectures for hardware implementations. The first of these processor architectures was
known as the dynamic adaptive neural network array (DANNA) [79–81]. DANNA began as a design targeted and optimized for FPGA deployment, but was later adapted for a digital VLSI implementation with a 130 nm CMOS process in 2015. An extension of this architecture, known as memristive dynamic adaptive neural network array (mrDANNA), was later developed that utilized memristor devices in the synapses to improve the efficiency of the neuromorphic system [82]. The use of memristive synapses necessitated a shift to a mixed-mode architecture to drive these inherently analog devices. In 2018, the DANNA architecture was improved upon in the second generation DANNA2 architecture [83]. This new digital architecture improved the network density, achievable clock speeds, and training convergence rate over the original DANNA architecture. DANNA2 was implemented and tested on FPGAs, but was also designed with deployment using VLSI processes in mind. Currently, work is ongoing for the TENNLab research group in the development of a more convergent and flexible architecture as part of a new neuromorphic system known as the reconfigurable and very efficient neuromorphic system or RAVENS [84].

4.7.1. DANNA

The early DANNA architecture consists of an array of dynamically programmable neuromorphic elements that are connected together to create a SNN [79]. Each element in a DANNA array can be programmed to operate as a neuron, a synapse, or as a pass-through synapse to extend the range of connectivity. Elements programmed as neurons have a programmable threshold and input enables for each of its connections to other elements. When programmed as synapses, elements have programmable weights, refractory periods, delays, and the same input enables. As a synapse, only one input port and one output port is used, but an element can be programmed as a special pass-through type synapse that has one input enabled and multiple outputs enabled to increase its fanout capability and overall range. Each element in a DANNA array is only capable of connecting to its nearest 16 neighboring elements. When tiled together into an array, the left-side edge elements act as inputs into the array from external stimulus and the right-side edge elements act as outputs from the array. An example DANNA network array is shown in figure 5.

Internally, a DANNA element is comprised of four main components: input sampling, accumulate and fire, long-term potentiation/depression (LTP/LTD), and synapse delay. The input sampling component in neurons sequentially samples the output of all connected elements to check for input before the accumulate and fire component handles charge updates and threshold comparisons for fire events. For synapses, the input sampling component only samples from one input and passes the signal through to either one output or multiple outputs if it is configured as a pass-through synapse. The accumulate and fire component for synapses stores the synaptic weight of a connection and updates it during potentiation/depression. The LTP/LTD component is only used when elements are configured as synapses and provides logic that determines when potentiation or depression conditions are met. Finally, the synapse delay component enables programmable synaptic delay and is implemented as an addressable shift register.
Figure 6. Block diagram of mrDANNA architecture, showing a zoomed in view of each mrDANNA core as well as the programmable switch matrix for programmable inter-core connectivity.

TENNLab’s software development framework [85] was used to train and simulate network performance for DANNA and ultimately led to an FPGA deployment on a robotic platform [86]. This robotic platform, known as NeoN, ran a DANNA network trained to perform real-time, autonomous navigation around a space with the use of a sweeping LIDAR. The LIDAR data was converted into spikes which were used as inputs into the DANNA array while the outputs controlled the left and right motor speeds. This platform was rather successful at exploring a majority of the space it was placed in without colliding with obstacles along the way.

4.7.2. mrDANNA

mrDANNA [87] can be thought of as the CMOS-memristor hybrid equivalent of the DANNA architecture presented in section 4.7.1. However, there is a key difference between them, as the elements of the DANNA architecture is replaced by self-contained mrDANNA cores. Each mrDANNA core houses 6 synapses and a neuron whereas each element in DANNA is either a neuron or synapse/pass-through synapse. Programmable switch blocks between the cores allow them to make 6 nearest neighbor connections through 6 synapses. The re-configurable interconnect between the cores as well as their internal structure is shown in figure 6.

