Design and Implementation of Multipattern Generators in Analog VLSI

Ryan J. Kier, *Student Member, IEEE*, Jeffrey C. Ames, Randall D. Beer, and Reid R. Harrison, *Member, IEEE*

***Abstract*—In recent years, computational biologists have shown through simulation that small neural networks with fixed connectivity are capable of producing multiple output rhythms in response to transient inputs. It is believed that such networks may play a key role in certain biological behaviors such as dynamic gait control. In this paper, we present a novel method for designing continuous-time recurrent neural networks (CTRNNs) that contain multiple embedded limit cycles, and we show that it is possible to switch the networks between these embedded limit cycles with simple transient inputs. We also describe the design and testing of a fully integrated four-neuron CTRNN chip that is used to implement the neural network pattern generators. We provide two example multipattern generators and show that the measured waveforms from the chip agree well with numerical simulations.**

***Index Terms*—Analog neural network, analog VLSI, central pattern generator (CPG) implementations, continuous-time recurrent neural network (CTRNN), multipattern generators.**

I. Introduction

A. Central Pattern Generators

MANY activities vital to animal survival such as walking, breathing, and digestion require repetitive, rhythmic activation of an animal's muscles. These rhythmic muscle contractions and relaxations are controlled by specialized neural networks called central pattern generators (CPGs) [1]. CPGs have been studied by biologists and neuroscientists since the early 1900s, and two key principles have emerged from these studies. First, CPGs are independent neural networks located outside of the brain, i.e., control of locomotion is distributed. Second, though sensory feedback and descending inputs from the brain (i.e., the cerebellum) can improve the quality of the motor patterns, CPGs are capable of sustaining rhythmic outputs in the absence of such inputs [2], [3].

Modeling CPGs provides a way of testing existing theories about their behavior. For example, it has long been observed that four-legged animals vary their gait according to their speed [4], yet experimentation has not made clear the underlying mechanisms responsible for such behavior. Recently, researchers have demonstrated that multiple output rhythms, similar to those observed in animal gait control, can be produced by small neural networks with fixed connectivity [5], [6]. Furthermore, these studies have shown that, like their biological counterparts, the artificial networks are *multistable* or reconfigurable, i.e., they can be switched between the multiple output patterns through the application of a transient input. These studies employed biologically realistic (Hodgkin–Huxley type) neuron models which involve several differential equations per neuron. In this paper, we show that the same qualitative reconfigurable behavior can be realized using a significantly less complex, and hence simpler to implement, neuron model.

Our work is similar to that demonstrated in [7]. However, in this prior work, the authors were able to embed specific patterns into a network, but their method required a neuron model with two synaptic connections per neuron: one fast time constant connection to stabilize the network state and one slow time constant connection to compute the network's next state. In contrast, we employ a model neuron that requires only a single time constant per neuron while maintaining the ability to embed specific patterns into a network.

B. Applications

Biology often offers attractive, elegant solutions to engineering problems. Take, for example, legged robotics, where engineers often place an emphasis on centralized control. In contrast, biological systems rarely make use of centralized control; more often control is distributed throughout the body. CPGs are a classic example of this distributed control. CPGs handle the low-level limb coordination required for walking, leaving the brain available for higher level tasks such as navigation and obstacle avoidance. Implementing this type of distributed control with neural networks has been difficult in engineered systems because simulation of even the simplest neural network models requires significant computational power. As a result, PCs were used to simulate numerically neural networks in many of the early legged robotic systems which used CPG-based control (e.g., [8] and [9]). This configuration requires the robot to be tethered to an umbilical cord, limiting its range and usefulness. Ideally, a hardware neural network and a small microcontroller could be used onboard the robot, thereby removing the need for an undesirable umbilical cord. The microcontroller could handle the robot's higher functions (i.e., decision-making, navigation, etc.), leaving the low-level limb coordination to the neural network controller.

Although hardware (both analog and digital) neural networks have been around for some time, they have not been well-suited for CPG implementations because they usually lacked dynamics. However, in recent years, hardware CPG implementations have received more attention. Some work has

Manuscript received January 5, 2005; revised October 27, 2005. This work was supported by the National Science Foundation under Grant EIA-0130773.

R. J. Kier and R. R. Harrison are with the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112 USA (e-mail: kier@eng.utah.edu).

J. C. Ames and R. D. Beer are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106 USA.

Digital Object Identifier 10.1109/TNN.2006.875983

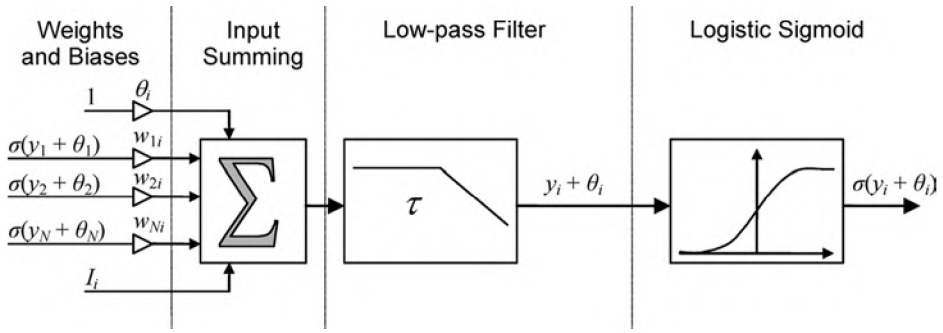

Fig. 1. Block diagram of a single CTRNN neuron. Note that the bias input θ_i is applied to the input of the low-pass filter rather than its output. Because θ_i is a constant, this system is equivalent to that described by (1).

been inspired by observations of CPGs in swimming animals. For example, an analog CPG based on coupled intrinsic oscillators was developed in [10]. In contrast, other researchers have developed CPGs for quadrupedal gait control [11]. Finally, other researchers have been working on bipedal locomotion controllers [12]–[14]. All of these hardware CPGs are capable of creating sustained output rhythms similar to those found in biological systems. However, none of these previous implementations has demonstrated reconfigurable (i.e., multistable) behavior.

In this paper, we present a new implementation of an artificial CPG that can be switched between multiple output patterns via brief transient inputs. In Section II, we review the continuous-time neural network (CTRNN) neuron model used in this paper. Next, Section III describes the design method used to embed multiple limit cycles (i.e., patterns) into a fixed-connectivity CTRNN. Additionally, we present two illustrative examples of the design procedure. Section IV describes the analog circuitry used to implement the CTRNN neurons. In Section V, results from our CPG chips are presented and compared with our simulations for the two example networks designed in Section III. Finally, we give some conclusions and future directions for this work in Section VI.

II. Continuous-Time Recurrent Neural Networks (CTRNNs)

The state of each neuron in a CTRNN is described by the first-order differential equation

$$\tau_i \cdot \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ji} \cdot \sigma(y_j + \theta_j) + I_i \qquad (1)$$

where y_i is the state of the i^{th} neuron, τ_i is the time constant of the ith neuron, θ_j is a bias for the jth neuron, w_{ji} is the synaptic weight from the jth neuron to the ith neuron, $\sigma(x)$ is the logistic sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

and I_i is an external input to the ith neuron [15]. CTRNNs are identical to Hopfield's continuous model [16] except that the connection weights between any pair of neurons need not be symmetric and self-connections are allowed. Fig. 1 shows a block diagram of a single neuron corresponding to the description given by (1).

While this artificial neuron model is much simpler than other, more biologically accurate neuron models (e.g., the Hodgkin–Huxley model), it remains attractive for many reasons.

1) It is computationally inexpensive to simulate; hence offline simulations can be used to quickly evolve desirable network patterns using genetic algorithms.
2) The model is mathematically tractable.
3) CTRNNs are universal approximators of smooth dynamics [17].
4) The CTRNN neuron has a plausible biological interpretation. The mean membrane potential is represented by y_i, θ_i captures the firing threshold, and τ_i corresponds to the membrane time constant. The parameter $\sigma(y_i+\theta_i)$ represents the mean firing rate while w_{ji} describes the synaptic interactions with the self-connections (w_{ii}) endowing each neuron with bistability properties.
5) The model is amenable to analog circuit implementation.

III. Design of Multipattern Generators

A. Design Method

In this section, we describe a way to program the weights and biases of a CTRNN so that it generates multiple desired temporal patterns. Each pattern consists of a sequence of quasi-stable states in which different combinations of neurons are saturated on and off. Although the patterns will typically be cycles, they need not be. We use previous work on the dynamics of CTRNNs to derive a set of inequalities involving the connection weights and biases from a specification of the desired temporal patterns. If this set of inequalities is solvable, then any solution will lead to a CTRNN that exhibits the desired patterns. Because the neurons are being operated in saturation, the dynamics of the resulting CTRNN resembles a finite-state machine and part of the design process is reminiscent of conventional state machine design techniques.