The building blocks of the mrDANNA core are twin-memristive synapses and a mixed-signal neuron, which takes analog input currents and produces digital output pulses or spikes. The analog input currents are generated by the twin memristor synapse, the conductance of which mimics the synaptic weight. The two memristors in each synapse is connected such that opposite polarity voltages are driven at one end, while the other end is shorted together. Depending on the memristance value of the two memristors, a net positive or negative current is produced at the shorted end, which acts as input to the neuron. Hence, the synaptic weight is given by the difference in conductance of the twin memristors [82]. Another property of the synapse is the programmable delay, which is realized by D flip-flops inside the synaptic buffer to delay an input signal by the programmed number of clock cycles. The mixed-signal neuron is implemented as an I & F neuron, that accumulates charge proportional to the input current and upon crossing a preset threshold, produces a spike that is sampled at the clock edged and passed on to the next mrDANNA core through the switch block.

The mrDANNA architecture combines both off-line and on-line learning through evolutionary optimization (EO) and STDP, respectively. At first, the EO algorithm [88, 89] generates a recurrent SNN comprising neurons and their connections (synapses), along with synaptic properties such as delay and weight of all synapses used. Then, the SNN is mapped onto a mrDANNA system, with each neuron being replaced by a mrDANNA core and the connectivity configured via switch blocks. During runtime, the synaptic weights are
changed incrementally to achieve expected results by means of pulse width modulated STDP [90] or a binarized version of STDP, namely digital long term plasticity (DLTP) [82]. Three classification tasks, namely Iris, Wisconsin Breast Cancer and Pima Indian Diabetes dataset taken from UCI machine learning repository [91] were considered for evaluating the performance of the mrDANNA architecture. The attained accuracy for the classification tasks was reported to be 90.70%, 96.56%, and 73.95%, respectively, with DLTP [82] and 96%, 84.24%, and 73.44%, respectively, using STDP [90].

4.7.3. DANNA2

After many lessons learned at both the hardware architectural level and at the software training and simulation level, the second generation DANNA2 architecture was developed [83]. This improved neuroprocessor design had several fundamental changes over its predecessor including a merged synapse/neuron element, enhanced connectivity, and added leak functionality. Although it had these changes, the elements were still restricted to a nearest neighbor connectivity scheme, a two-dimensional grid layout, and edge input/output elements.

The core neuromorphic element in DANNA2 is comprised of one post-synaptic, LIF neuron and 24 tightly coupled synapses [92]. To operate, the element relies on the concept of network cycles and element cycles. Each network cycle is broken into 10 element cycles in which the work performed by the element is broken up. This differs from DANNA in that DANNA had an accumulation clock that was 32 times faster than the network cycle clock. A depiction of the DANNA2 element’s structure is shown in figure 7. Beginning at the top of this element depiction are a group of 24 distance registers that gather the spike inputs from that element's 24 nearest neighboring elements. During each element cycle, three synapses are sampled by reading the corresponding distance registers as well as the synaptic weight programmed into the synapse table. The synapse units forward the correct synapse weight for accumulation to the accumulator and also handle the calculation of weight updates caused by on-line learning due to spike-time-dependent plasticity (STDP). STDP is touched on more in section 6. The accumulator then sums the weight of each of the synapses, minus a linear leak value, along with forwarded weight from up to four fan-in ports. Fan-in ports are used when an element is configured as a new fan-in type element which allows for an extended connectivity range and effectively more synapse connections at the cost of an added network cycle of delay. When an element is configured as a fan-in element its total accumulated value is forwarded to another neuron then reset for each network cycle rather than operating as a normal neuron. Finally, the total accumulation is compared to the programmed threshold value during the final element cycles for output fire generation.

DANNA2 also introduced an interesting concept at the array level that optimized resource allocation on FPGAs. This concept was sparse arrays that only implement the elements and connectivity required for a particular network topology. This meant that the connectivity was no longer limited to the nearest 24 nearest neighbors at the cost of losing external programmability at runtime. Sparse networks find their primary viability on FPGAs or at least very focused use cases. For this reason, sparse DANNA2 networks are not considered a truly ‘general purpose’ neuroprocessor and are not given focus for this review.

When compared to the original DANNA architecture, DANNA2 outperformed it in terms of network and element density, network speed, and achievable connectivity. The merged element design was largely responsible for the increase in element density along with optimized element logic which equated to around a 60% increase in usable neurons on a Xilinx UltraScale FPGA [92]. Along with the improved connectivity due to an increase in the number of neighboring connections and fan in elements, the neuron density improvement allowed DANNA2 to show a significant uplift in network capacity. Changing the element clock to be faster than the network clock by only a factor of 10 also led to an effective network speed of 5–10 times faster than DANNA [83]. Similar to DANNA, DANNA2 was proven to perform well in real time with a robotic roaming platform known as GRANT [93]. This robot improved upon the NeoN robot [86] by adding in the ability to target and even follow an object with a color recognition sensor. Network training time and overall network fitness for the same robotic navigation task was also improved as compared to DANNA2, with some of these training results are shown in table 1.

4.7.4. RA VENS neural core

While the DANNA2 architecture showed much improvement for both the hardware and the software model over DANNA, the fact remained that it was fundamentally a digital architecture. The neural core, or element, still relied on several element subcycles of the network clock to operate as expected. This meant that for a version of this architecture that catered to a mixed-signal context for testing memristive or simply analog neural circuits, many system changes would be necessary to adjust for analog timing. In an attempt to have a more convergent, multi-context architecture as well as move away from the restrictive ‘nearest-neighbor’ connectivity, development started on a modified neural core architecture as part of the RA VENS project [84].
This new neural core (nCore) design takes a lot of inspiration from the general structure of DANNA2, but makes some key changes. Firstly, the nCore architecture reduces accumulation into a single cycle operation. This likely means that the combinational delay of the charge summation limits the nCore clock frequency, but achieves a convergence in timing for accumulation cycles between a digital and analog accumulator. Most importantly, the concept of ‘virtual neurons’ is introduced. As compared to DANNA2, each nCore represents some number of synapses tightly coupled with a number of virtual neurons. Virtual neurons differ from true physical neurons in that they share some of the hardware with other virtual neurons yet still function as some number of independent functionally complete neurons. To facilitate the sharing of hardware for the virtual
neurons, time-division multiplexing (TDM) is used to create a two stage pipeline. Each virtual neuron is assigned a time slice and a complete network cycle is segmented into enough of these time slices such that each virtual neuron in the nCore has one. The difference between these time slices and the old element cycle is that on each time slice, a neuron is computing whether or not it should produce a fire event.

A block diagram showing the dataflow through a generic, multi-neuron nCore structure is shown in figure 8. In this diagram, many similar components as found in DANNA2 are present such as the synapse tables, delay units, accumulator, and compare and fire. The key differences are in the TDM controls and the organization of the structure. The nCore splits the whole structure into two main components, the synapse unit and the neuron unit. The synapse unit comprises all of the structures that handle synaptic weight storage, weight updates via on-line learning, and delay. The neuron unit contains all of the accumulation and spike generation logic. On the first time slice of a network cycle, known as the accumulation stage for that virtual neuron, spikes are pushed into a delay unit reserved for each virtual neuron. The spikes are then output to the learning units on the appropriate network cycle determined by the delay of its synapse. Each learning unit
grabs the weight of its synapse from the synapse table active on that time slice and forwards it to the accumulator where the summation of weights is performed combinationally. The summed weight is forwarded to the compare and fire module which can generate a spike for that virtual neuron if the threshold conditions are met. On the following time slice the same process will begin for the next virtual neuron, but the prior virtual neuron will undergo its update stage. In the update stage, the synapse table gets updated with weight updates caused by STDP, accumulated charge is updated due to linear leak or a fire event, refractory period counters update, and spikes are shifted through the delay units. This two stage process repeats until all the virtual neurons have completed their accumulation phase before the network cycle increments and output fires from an nCore become valid.

This new nCore design is only a building block in the early system design of RAVENS, but it has shown some promise in not only creating a convergent, multi-context basis for system design, but also in improving area and power requirements per functional neuron when compared to DANNA2 [84]. The TDM control scheme also opens up the opportunity to move away from a restricted, ‘nearest neighbor’ connectivity scheme, used by both DANNA and DANNA2, to a fully connected structure. This could be accomplished on a local level through several methods, such as a fully connected crossbar, as demonstrated by TrueNorth [77], or a pure, single layer mesh as in Loihi [78]. Although, through the use of TDM, more traditional connectivity styles, such as an FPGA-esque switch block architecture, become plausible. The rationale for, and implementation of, an interconnect structure utilizing such a style with the TDM nCores is further elaborated upon in section 5.2.