A useful way of looking at the dynamics of the individual neurons of a given network is in terms of their steady-state output. For a given input, a neuron will eventually settle on some output value, which we can plot versus the input value to generate a

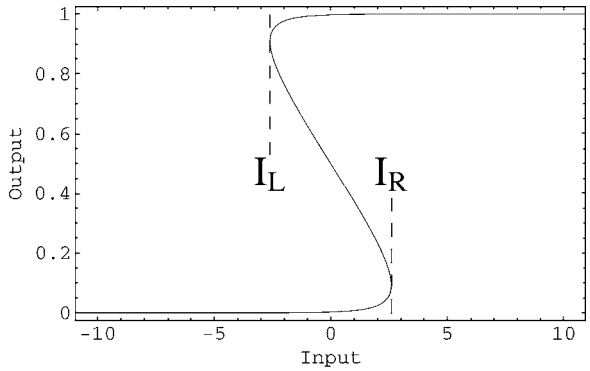

Fig. 2. Steady-state input/output (SSIO) curve corresponding to a self-weight of 12 and bias of -6. This curve shows how the location of the equilibrium points of a single neuron changes as a function of the synaptic input. In the folded region, the system is bistable, where the outer branches are stable and the inner branch is unstable.

steady-state input/output (SSIO) curve [15]. An SSIO curve is obtained by setting (1) for a single neuron to zero, solving for I in terms of y, and then plotting $\sigma(y+\theta)$ versus I. If the self-weight is less than four, there will be only a single stable equilibrium point for each input. But when the self-weight goes above four, the SSIO curve folds back on itself, so that there is a region of bistability—two stable equilibrium points separated by an unstable equilibrium point. In general, the self-weight boundary between unistability and multistability will be determined by the reciprocal of the maximum of the activation function derivative; the number 4 arises here because the maximum of $\sigma'(x)$ is 1/4 [15].

Fig. 2 shows an SSIO curve for a self-weight of 12 and a bias of -6. The horizontal axis is the total input I to the neuron [not including the self-input, which enters nonlinearly into (1)], and the vertical axis is the neuron's output $\sigma(y+\theta)$. In this case, the bistable region lies between inputs of -2.607 and 2.607.

The left and right boundaries of this region, denoted I_L and I_R, indicate where a neuron will make a transition from on to off (I_L) or from off to on (I_R). The boundary values for the bistable region of the SSIO curve are functions of the neuron's self-weight w and bias θ [15]

$$I_L(w,\theta) = 2\ln\left(\frac{\sqrt{w}+\sqrt{w-4}}{2}\right) - \frac{w+\sqrt{w(w-4)}}{2} - \theta \tag{3}$$

$$I_R(w,\theta) = -2\ln\left(\frac{\sqrt{w}+\sqrt{w-4}}{2}\right) - \frac{w-\sqrt{w(w-4)}}{2} - \theta. \tag{4}$$

These expressions are obtained by solving for the simultaneous zeros of the single-neuron vector field and its derivative to pick out the manifold of saddle-node bifurcations that delineate the boundary between unistable and bistable behavior.

In this section, we look at using the SSIO curves to predict when a neuron will transition from off to on, based on the inputs of the other neurons. To simplify the analysis, we assume that all neurons have binary outputs, and that transitions occur instantaneously. When neurons are driven sufficiently far into saturation, the binary approximation is quite accurate.

We can form the network state at any given time by assuming each neuron is either on (1) or off (0), and concatenating these states to make a binary vector. We also define allowed transitions between states by assuming that only one neuron can change state at a time. Fig. 3 shows all possible transitions between states for three- and four-neuron networks. With these assumptions, the states and possible transitions for an N-neuron network form an N-dimensional hypercube.

For simplicity, we require that all neurons be bistable, which requires that $w > 4$. We set $w = 12$, to give a fairly curved SSIO diagram, with a bistable region of width 5.21. Using a fixed self-weight reduces (3) and (4) to linear functions of the neuron's bias.

Given a desired transition from one network state A to a second state B, we can compute an inequality corresponding to this transition as follows. Denote the neuron changing from state A to state B as the jth neuron. Then the synaptic input to the jth neuron, $\sum_{i=1,i\neq j}^{N} w_{ij}\sigma(y_i+\theta_i)$, must be greater than $I_R(w_j,\theta_j)$ if it is turning on, or less than $I_L(w_j,\theta_j)$ if it is turning off. Note that, assuming that the external input I_j is zero (which is reasonable for a CPG), the inequality generated is only a function of the weights on edges coming into the jth neuron and the bias of the jth neuron. Therefore, the inequalities for each neuron will be independent of those for any other neuron and can be solved separately. In total, we must solve the inequalities for N^2-N weights and N biases.

We also need to restrict the other transitions involving the states we have specified, by requiring that every network state that has a defined transition leaving it have no other transition leaving it; that is, that every network state we include in our cycles be unambiguous regarding what its next state will be. As a direct consequence, any cycles we wish to embed in the network must be vertex disjoint. They can overlap in their use of neurons, but not in their use of network states.

Furthermore, the system is never allowed to transition from a state A to another state B and back to A without an intermediate state. Because the self-weight is not taken as part of the neural input, relative to the SSIO curve, the transition from one state to another and its reverse have the same input value to the neuron, and it can only be above I_R or below I_L, but not both. Due to the mutual exclusivity of this situation, it is not necessary to include explicit restrictions of these backward transitions. Note that without the binary neuron assumption, neurons could change their outputs slightly, without crossing the 0.5 threshold, and then such a cycle would be possible.

We also must be sure that the cycles we program in are in fact stable limit cycles. We test this empirically, by perturbing each cycle in the system by small random amounts and verifying that it settles back onto the limit cycle. For each of the circuits we generate, we perform this test for each cycle, applying a constant external input to each neuron, of a magnitude randomly generated, uniformly distributed in the range of [−1,1]. All cycles shown in this paper are stable.

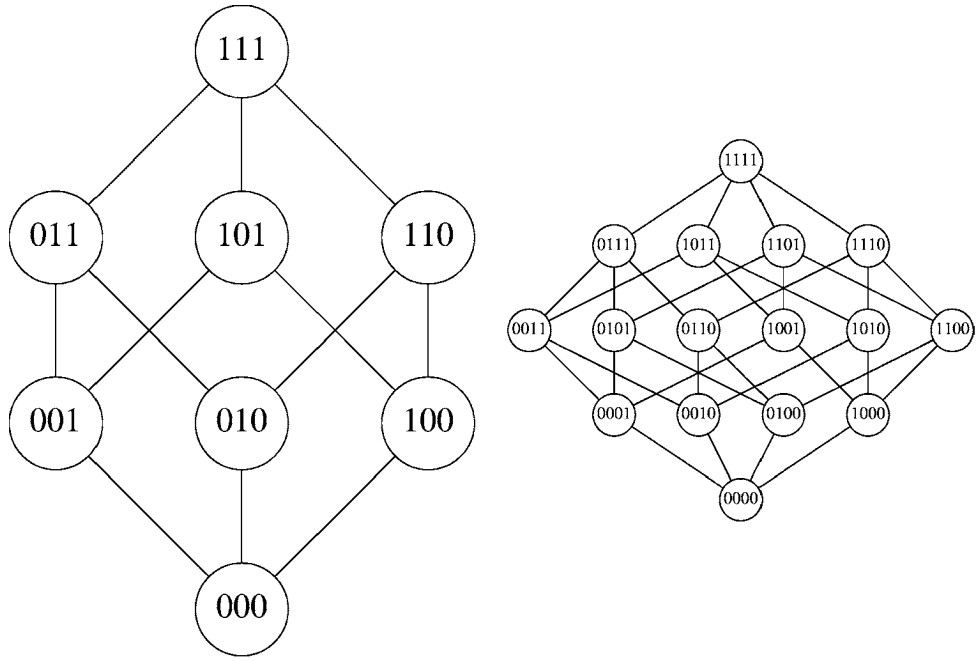

Fig. 3. Possible transitions in three- and four-neuron networks. Circles indicate network states, labeled by the outputs of each neuron when in that state. Edges indicate possible transitions, where one neuron is allowed to change state at a time. Note that these maps are isomorphic to the subset inclusion ordering of the power sets of sets of three and four elements, respectively.

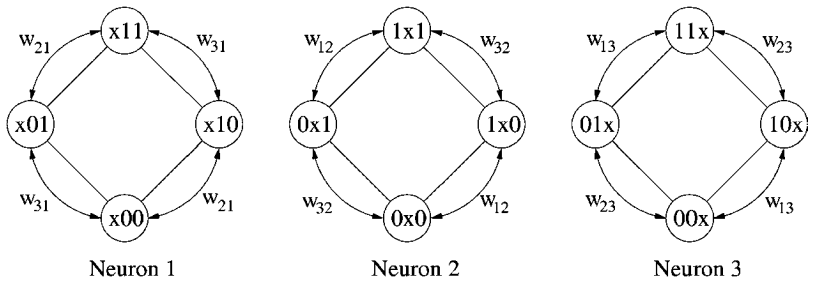

Fig. 4. Diagrams of systems A_1, A_2, and A_3 after independent inequalities in a three-neuron system are separated. Circles indicate network states, labeled by the outputs of all neurons besides the one transitioning (designated "x"). Straight lines indicate subset inclusion. Arcs indicate which states are flipped when the corresponding weight changes sign. Using these arcs and our transition ordering, the solvability of the system can be predicted.