5. Physical interconnect for brain-like connections

5.1. Address event representations
AER is an asynchronous digital multiplexing technique that was first developed as an inter-chip communication protocol in [94]. AER has many similarities with the action-potential representation used by real neurons. In this system, the interval between the events is analog whereas the amplitude of the event is a standard digital amplitude. Therefore, the pattern of the events are similar to the action potential of neurons. The time interval between the events is used to encode the information. In order to encode the output, a temporal sequence of digital amplitude events is produced by the neurons in the sender, similar to a train of action potentials. Whenever a neuron notifies an event, the multiplexing circuitry announces that neuron’s address on the inter-chip data bus. The announcement is based on the assumption that the interval between the events is much longer than the time needed to transmit the address. Due to this, a large number of addresses can be multiplexed on one bus. The transmission of the address is annotated as an event by the receiver. Therefore, the communication code is named as an address-event representation, AER.

5.2. Physical interconnect for communication of spikes
While implementations exist for a unified interconnect methodology, such as [78], often, different approaches for local and global connectivity are utilized [95–97]. Both forms of connectivity can be implemented synchronously, asynchronously, or, most commonly, a combination of the two. Connectivity methods are divided into two groups: designs which provide full connectivity or those that only provide restricted, or partial, connectivity [77, 95, 96, 98]. Fully connected architectures provide the most opportunity to achieve the maximal configurability for an implementation. However, full connectivity is problematic when scaling, so partitioning the routing into levels helps alleviate this by limiting the required amount of dense interconnect area, which can also consume a significant amount of power.

Frequently, particularly at the global scale, packets are routed instead of wires, or even pure spikes. Generally, AER, as discussed in section 5.1, is utilized in global connectivity. Network on chip (NoC) mesh architectures are often used, and they typically fall into the following categories: tree, grid, or a hybrid. There are a multitude of options for grid designs, in which ‘router’ elements connect to other, adjacent, ‘routers.’ Several examples include: Loihi, which implements an orthogonal, two dimensional, structure [78]; SpiNNaker, which uses the SpiNNlink interconnect architecture, maintains a two dimensional toroidal mesh [95, 97]; and TrueNorth, which also implements an orthogonal, two dimensional, structure [77, 95, 96]. However, while Loihi’s interconnect spans all cores, TrueNorth allows for a spike packet to be sent through the mesh in a hierarchical manner [77].

Alternatively, BrainScaleS and neurogrid [99], a neuromorphic system designed and implemented by Stanford University, use H-trees and binary trees, respectively, which are hierarchical trees. These trees provide lower latency at the global scale due to the fewer number of total hops required for a compared to the single layer implementations of SpiNNaker and Loihi [74, 95].

For local connectivity, it is possible to implement schemes using configurable, or fixed, interconnect structures [77, 96]. Often, crossbar arrays are utilized for some portion of the interconnect design at this level. In
TrueNorth, as detailed in section 4.5, neurosynaptic cores are spanned by a ‘synaptic crossbar,’ which implements the local interconnect, in addition to the synaptic functionality. However, because of the large storage requirement for the configuration data, both on-chip and off-chip memory is used [77, 95, 96]. Additionally, at the global scale, the bandwidth between multiple chips is low, due to the extreme density [96].

Given the constraints of area and power, and the large range of available network styles, the partitioning of designs becomes interesting. Combining the architecture described in section 4.7.4 that utilizes virtualized neurons and TDM to share physical cores, the creation of fully connected ‘pseudo arrays’ becomes possible. By limiting the number of the physical cores in a cluster, an interconnect of reconfigurable routing structures, potentially in the same vein as FPGA switch blocks, allows emulation of an entire ‘pseudo array’ of cores with a significant reduction in area. Additionally, routing spikes between physical cores becomes very appealing due to the reduction of complexity. Due to the larger size of the emulated array, as well as the principle of locality, global connectivity use would be reduced. This methodology can emulate a fully connected array of neuromorphic cores, which utilize spike-based routing, while allowing greater scalability of a fully connected partition and reducing the area impact of the interconnect.