B. Solvability

We have shown how to generate a system of inequalities corresponding to a set of desired transitions between network states, but the question arises of whether this system will always be solvable; that is, can arbitrary patterns be embedded in an N-neuron network? To clarify this question, we develop a formal specification of the task at hand.

Since each neuron's inequalities involve only its own self-weight, bias, and the weights of incoming connections, we can separate the inequalities by the neuron they relate to. In an N-neuron system, this will give us N new systems $A_1, A_2, \ldots, A_N$ containing 2^{N-1} states each, where in system A_i all inequalities are functions of w_{ii}, θ_i, and weights $w_{ji} (j \neq i)$. Fig. 4 shows the three systems generated when $N = 3$.

Let $T = \{L, R, \emptyset\}$ represent the set of possible transitions from a given network state, where L indicates transitioning off, R indicates transitioning on, and $\emptyset$ indicates that no transition is allowed. The input to the neuron is $x = \sum_{i=1, i \neq j}^{N} w_{ij} y_i$. Recall that according to the SSIO curve (see Fig. 2), if $x \leq I_L$, the neuron will settle in the "off" state; if $I_L < x < I_R$, the neuron will remain in its current state; and if $I_R \leq x$, the neuron will settle in the "on" state. Using this idea of the natural ordering on the input from other neurons, we create an ordering on T. We define a binary relation $\preceq$ on T such that $L \preceq \emptyset \preceq R$. Then T combined with $\preceq$ forms a totally ordered set.

Let Y be the set of binary vectors of length $N - 1$. Then we define a function $\varphi : Y \rightarrow T$, which, for a given network state $y \in Y$ (the outputs of all other neurons), describes the transition $t \in T$ allowed. Note that Y along with the subset inclusion relation $\subseteq$ forms a partially ordered set. This raises the question of whether φ is order-preserving (that is, whether, given $a, b \in Y$, $a \subseteq b$ implies $\varphi(a) \preceq \varphi(b)$).

It appears that if φ is order-preserving, the system of inequalities A_i must have a solution. For $a, b \in Y$, $a \subseteq b$, a represents a state receiving less input from other neurons than b. Thus it would make sense that the range of the SSIO curve accessible to a should also be "less than" b's, in the sense that corresponding boundaries (left or right) of a's range should be less than those

TABLE I
DESIRED TRANSITION INEQUALITIES FOR EXAMPLE 1

From	To	Inequality	From	To	Inequality
0000	0001	$0 > I_R(w_{44}, \theta_4)$	0010	0110	$w_{32} > I_R(w_{22}, \theta_2)$
0001	0011	$w_{43} > I_R(w_{33}, \theta_3)$	0110	0100	$w_{23} < I_L(w_{33}, \theta_3)$
0011	0111	$w_{32} + w_{42} > I_R(w_{22}, \theta_2)$	0100	0101	$w_{24} > I_R(w_{44}, \theta_4)$
0111	1111	$w_{21} + w_{31} + w_{41} > I_R(w_{11}, \theta_1)$	0101	1101	$w_{21} + w_{41} > I_R(w_{11}, \theta_1)$
1111	1110	$w_{14} + w_{24} + w_{34} < I_L(w_{44}, \theta_4)$	1101	1001	$w_{12} + w_{42} < I_L(w_{22}, \theta_2)$
1110	1100	$w_{13} + w_{23} < I_L(w_{33}, \theta_3)$	1001	1011	$w_{13} + w_{43} > I_R(w_{33}, \theta_3)$
1100	1000	$w_{12} < I_L(w_{22}, \theta_2)$	1011	1010	$w_{14} + w_{34} < I_L(w_{44}, \theta_4)$
1000	0000	$0 < I_L(w_{11}, \theta_1)$	1010	0010	$w_{31} < I_L(w_{11}, \theta_1)$

of b. Some care is necessary in stating this, as the SSIO curve's bistable region may not be fully within the range reachable from any network states, for some values of the neuron self-weight. We can use the bias to shift the SSIO curve left and right, but only to a limited degree. Exactly how much of a restriction this is, and in exactly what cases the system is solvable, remains for future work.

This gives us a tentative criterion for determining whether a system is solvable. However, we have thus far implicitly assumed that input from other neurons is positive. In the CTRNN, this is not always the case. Specifically, the input's sign is determined by the weight w_{ij} on the input coming from neuron i into the jth neuron, and these weights are variables in our systems of inequalities.

What happens when we flip a given weight, say, w_{ij}, from positive to negative? Let us define the vector S of signs of the weights on connections coming into neuron j as $S = (s_1, s_2, \ldots, s_{j-1}, s_{j+1}, \ldots, s_N)$, where each s_i is equal to 1 if the corresponding weight w_{ij} is positive and -1 if w_{ij} is negative.

Suppose we have an S where each $s_i = 1$. Choose one weight w_{kj} and make it negative, so $s_k = -1$, and call this new vector S'. In the case of S', a Y where $y_k = 0$ will yield a greater net input x to neuron j than will a Y where $y_k = 1$. If φ is the mapping we use in the case of S, let us create a new mapping φ' based on φ which reflects the fact that y_k is now subtracting from our net input. For each $y \in Y$, let y' be the vector such that $y'_i = y_i$ if $i \neq k$ and $y'_k = 1 - y_k$. Then define $\varphi'(y) = \varphi(y')$. Roughly speaking, φ' is φ "reflected" across the kth axis of its domain. The arcs in Fig. 4 labeled by a weight designation show how this reflection can be visualized in the case $N = 3$. Therefore, by manipulating the sign of the interneuron weights, we can create a family of functions $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_K\}$, where K, the number of such functions, is equal to 2^{N-1}, since we have $N-1$ weights we can make positive or negative.

If any one of these functions is order-preserving, it seems that the system can be solved. And conversely, it seems that if none of the functions preserves order, there will be no solution to the system of inequalities. The caveats mentioned earlier regarding the SSIO curve's exact position versus the range of inputs realizable by the network also apply here, and we leave the specification of the exact circumstances in which this will limit us to future work. For our present purposes, this rudimentary method alone is sufficient to guide the design process and allow us to embed multiple limit cycles.

In summary, we have shown one way to formalize the conditions under which the set of inequalities corresponding to a desired set of transitions is solvable. A number of questions remain open to further research. For example, rather than testing all 2^{N-1} mappings from Y to T, is there a general property of the cycles that can be used to determine the solvability? If a given set of inequalities is not solvable, can we transform it into a solvable system by adding a minimal number of neurons to separate conflicting transitions? Can the approach be generalized to allow multiple neurons to change state simultaneously or to allow the duration of the transitions to be programmed rather than treating them as instantaneous?

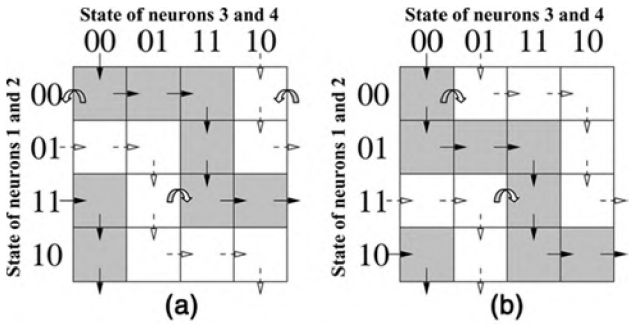

Fig. 5. Two embeddings of pairs of eight cycles in four-neuron state space. Straight arrows indicate embedded transitions, while curved arrows indicate transitions caused by external inputs that switch from one embedded cycle to the other. For both examples, one of the embedded patterns is shaded in gray while the other embedded pattern is not shaded. (a) Example 1. (b) Example 2.

C. Examples

Although a number of open theoretical questions remain about the general solvability of an arbitrary set of transitions, we demonstrate in this section that the method can be practically applied. Specifically, we present two examples of pairs of embedded cycles. Both examples create two cycles of length eight in four-neuron networks, making use of every network state available. The state transitions are shown in Fig. 5.

The first example's cycles, as shown in Fig. 5(a), are {0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000} and {0010, 0110, 0100, 0101, 1101, 1001, 1011, 1010}. We convert each of these transitions to an inequality, based on which neuron is transitioning and what the originating network state is. For example, in the transition {0000, 0001}, neurons 1 through 3 are off, while neuron 4 is turning on. So neuron 4's internal state must be pushed above I_R. Thus the inequality becomes $0 * w_{14} + 0 * w_{24} + 0 * w_{34} > I_R(w_{44}, \theta_4)$, or simply $0 > I_R(w_{44}, \theta_4)$. The set of inequalities this generates is shown in Table I.

TABLE II
SELECTED FORBIDDEN TRANSITION INEQUALITIES FOR EXAMPLE 1

From	To		From	To	Inequality
0000	0010	$0 < I_R(w_{33}, \theta_3)$	1111	1101	$w_{13} + w_{23} + w_{43} > I_L(w_{33}, \theta_3)$
0000	0100	$0 < I_R(w_{22}, \theta_2)$	1111	1011	$w_{12} + w_{32} + w_{42} > I_L(w_{22}, \theta_2)$

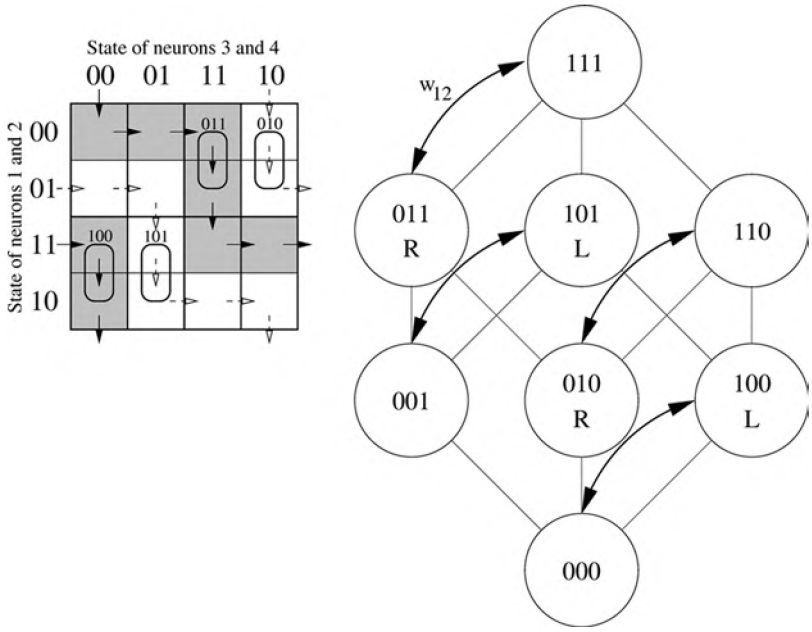

Fig. 6. Example of solvability method with respect to neuron 2, using the transitions defined for Example 1. The transition diagram on the left shows the transitions that neuron 2 undergoes (circled and labeled with the network state). These correspond to the L and R states labeled in the network diagram on the right (the other states have no transition, corresponding to the $\emptyset$ state, but are not marked here for clarity). The straight lines between network states illustrate the subset inclusion ordering. The four dark arcs show the effect of flipping the sign of weight w_{12}, which in this case is necessary to maintain the ordering $L \preceq \emptyset \preceq R$.

As mentioned above, we also want to restrict these neural states so that they cannot transition elsewhere. Each state could potentially transition to three other states in the network (excluding a self-transition), namely, those resulting from any of the other three neurons changing state, so we must explicitly forbid these. Thus there are a total of 32 inequalities for the forbidden transitions. The way we forbid transitions is by requiring that the neuron's internal state be greater than I_L for a forbidden "off" transition and less than I_R for a forbidden "on" transition. For example, Table II shows the inequalities corresponding to the two forbidden transitions from network state 0000 and from 1111.

Once we build a complete list of desired and forbidden transition inequalities, we can generate a parameter set that simultaneously satisfies these inequalities. Since we sometimes need to try multiple configurations to get one where all the limit cycles we want are stable, we generally pick parameters randomly within the valid ranges. While a more sophisticated method of solution will likely be necessary for larger networks, this method has been sufficient for the examples presented here. Picking random parameters in this case, we arrive at the following weight matrix and biases, where the row number designates the originating neuron and the column number designates the destination neuron:

$$w = \begin{bmatrix} 12 & -11.7235 & -3.44359 & -6.63136 \\ 7.18571 & 12 & -10.5646 & 2.54641 \\ -0.624068 & 8.93508 & 12 & -7.26261 \\ 7.31 & -1.47654 & 10.2795 & 12 \end{bmatrix}$$

$$\theta = [-10.8512 \quad -3.64164 \quad -4.78549 \quad -0.415379].$$

The solvability method described in the previous section can be illustrated using this example (Fig. 6). There are four transitions of neuron 2's state (circled in the state diagram on the left), which create two 'L' and two 'R' states in the lattice on the right (for clarity, '$\emptyset$' states are not marked). Because in this case, we have some states where $\emptyset \preceq L$ (e.g., 000 and 100) and some states where $R \preceq \emptyset$ (e.g., 011 and 111), we can deduce that we must flip the sign of weight w_{12} (shown by the dark arrows) to ensure the ordering L $\preceq \emptyset \preceq$ R. The other two weights (w_{32} and w_{42}) can be either positive or negative. This agrees with the empirical results shown in the weight matrix.

The second example's cycles, shown in Fig. 5(b), are {0001, 0011, 0010, 0110, 1110, 1100, 1101, 1001} and {0000, 0100, 0101, 0111, 1111, 1011, 1010, 1000}. We use the same method

to generate inequalities, but in this case we restrict the possible parameter values to integers due to the 5-bit precision of the synapse circuitry. Thus we obtain the following weight matrix and biases:

$$w = \begin{bmatrix} 12 & -5 & -5 & 0 \\ 5 & 12 & 0 & 5 \\ 5 & 0 & 12 & -5 \\ 0 & -5 & 5 & 12 \end{bmatrix}$$
$$\theta = [-11 \quad -1 \quad -6 \quad -6].$$

Fig. 7 shows the simulation results for both example networks. The thick bars represent the duration of a transient $I_i = 10$ input to the neuron. Note that the time constant parameters for each neuron do not enter into the design procedure, hence they can be chosen arbitrarily to scale the pattern frequency. For the simulations shown in Fig. 7, all time constants were set to one.

IV. Analog VLSI Implementation

A. Silicon Neuron Overview

We have designed and tested a four-neuron CPG chip in AMI's 1.5 μm CMOS process. The block diagram and a labeled die photo of the CTRNN chip are shown in Fig. 8. The chip contains a fully programmable four-neuron CTRNN along with a suite of test devices. The chip uses ± 2.5 V supply voltages. Each neuron on the chip implements the CTRNN model and is fully programmable. Fig. 9 shows the schematic for one neuron on the chip with boxes indicating the major subcircuits that correspond to the block diagram in Fig. 1. Note that for the convenience of implementation, the biases θ_i are summed with the synaptic inputs rather than the neuron state y_i. This change does not affect the dynamics of the CTRNN because the bias is a constant and unaffected by the low-pass filter operation in (1). The weights and biases of each neuron are programmable on the range $-15 \leq w_{ji} \leq 15$ via 5-bit multiplying digital-to-analog converters (MDACs). The time constants, which are set by the bias currents to G_m-C filters, also are programmed with 5-bit precision on the range $0 < \tau_i \leq 0.8$ s. Additionally, each neuron has an analog voltage input-$V_{\mathrm{REF}i}$, corresponding to the I_i parameter in (1), that can be used either to provide sensory feedback or to switch the CPG between stable limit cycles. Finally, the output of each neuron is an analog current $i_{\mathrm{OUT}}i$ representing the neuron's mean firing rate. In the following sections, we will discuss each neuron subcircuit in greater detail.

B. Sigmoid Circuit

The CTRNN model in (1) employs the standard logistic sigmoid function (2) to provide a mapping from a neuron's membrane potential to its mean firing rate. The sigmoid circuit in Fig. 9 encodes the neuron membrane potential as a voltage while the mean firing rate is encoded as a current. It is a simple four-transistor CMOS circuit based on a differential pair biased in the subthreshold region of operation by an nMOS current source ($\mathrm{M_{B2}}$). The output is taken as the drain current $i_{\mathrm{OUT}}i$ flowing through the noninverting input transistor $\mathrm{M_1}$. The diode-connected pMOS transistor $\mathrm{M_{OUT}}$ serves as the input of a current mirror that copies the output current to the inputs of the synapse circuitry. The input to the circuit is taken as v_{LPF}, the gate voltage of the noninverting input transistor $\mathrm{M_1}$. The inverting input of the differential pair (the gate of $\mathrm{M_2}$) is connected to a reference voltage $V_{\mathrm{REF}i}$. This reference voltage is normally at circuit ground (i.e., halfway between the positive and negative power supplies). However, $V_{\mathrm{REF}i}$ can take on large transient values to realize reconfigurablility [i.e., it can serve as the I_i inputs in (1)]. It can be shown that the output current is given by

$$i_{\mathrm{OUT}_i} = \frac{I_B}{1 + e^{-\frac{\kappa v_{\mathrm{LPF}} - v_{\mathrm{REF}}}{U_T}}} \tag{5}$$

where $U_T = kT/q \approx 26$ mV is the thermal voltage, $\kappa \approx 0.7$ is the subthreshold gate coupling coefficient, and I_B is the bias current (100 nA) [18]. Comparing (2) and (5) reveals that except for a scaling factor of κ/U_T on the input, the circuit directly implements the logistic sigmoid function. This scaling factor does not present a serious problem because it can be compensated for elsewhere in the neuron circuit.