6. Neuromorphic hardware for learning

The key feature that sets neuromorphic computing apart from the traditional computing paradigm is the ability to adapt its internal structure, commonly referred to as learning. Neuromorphic hardware takes motivation from the mammalian (e.g. human) brain, wherein a complex matrix of neurons, synapses, and their chemistry allows it to process a multitude of information in parallel for performing cognitive tasks such as speed, image, and text recognition with low power and latency. Electro-chemical signals are propagated between neurons through synapses and the signal strength is adjusted to produce more accurate outputs by virtue of learning [100]. Taking inspiration from the brain, neuromorphic hardware mimics the properties of neurons and synapses, as well as the learning mechanism. Learning mechanisms can broadly be classified as supervised and unsupervised learning. The central action of these learning mechanisms is the updating of synaptic weights, which can be implemented by leveraging devices such as resistors, capacitors, FG transistors, and more recently, memristors.

Supervised learning is the most common form of learning in neural networks. The learning commences by providing a labeled dataset of inputs and outputs, called a training dataset. The network infers the relationship between the input and output of the training dataset. Then this relationship is used to map new inputs to outputs. Supervised learning algorithms can be applied effectively to various classification problems such as image classification [101]. At first, a large dataset containing different types of labeled images are gathered and provided to the network. The error between the expected result and obtained result is calculated. Based on the error, the network adjusts its internal parameters to minimize the error. The trained network is then subjected to the testing dataset to evaluate the performance.

The simplest form of supervised learning in neuromorphic hardware can be realized by the idea of a perceptron. The perceptron is a concept developed by Rosenblatt [20] that can be thought of as a building block for neuromorphic circuits. The single neuron named perceptron, takes a weighted combination of inputs and sums them to compare against a predefined threshold to provide an output of logic high or low. It is to be noted that the perceptron acts as a linear classifier and hence the data vectors must be linearly separable. Therefore, only linearly separable Boolean functions such as AND, OR, NAND, NOR can be implemented using a perceptron. For functions which are not linearly separable (i.e. XOR), two layers of perceptron are needed. The learning mechanism of the perceptron involves backpropagation and gradient descent algorithms. For each input vector, the error is calculated as the difference between the expected and the obtained output and is propagated backwards to change the associated weights. The process is repeated until the optimal set of weights which gives the minimum error is attained. Using the learning algorithms, MLP have been demonstrated to perform various pattern recognition tasks [102–110] using analog, digital or CMOS-memristor hybrid approaches.

Vast majority of the supervised MLPs found in the literature are not spiking in nature, as they employ a wide variety of activation functions (e.g., ReLU, tanh) between the neuron layers. However, supervised learning approaches have been developed for spiking neuromorphic hardware as well in recent times. Most of the mechanisms involve some form of back-propagation [51, 111–116], while a few of the methods try to convert ANNs to SNNs by time coding the inputs [117–125]. CNN to SNN conversion was demonstrated in [118] for CIFAR-10 dataset, proving to be two orders of magnitude more energy-efficient. A recurrent ANN to SNN conversion was presented in [119], where the network was implemented on TrueNorth to perform a natural language processing task with 74% accuracy and an estimated power of 17 μW. In [125], a 1T1R synapse array of drift memristors coupled with diffusive memristive neuron was used to convert an ANN directly to SNN. In [51], back-propagation based training was performed by taking spikes and discrete synapses as continuous probabilities and the trained network was mapped to a TrueNorth chip for MNIST dataset classification.
achieving 92.7% accuracy at 0.268 μJ energy per image. Another back-propagation based learning algorithm was presented in [114], with dedicated neuromorphic hardware in 40 nm CMOS process with 98% accuracy and 48.4–773 nJ energy per image for MNIST classification. In addition to ANN inspired back-propagation or ANN to SNN conversion, there exists a few supervised learning algorithms which are directly developed for SNNs [40, 126–129]. In [126, 130], the remote supervised method was presented which can be thought of as a supervised version of STDP, which is assumed to be applicable for unsupervised learning. The algorithm was implemented using various approaches and devices such as FPGA [131], CMOS mixed-signal [132], complementary graphene-ferroelectric transistors [133], memristors [134] etc. Another approach with LIF neuron and memristive supervised STDP with winner-take-all was presented in [128, 129]. A similar approach but with configurable digital neurons for compatibility with different memristor properties was proposed in [40].

Unsupervised learning infers learning without a labeled training dataset. This mechanism does not assume any feedback and tries to classify inputs based on the underlying statistics of the data. A good example of unsupervised learning is clustering of images based on their features. A learning algorithm for unsupervised learning is somewhat different than supervised learning. As the input dataset does not have any labels associated with it, error cannot be calculated in the learning phase. As a result, backpropagation or gradient descent algorithm does not apply to unsupervised networks. Unsupervised learning is related more closely to the learning mechanism in the biological brain, which follows Hebbian learning [23]. The Hebbian learning rule theorizes that the neurons that fire together should be wired together, meaning if there is a causal relation between two neurons firing, the connection between them should be strengthened. However, the learning rule is incomplete in the sense that it does not specify the amount of change in the synaptic weight. Also, the anti-causal scenario is not covered, wherein a post-synaptic neuron fires before the pre-synaptic neuron. An extension of Hebb’s postulate is spike timing dependent plasticity (STDP) learning, wherein both the causality and anti-causality of neuronal firing events, as well as the temporal difference are considered [135–137]. According to STDP, when a pre-synaptic neuron fires before a post-synaptic neuron, the synaptic weight between them increases (potentiation), but when a pre-synaptic neuron fires after a post-synaptic neuron, the synaptic weight between them is decreased (depression). The amount of potentiation and depression follows an anti-symmetric exponential curve. When the firing events occur closer together in time, the synaptic weight change is higher. As the fires occur further away in time, the amount of weight change decreases exponentially, dropping to zero beyond a certain window, which is defined as the STDP time window.

Various neuromorphic systems have been built around STDP and its variations to implement unsupervised learning platforms [138–143]. Transition metal oxide memristors have been widely used as the device of choice for unsupervised SNNs with STDP due to its inherent plasticity, although a few PCM devices have also been used in the literature [144, 145]. In [141], an unsupervised STDP rules constructed with a homeostasis neuron architecture was presented to demonstrate the effect of competitive learning, achieving 95% accuracy for MNIST dataset. A similar architecture was shown in [140] with the SNN implemented with memristive synapses, reaching 93% accuracy and resilience against memristor non-idealities and process variations. A more practical approach was considered in [142] with practical memristive devices, showing good immunity against non-idealities. However, the neurons, homeostasis mechanism and WTA logic was implemented in software simulations without any correspondence to practical devices. Another unsupervised STDP framework was presented in [143] concentrating on the robustness of the design with practical memristive synapses and homeostasis neurons in hardware, showing 90% accuracy in a digit classification application, consuming only 3.16 pJ energy per spike per synapse. Although the unsupervised neuromorphic systems are thought to be more biologically plausible and energy efficient, the accuracy is well short of those achieved with supervised learning platforms. Hence, the combination of both approaches by merging spatio temporal learning with backpropagation [115] to achieve high accuracy and energy efficiency remains an active field of research.

7. Summary

Biological neural systems are analog systems that make use of physical properties to perform classification, prediction, decision making, sensory-motor control, and other computation processes. Neuromorphic hardware includes a number of computations that can naturally be performed using analog circuits. Analog systems generally operate asynchronously, unlike digital systems, which generally operate synchronously. However, neuromorphic systems often follow this rule of thumb, since analog systems often drive events while digital systems may use synchronization clocks. In addition to programmable architectures such as FPGAs and FPAAAs, there also exist custom chips that may be digital, analog, or a combination of both. Table 2 summarizes and compares the performance of several state-of-the-art neuromorphic architectures reported in the literature. Intel’s Loihi and IBM’s TrueNorth are two of the more popular contemporary digital neuromorphic processors, both implemented as full custom integrated circuits. There are some operations on the TrueNorth chip that occur asynchronously, but the clock governs basic time steps in the system. TrueNorth’s architecture is a
hybrid, whereas Loihi is synchronous. TrueNorth uses a fixed network model with spikes for neurons, while SpiNNaker gives users more options for neurons, synapses, and learning algorithms. Energy efficiency of SpiNNaker, however, is sacrificed as a result of this flexibility. For the majority of implementations, STDP on-line learning is employed because it is a pretty good match for what occurs during biological synaptic transmission. However, no on-chip learning mechanism exists for TrueNorth. Since neuromorphic originally referred to analog designs, analog implementations, such as FPAs and ROLLS, are biologically plausible. In contrast, mrDANNA combines both analog and digital functions to achieve improved energy-efficiency. Furthermore, the nanoscale memristive devices used in mrDANNA promise significant reductions in system area and power consumption. However, since memristors remain primarily experimental devices, the promised advantages of neuromorphic components built using memristors have yet to realized at large scales.