C. Synapse Circuitry

The output of each neuron $i_{\mathrm{OUT}}i$ is copied to the input of every neuron in the network. As shown in Fig. 9, this is easily accomplished with a cascoded diode-connected transistor (M_{OUT}) which forms the input to a pMOS current mirror. Each neuron in the circuit has an identical transistor to complete the current mirror. The strength of the connection between a pair of neurons is controlled by a programmable 5-bit MDAC synapse.

In contrast to the more common synapse circuits that use compact analog multipliers and capacitive weight storage [19]–[22] or floating gate circuitry [23]–[27], the synapse circuit used in this paper employs a hybrid analog-digital approach. MDAC synapses are usually based on cascoded current mirror DACs [14], [19], [28], [29]. This paper employs a current-mode MDAC synapse that is different than the commonly used cascoded current mirror MDAC mentioned above. The circuit is similar to the mini-DAC calibration technique for artificial neural networks that has been reported recently [30].

The MDAC synapse circuit in Fig. 10 is based on a resistive R-$2R$ DAC. This circuit is similar to our previously reported MDAC synapse [31], [32]. The primary difference between the two circuits lies in the sign bit circuitry. In this version, the output is always directed through a cascoded nMOS current mirror to reduce finite drain resistance effects due to variations in output voltage. However, the R-$2R$ divider network core remains the same.

It is clear from Fig. 10 that resistors have not been used to implement the R-$2R$ divider. Instead, a network of pMOS transistors is used to perform the linear current division [33], [34]. All the transistors in Fig. 10 have the same width-to-length ratio. Therefore, series combinations of the branch transistors ($\mathrm{M_{Bx}}$) and the activated switch transistors ($\mathrm{M_{SWxa}}$ or $\mathrm{M_{SWxb}}$) are used to realize the $2R$ branches of the divider. Note that only one

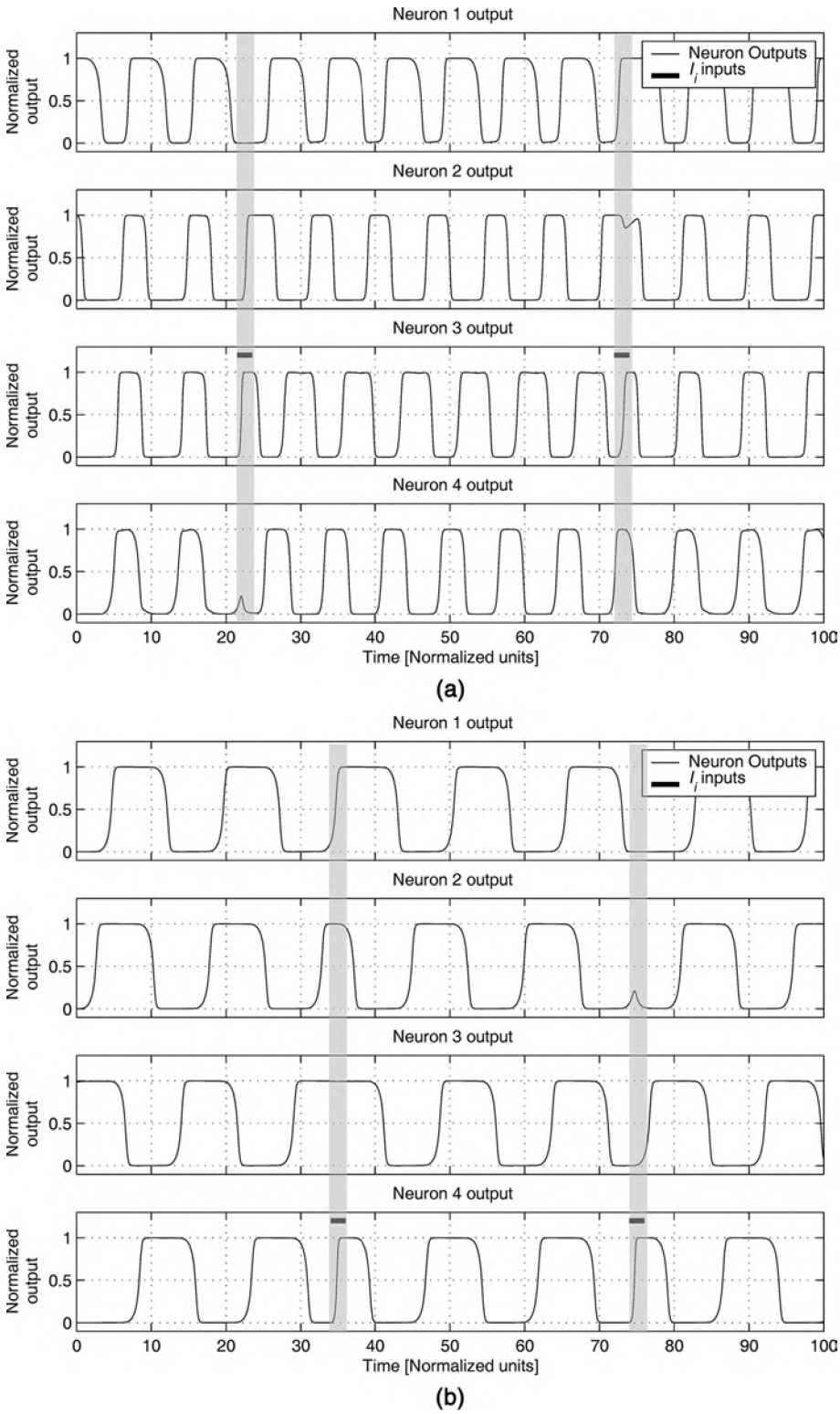

Fig. 7. Simulated output patterns for the two example CTRNNs. The thick bar represents the duration of an $I_i = 10$ external input used to switch between output patterns. Both figures illustrate the following sequence of events for the two example networks. 1) The network is shown initially oscillating in one of the two embedded limit cycles. 2) A transient external input is applied to one neuron's I_i input causing the network to enter a new part of the state space where the alternate embedded limit cycle dominates. 3) The network is allowed to oscillate in the alternate limit cycle for a few cycles. 4) A transient external input is applied a second time causing the network to return to the original limit cycle. (a) Example 1 CTRNN. Neuron 3's external input is used to control the output pattern. (b) Example 2 CTRNN. Neuron 4's external input is used to control the output pattern.

switch transistor is on at a time because the pair is driven with complementary signals from the SRAM storage circuitry. The current in each branch is switched either into a dummy cas-coded diode-connected transistor or into the nMOS output current mirror. Note that although the voltages at these nodes may differ, the linear current division is not affected. Unlike its re-

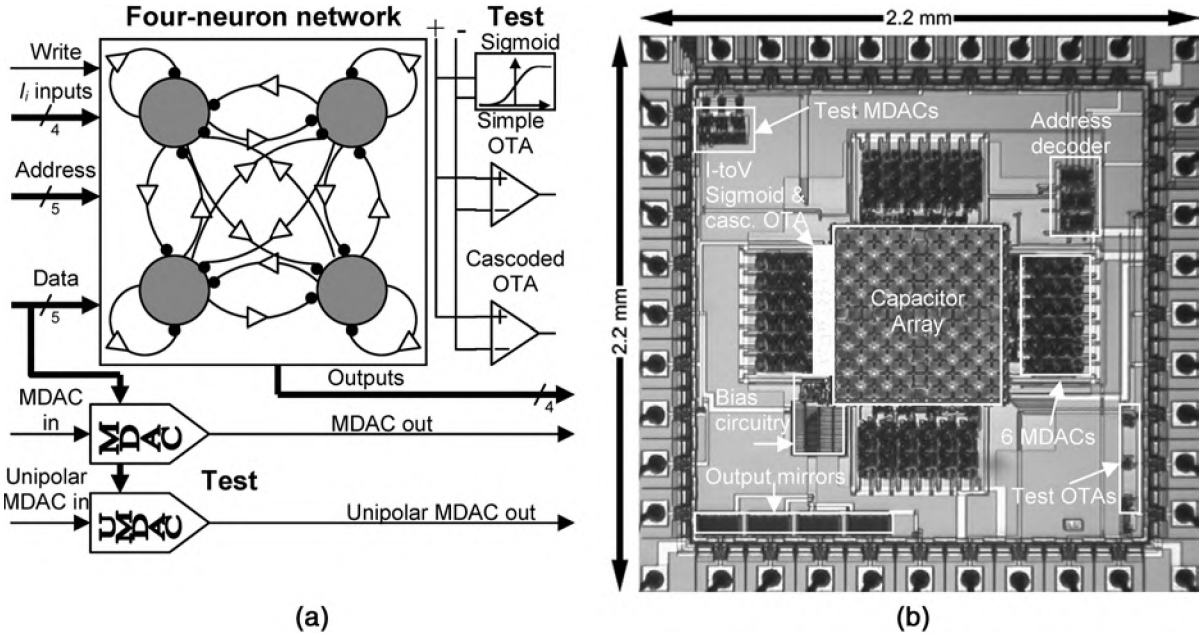

Fig. 8 (a) Chip block diagram. (b) Labeled die photograph.