Neuromorphic circuit design, including neuroprocessor architectural advances, are expected to continue for some time to come. In fact, it is likely safe to say the field is still in its infancy. The continued improvement of nanoelectronic devices, such as transition metal-oxide memristors and ferroelectric switches, present an opportunity for ongoing research where significant gains can be achieved. However, advances in device technology alone will not solve all problems. For the neuroprocessor to truly reach a status like that of the well known von Neumann architecture, several architectural advances are also required. For example, while many of the architectures reviewed in this paper provide some form of on-line learning, usually via some form of spike-time dependent plasticity, neural networks still must be trained or initialized off-line before being uploaded to the neuroprocessor. The computer systems typically used to perform this off-line training are typically built using more conventional CPU and GPU processing elements. Thus, there exists a real opportunity to consider neuroprocessor architectural advances that can be leveraged in such a way that the neuroprocessor is used to train the initialized networks that will eventually run on the same or similar systems. Recent advances in more bio-realistic training algorithms, including backpropagation through time for spiking neuromorphic systems, may provide some opportunities for hardware that can improve training.

A final thought from a more architectural perspective relates to the need for continued work in developing software systems for neuromorphic computing. For neuroprocessors, such as several discussed here, there could even be opportunities for a sort of instruction set architecture (ISA) that can be used to guide several operations on the neuroprocessor. Specific commands or instructions are not uncommon for monitoring system behavior. A neuroprocessor ISA could be potentially be expanded to include operations related training and/or inference. However, neuromorphic computing should not be expected to evolve exactly like general purpose von Neumann architectures. In fact, computer architects have an added challenge in that they must think creatively, beyond what is typically done for conventional ISAs, to consider new approaches for neuromorphic. For example, there is a subtle idea of the application-specific neural network as something that should be uploaded or configured into neuroprocessor hardware. Such a view is not terribly different from reconfigurable computing using FPGAs. Regardless of how things emerge in the coming years, neuromorphic computing has plenty of room to grow, including the development of new neuroprocessor hardware and software.

**Acknowledgments**

This material is based in part on research sponsored by Air Force Research Laboratory under Agreement No. FA8750-19-1-0025. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein
are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the US Government.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