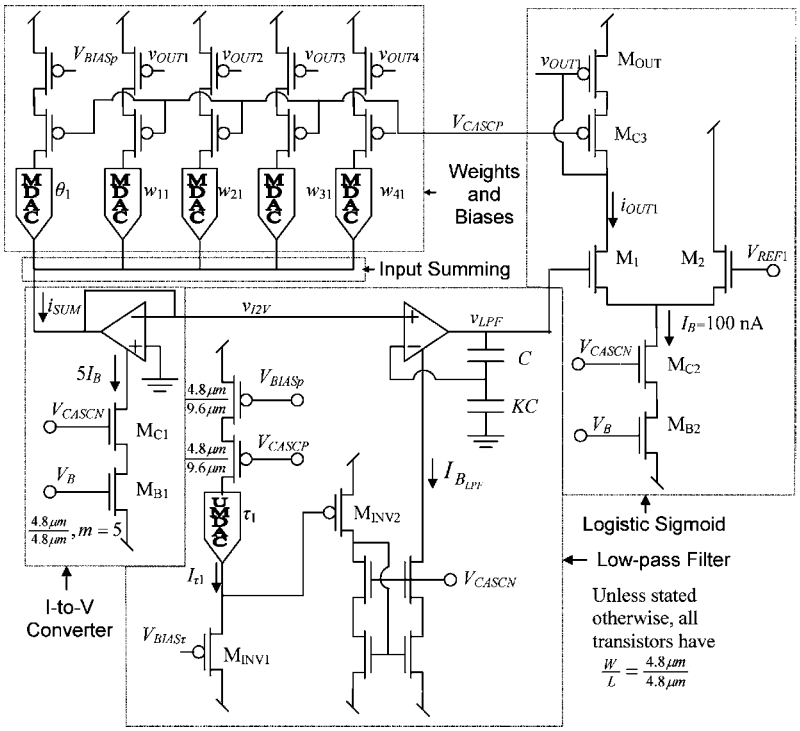

Fig. 9. Complete schematic for a single neuron used on the chip. Major functional blocks corresponding to the block diagram in Fig. 1 are highlighted.

sistive counterpart, all that is required for an MOS current divider is a large network voltage drop to keep the switch transistors M_{SWxy} in saturation. This feature of MOS pseudoresistive networks removes the need for an op-amp to provide a virtual ground (which is required in resistive implementations). In practice, finite drain resistance in saturation does impact linear current division in the MDAC splitters. To mitigate this problem, we have designed the sign bit circuitry to provide roughly equal voltages on both branches of the splitter cells. The output of the MDAC is formed by summing the binary-weighed components of the input, giving

$$I_{\mathrm{OUT}} = D \cdot I_{\mathrm{IN}}$$
$$D = (-1)^{S_{\mathrm{SIGN}}} \sum_{i=0}^{3} \frac{S_i}{2^{i+1}} \tag{6}$$

where D is the stored weight. It can be seen from (6) that the weight magnitudes are always less than one. The gain of the low-

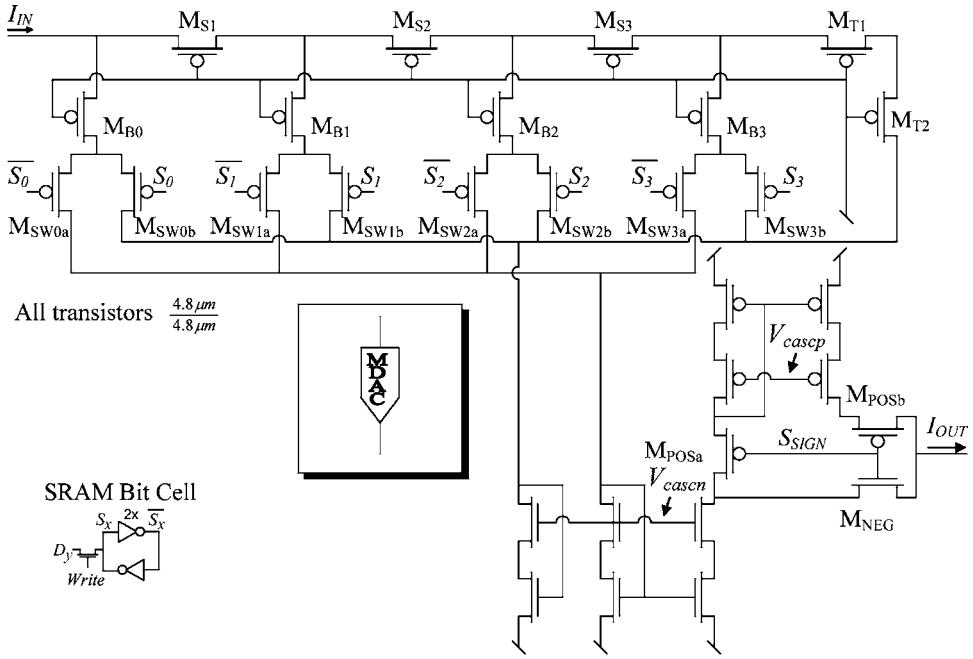

Fig. 10. Circuit diagram of the 5-bit R-$2R$ pMOS MDAC synapse and its symbol (inset).

pass filter circuitry can be used to realize weight magnitudes greater than unity.

The chip contains 24 individually programmable 5-bit MDACs. Most are used either as synapses (16) or neuron threshold biases (4), but four are used to set each neuron's time constant. For a network of size N, $N(N+2)$ synapse circuits are required for full interconnectivity. Therefore, the synapse layout area becomes a critical factor in determining the size of network that can be fabricated on a given chip. The MDAC synapse described above requires only 6900 μm^2 in a 1.5-μm process. If programmability is necessary, an additional 7300 μm^2 of area (14 200 μm^2 total) is required for the 5-bit SRAM (the SRAM is implemented using a standard five-transistor topology).

D. Input Summing and I–V Converter

After the input currents from each neuron are weighted by the synapse circuitry, they are summed together at the input to current-to-voltage converter as shown in Fig. 9. The I–V conversion is necessary because the input to the sigmoid circuitry described above is a voltage signal while the synapse circuits produce a current signal. Ideally, a resistor could be used to perform the I–V conversion. However, the subthreshold currents used in the sigmoid circuitry ($I_B = 100$ nA) would require a prohibitively large resistance to give a reasonable voltage swing.

Alternatively, an operational transconductance amplifier (OTA) connected in the unity-gain follower configuration can be used to emulate a high resistance by forcing a current into its output terminal. The negative feedback of the circuit forces the OTA to change its output current to match the input current that is forced into its output. The output voltage of the OTA I–V converter can be written as

$$v_{I2V} = \frac{1}{G_m} i_{\mathrm{SUM}};\ |i_{\mathrm{SUM}}| \leq 2.5 I_B \tag{7}$$

where $G_m = \kappa I_{\mathrm{BIAS}}/2U_T$ is the transconductance of the OTA. It can be seen from (7) that the circuit has built-in compensation for the sigmoid circuit scale factor. Unfortunately, the expression in (7) holds only for a limited range which corresponds to the 80-mV input linear range of the OTA ($|v_+ - v_-| \leq 40$ mV). Despite this limitation, an OTA-based I–V converter is used in this paper. It is possible to extend the input linear range of an OTA by using a larger bias current, but this in turn increases the transconductance of the OTA resulting in a lower overall voltage swing for the same input current swing. There are other methods to increase the linear range of the OTA such as source degeneration and gate degeneration [35], [36]. Unfortunately, both techniques force the synapse gain to become dependent on κ, and, though well matched across similarly biased devices, the exact value of κ is not well known at design time.

Instead of using special techniques to extend the linear range of the OTA, this paper uses a simple OTA (with an 80-mV input linear range) that is biased with a larger current. As mentioned above, this decreases the effective resistance of the I–V converter. However, this loss can be compensated for by using a high-gain low-pass filter in the subsequent stage.