ORCID iDs

Garrett S Rose https://orcid.org/0000-0003-3070-4087

Mst Shamim Ara Shawkat https://orcid.org/0000-0003-4823-4562

References

[1] Ciresan D C, Meier U, Masci I, Gambardella L M and Schmidhuber J 2011 Flexible, high performance convolutional neural networks for image classification Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence vol 2 (Barcelona, Catalonia: AAAI Press) pp 1237–42
[2] Krizhevsky A, Sutskever I and Hinton G E 2017 Commun. ACM 60 84–90
[3] Rumelhart D E, Hinton G E and Williams R J 1986 Learning Internal Representations by Error Propagation (Cambridge, MA: MIT Press) pp 318–62
[4] Jouppi N P et al 2017 Comput. Archit. News 45 1–12
[5] Jouppi N P et al 2017 In-datacenter performance analysis of a tensor processing unit Proc. of the Annual Int. Symp. on Computer Architecture (ACM) pp 1–12
[6] Cadambi S, Majumdar A, Becchi M, Chakradhar S and Graf H P 2010 A programmable parallel accelerator for learning and classification Proc. of the ACM Int. Conf. on Parallel Architectures and Compilation Techniques (Vienna, Austria) pp 273–84
[7] Aydonat U, O’Connell S, Capalija D, Láng A C and Chiu G R 2017 An OpenCL® deep learning accelerator on Arria 10 Proc. of the ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays (New York, NY, USA) pp 55–64
[8] Ling A C et al 2018 Harnessing numerical flexibility for deep learning on fpgas Proc. of the ACM Int. Symp. on Highly-Effective Accelerators and Reconfigurable Technologies (New York, NY, USA)
[9] Wang Q, Wang X, Lee S H, Meng F H and Lu W D 2019 A deep neural network accelerator based on tiled rram architecture Proc. of the IEEE Int. Electron Devices Meeting (IEDM) pp 14.4.1–4
[10] Song L, Qian X, Li H and Chen Y 2017 Pipelayer: a pipelined ReRAM-based accelerator for deep learning Proc. of the IEEE Int. Symp. on High Performance Computer Architecture (HPCA) pp 541–52
[11] Luo Y and Yu S 2020 IEEE Trans. Comput. 69 1113–27
[12] Meng J, Yang L, Peng X, Yu S, Fan D and Seo J-S 2021 IEEE Trans. Circuits Syst. II 68 1576–80
[13] Misra J and Saha I 2010 Neurocomputing 74 239–55
[14] Mead C 1989 Analog VLSI and Neural Systems (Reading, MA: Addison-Wesley)
[15] Indiveri G et al 2011 Frontiers Neurosci. 5 73
[16] Hodgkin A L and Huxley A F 1952 J. Physiol. 117 500–44
[17] Izhikevich E M 2003 IEEE Trans. Neural Netw. 14 1569–72
[18] Aamir S A, Müller P, Hartel A, Schemmel J and Meier K 2016 A highly tunable 65 nm CMOS LIF neuron for a large scale neuromorphic system ESSCIRC Conf. 2016: 42nd European Solid-State Circuits Conf. (IEEE) pp 71–4
[19] Cassidy A S, Georgiou J and Andreou A G 2013 Neural Netw. 45 4–26
[20] Rosenblatt F 1958 Psychol. Rev. 65 386–408
[21] Rosenblatt F 1962 Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms (Washington, D.C.: Spartan Books)
[22] McCulloch W S and Pitts W 1943 Bull. Math. Biophys. 5 115–33
[23] Hebb D O 1949 The Organization of Behavior: A Neuropsychological Theory (New York: Wiley)
[24] Bishop C M 2006 Pattern Recognition and Machine Learning (New York: Springer)
[25] Minsky M and Papert S 1969 Perceptrons: An Introduction to Computational Geometry (Cambridge, MA: MIT Press)
[26] Mead C 1990 Proc. IEEE 78 1629–36
[27] Farquhar E and Hasler P 2005 IEEE Trans. Circuits Syst. I 52 477–88
[28] Chua L 1971 IEEE Trans. Circuit Theory 18 507–19
[29] Strukov D B, Snider G S, Stewart D R and Williams R S 2008 Nature 453 80–3
[30] Bartolozzi C and Indiveri G 2007 Neural Comput. 19 2581–603
[31] Peng L, Wang D, Zhang Z, Tang P, Li G and Pei J 2016 Phys. Lett. A 380 903–9
[32] Indiveri G, Stefanini F and Chiaca E 2010 Spike-based learning with a generalized integrate and fire silicon neuron Proc. of 2010 IEEE Int. Symp. on Circuits and Systems (IEEE) pp 1951–4
[33] Wu X, Saxena V and Campbell K A 2014 Energy-efficient STDP-based learning circuits with memristor synapses Machine Intelligence and Bio-Inspired Computation: Theory and Applications VIII vol 9119 (International Society for Optics and Photonics) p 911906
[34] Indiveri G 2003 A low-power adaptive integrate-and-fire neuron circuit Proc. of the Int. Symp. on Circuits and Systems (ISCAS) vol 4 (Bangkok, Thailand)
[35] Mathuramalingam A, Himavathi S and Srinivasan E 2008 Int. J. Inf. Technol. 4 86–92
[36] Hikawa H 2003 IEEE Trans. Neural Netw. 14 1028–37
[37] Shave-Taylor J, Jeavons P and Daalen M V 1991 Comput. Sci. 3 317–28
[38] van Daalen M, Kosel T, Shave-Taylor J and Jeavons P 1994 Electron. Lett. 30 331–3
[39] Skrbek M 1999 Neural Netw. World 9 375–91
[40] Sayyaparaju S, Adnan M M, Amer S and Rose G S 2020 J. Emerg. Technol. Comput. Syst. 16 1–25

21
[146] Yin B, Corradi F and Bohte S M 2021 Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks (arXiv:2103.12593)

[147] Bellec G, Scherr F, Subramoney A, Hajek E, Salaj D, Legenstein R and Maass W 2020 Nat. Commun. 11 3625