E. Low-Pass Filter

Each neuron in a CTRNN is endowed with first-order dynamics as described by (1). In terms of circuit implementation, the neuron's dynamics is identical to that of a first-order

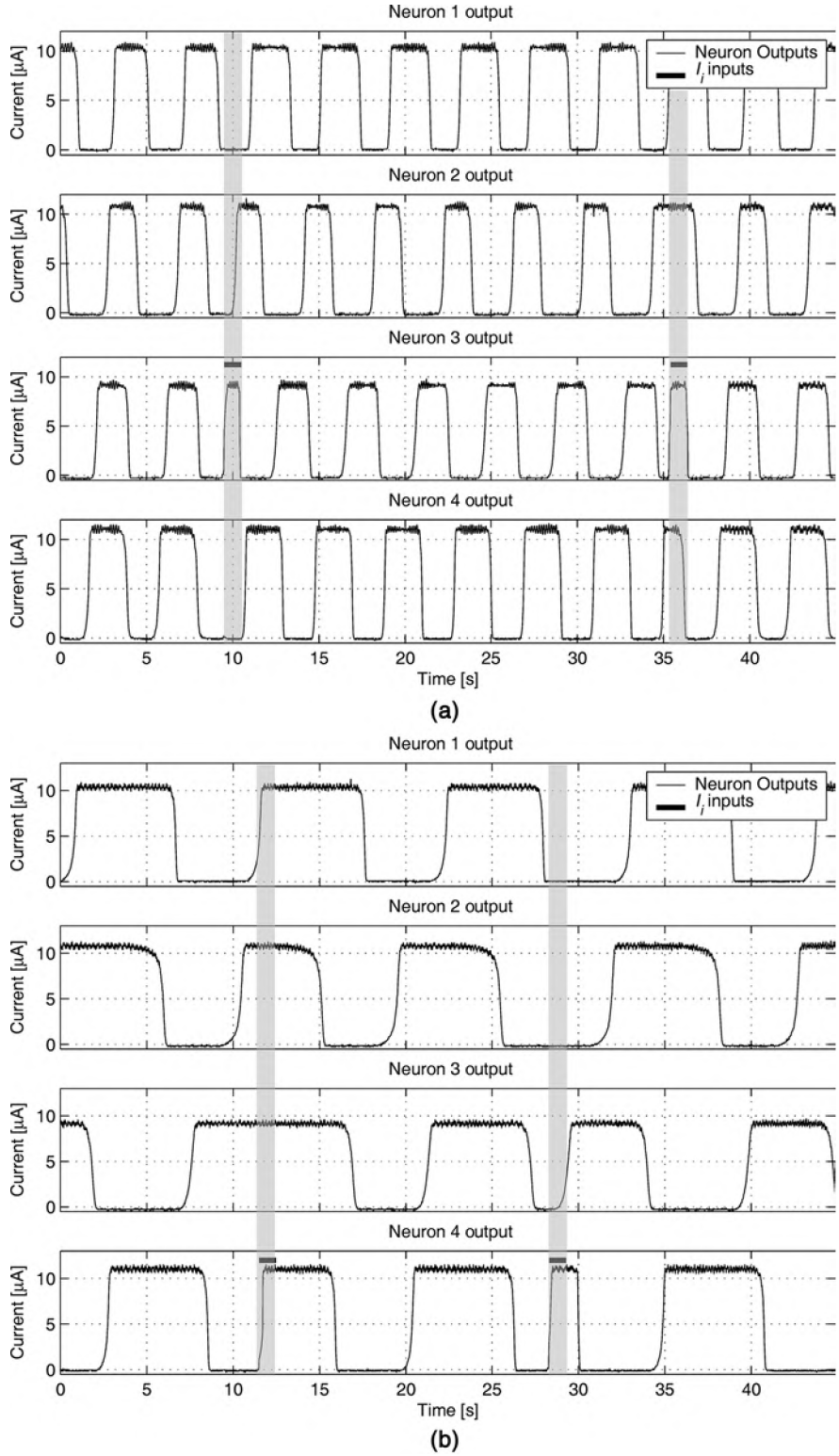

Fig. 11. Measured output patterns from the CTRNN chip. The thick bar represents the duration of an $I_i = 11$ external input used to switch between output patterns. Both figures illustrate the following sequence of events for the two example networks. 1) The network is shown initially oscillating in one of the two embedded limit cycles. 2) A transient external input is applied to one neuron's I_i input, causing the network to enter a new part of the state space where the alternate embedded limit cycle dominates. 3) The network is allowed to oscillate in the alternate limit cycle for a few cycles. 4) A transient external input is applied a second time causing the network to return to the original limit cycle. (a) Example 1 CTRNN. Neuron 3's external input is used to control the output pattern. (b) Example 2 CTRNN. Neuron 4's external input is used to control the output pattern.

low-pass filter. The simplest active low-pass filter is the integrator follower G_m-C filter realized by connecting a capacitance to the output of an OTA configured as a voltage follower. However, this circuit provides unity gain. Fortunately,

this simple G_m-C filter can be modified to provide gain if the short-circuit feedback path is replaced with a capacitive divider network as shown in Fig. 9. The circuit's response is given by

$$\frac{KC}{G_{m_{\mathrm{LPF}}}}\frac{dv_{\mathrm{LPF}}}{dt} = -v_{\mathrm{LPF}} + (1+K)v_{I2V} \tag{8}$$

which is easily recognizable as the state equation for a first-order low-pass filter. The $(1+K)$ term is the gain that is used to realize greater than unity synapse weights and to compensate for the attenuation of the I–V converter circuit. It can be seen that this circuit makes the time constant of the filter variable through the OTA bias current $I_{B_{\mathrm{LPF}}}$

$$\tau = \frac{KC}{G_{m_{\mathrm{LPF}}}} = \frac{2U_T KC}{\kappa I_{B_{\mathrm{LPF}}}} \tag{9}$$

where $C = 1$ pF. Note that the large size of unit capacitance makes the capacitor array in Fig. 8(b) artificially large. This large capacitance was selected so that $I_{B_{\mathrm{LPF}}}$ would be large enough to be measured reliably during testing.

1) Low-Pass Filter Gain: The gain of the low-pass filter must be large enough to compensate for the attenuation introduced by the I–V converter and allow for greater than unity synapse weights. The synapse circuitry implements weights normalized by 16, so the low-pass filter gain must be at least 16. The amount of additional gain required to compensate for the OTA's limited range can be computed by considering the input currents a neuron can expect to see. In a four-neuron network, each neuron receives input from five synapse circuits: one from each neuron and one bias. Since 1:1 current mirrors are used to connect the sigmoid outputs to the synapse circuits, each neuron can contribute at most a current equal to I_B, the sigmoid bias current. Therefore, the maximum possible input current for any neuron would be $\pm 5I_B$, but this event is unlikely because the weights can be positive and negative. Hence, the total input current to each neuron will seldom be greater than $2.5I_B$. As a result, the OTA is biased with $I_{\mathrm{BIAS}} = 5I_B$ to ensure good linearity, but this has the effect of attenuating the signals by a factor of 2.5. Therefore, the overall gain required is 40 ($K = 39$).

Thus far, two assumptions have been made about the low-pass filter circuit of Fig. 9. First, the node v_- must have a well-defined dc operating point for the circuit to operate properly. Second, the open-loop voltage gain of the OTA must be sufficiently high to ensure a closed-loop gain of $(1+K)$. In an earlier work, we used quasi-infinite resistors to establish a high resistance dc path to the v_- node [32]. Unfortunately, this introduced large offsets and a high-pass pole-zero pair that changed the dynamics of the filter. In this paper, the v_- node is fabricated as drawn in Fig. 9, and ultraviolet (UV) irradiation was to be used to remove any floating charge present on the node. However, a layout error (a ground shield over the capacitor array was not removed from the previous layout) prevented the UV from having the desired effect. Therefore, the offset present in each neuron is compensated for by adjusting the programmed bias value and/or adjusting the external input V_{REF} to each neuron. In future versions of the chip, Fowler–Nordheim tunneling will be used to remove unwanted floating charge.

The accuracy with which the closed-loop gain can be set to 40 depends on two things: 1) the matching between the capacitances C and KC and 2) the open-loop gain of the OTA. If proper layout techniques are used, capacitor ratios can be set with very good accuracy (to within $\pm 0.01\%$) [37]. However, even if the capacitor ratio is set perfectly, the accuracy of the closed-loop gain still depends on the OTA's open-loop gain. Therefore, a high-gain cascoded current mirror OTA is used to achieve an open-loop gain of 91.2 dB, allowing the desired closed-loop gain of 40 to be set to within 0.11%.

2) Time Constant Programmability: We have shown that the capacitance values are used to set the closed-loop gain of the low-pass filter circuit. This prevents using the capacitance to vary the time-constant of the filter, but as is indicated by (9), the time constant of the integrated low-pass filter is variable through the OTA bias current $I_{B_{\mathrm{LPF}}}$. Consequently, programmable time constants can be realized by using an MDAC to set the bias current. Unfortunately, the time constant is inversely proportional to the bias current, making it difficult to program a wide range of time constants with uniform precision. Therefore, an additional circuit is needed to compute the inverse of the MDAC current.

Fortunately, to realize long time constants, the bias current $I_{B_{\mathrm{LPF}}}$ must be a subthreshold current which allows translinear MOS circuits to be used [38]. The two transistors $\mathrm{M_{INV1}}$ and $\mathrm{M_{INV2}}$ form a translinear circuit that computes the inverse of the source current of $\mathrm{M_{INV1}}$ which is supplied by a special unipolar 5-bit MDAC. The bias voltage $V_{\mathrm{BIAS}\tau}$ is common to all neurons in the circuit, and it is used to scale simultaneously all time constants in the network by the same factor.

V. Measured Results

A. CTRNN Waveforms

The chip was tested by programming it with the parameters for the multipattern networks described in Section III. The time constants were set to approximately 0.5 s rather than the value of 1 s used in the simulations. This smaller value was chosen because the maximum recording time of the measurement equipment was only 45 s (compare this to the time scale of Fig. 7). External inputs were applied to the chip via the $V_{\mathrm{REF}i}$ pins. In Fig. 11, the input and output of each neuron are shown as the networks are switched between their stable oscillation patterns. The duration of external input to each neuron is indicated as a bar above the output waveform for each neuron. When applied, the intensity of the input signal is constant at $V_{\mathrm{REF}i} - 400$ mV, corresponding to an I_i input of $+11$ (recall that the $V_{\mathrm{REF}i}$ signals are inverted, corresponding to $-I_i$ in the CTRNN model). If the input bar is absent in Fig. 11, the corresponding I_i input is zero and the $V_{\mathrm{REF}i}$ signal is at its nominal value.

The output waveforms in Fig. 11(a) correspond to the first example CPG described in Section III. Two eight-state oscillatory patterns have been embedded into the CTRNN using the procedure described in Section III. The chip begins oscillating in the limit cycle shaded in gray in Fig. 5(a). It is switched between

the two patterns by applying a properly timed pulse to neuron 3's input. As the CTRNN passes through state 0000, neuron 3's input is activated to force the network into state 0010, which resides in the second limit cycle (shown in white) in Fig. 5(a). The CTRNN remains in this limit cycle until neuron 3's input is again pulsed, this time as the CTRNN passes through state 1101, forcing the next state to be 1111, which is in the first (gray) limit cycle. Note that the same input is used to switch between the patterns, with the timing of the input determining which transition will occur. Similarly, Fig. 11(b) shows the output waveforms for the second CPG described in Section III. In this network, the external input to neuron 4 is used to switch the network between the two embedded patterns.

It is clear from Fig. 11 that the CTRNN chip behaves as expected, demonstrating reconfigurability in multipattern CPGs. With the exception of a bit of noise and some small variation in the maximum neuron output levels, the measured waveforms in Fig. 11 agree well with the simulated waveforms shown in Fig. 7. The output level variation is due to mismatch in the output current mirrors for each neuron. This variation arises from the small size of the transistors that make up the neuron circuitry. If desired, this variation could be reduced by employing larger transistors, though this would increase the layout area of the network.

B. Power Consumption

The CTRNN chip operates on ± 2.5-V supplies and dissipates a total of 143.3 μW. Of this 143.3 μW, 56.8 μW (40%) is used by the output drivers, 53.5 μW (37%) is used by the master bias circuitry, and 33 μW (23%) is used by the CTRNN. This translates into 8.25 μW/neuron in the core. An additional 14.2μW is required for each neuron output that must be driven off-chip.

VI. Conclusion

In this paper, we have described a method of designing multipattern CPGs using CTRNNs. Our method allows multiple patterns to be embedded into a CTRNN with fixed connectivity. We have demonstrated that it is possible to switch between these multiple output patterns by using properly timed transient inputs to specific neurons. We also presented a fully integrated analog implementation of a four-neuron CTRNN. The chip has a digitally programmable weight matrix and also offers long time constants that are digitally programmable over a wide range. We have used our chip to demonstrate multipattern CPGs developed using our new design method. The measured results from our chip agree well with simulation results, making it possible to develop multipattern CPGs using offline simulations without being concerned with implementation details.

Future work in this area will be focused in three areas.

1) Larger networks: four-neuron networks are suitable for controlling a single leg [39], but larger networks are required for a complete locomotor controller.
2) Networks in which the self-weight is a freely-chosen parameter.
3) Embedding more diverse transitions.

In this paper, we restricted our attention to patterns in which only one neuron at a time was allowed to change state.

The implementation is scalable to larger networks so long as the maximum weight magnitude decreases as the number of neurons increases (i.e., the maximum currents in the I–V converters must remain reasonable). CTRNNs demonstrate the richest dynamical behavior when the input to each neuron remains close to the center of the sigmoidal activation function [15]. Therefore, this condition is likely to be met for networks with rich dynamics. This leaves layout area and interconnect the limiting factors in scaling the implementation to large numbers of neurons. Using this 1.5-μm technology, it is feasible to fit 10–15 neurons on a 2.2 by 2.2 mm die.

References

[1] S. Grillner, "The motor infrastructure: from ion channels to neuronal networks," *Nature Rev. Neurosci.*, vol. 4, pp. 573–586, Jul. 2003.

[2] M. Belanger, T. Drew, J. Provencher, and S. Rossignol, "A comparison of treadmill locomotion in adult cats before and after spinal transection," *J. Neurophysiol.*, vol. 76, pp. 471–491, 1996.

[3] S. Grillner and P. Zangger, "The effect of dorsal root transection on the efferent motor pattern in the cat's hindlimb during locomotion," *Acta Phys. Scand.*, vol. 120, pp. 393–405, 1984.

[4] K. Pearson and J. Gordon, "Locomotion," in *Principles of Neural Science*, E. R. Kandel, J. H. Schwartz, and T. M. Jessell, Eds. New York: McGraw-Hill, 2001.

[5] C. C. Canavier, D. A. Baxter, J. W. Clark, and J. H. Byrne, "Nonlinear dynamics in a model neuron provide a novel mechanism for transient synaptic inputs to produce long-term alterations of postsynaptic activity," *J. Neurophysiol.*, vol. 69, pp. 2252–2257, 1993.

[6] C. Luo, J. W. Clark, Jr., C. C. Canavier, D. A. Baxter, and J. H. Byrne, "Multimodal behavior in a four neuron ring circuit: mode switching," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 2, pp. 205–218, Feb. 2004.

[7] D. Kleinfeld and H. Sompolinsky, "Associative network models for central pattern generators," in *Methods in Neuronal Modeling*, C. Koch and I. Segev, Eds. Cambridge, MA: MIT Press, 1989.

[8] J. C. Gallagher, R. D. Beer, K. S. Espenschied, and R. D. Quinn, "Application of evolved locomotion controllers to a hexpod robot," *Robot. Auton. Syst.*, vol. 19, pp. 95–103, 1996.

[9] R. D. Beer, R. D. Quinn, H. J. Chiel, and R. E. Ritzmann, "Biologically inspired approaches to robotics," *Commun. ACM*, vol. 40, pp. 31–38, 1997.

[10] G. N. Patel, J. H. Holleman, and S. P. DeWeerth, "Analog VLSI model of intersegmental coordination with nearest-neighbor coupling," *Adv. Neural Inf. Process. Syst.*, vol. 11, pp. 719–725, 1998.

[11] S. Still, B. Schölkopf, K. Hepp, and R. Douglas, "Four-legged walking gait control using a neuromorphic chip interfaced to a support vector learning algorithm," *Adv. Neural Inf. Process. Syst.*, vol. 13, 2000.

[12] M. A. Lewis, R. Etienne-Cummings, M. J. Hartmann, A. H. Cohen, and Z. R. Xu, "An in silico central pattern generator: silicon oscillator, coupling, entrainment, and physical computation," *Biol. Cybern.*, vol. 88, pp. 137–151, 2003.

[13] F. Tenore, R. Etienne-Cummings, and M. A. Lewis, "Entrainment of silicon central pattern generators for legged locomotory control," *Adv. Neural Inf. Process. Syst.*, vol. 16, 2003.

[14] ——, "A programmable array of silicon neurons for the control of legged locomotion," in *Proc. IEEE Int. Symp. Circuits Systems (ISCAS)*, 2004, vol. V, pp. 349–352.

[15] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adapt. Behav.*, vol. 3, pp. 471–511, 1995.

[16] J. J. Hopfield, "Neurons with graded response properties have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sciences*, vol. 81, pp. 3088–3092, 1984.

[17] K. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Netw.*, vol. 6, pp. 801–806, 1993.

[18] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[19] B. E. Boser, E. Sackinger, J. Bromley, Y. L. Cun, and L. D. Jackel, "An analog neural network processor with programmable topology," *IEEE J. Solid-State Circuits*, vol. 26, no. 12, pp. 2017–2025, Dec. 1991.

[20] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, "A reconfigurable VLSI neural network," *IEEE J. Solid-State Circuits*, vol. 27, no. 1, pp. 67–81, Jan. 1992.

[21] A. J. Montalvo, R. S. Gyurcsik, and J. J. Paulos, "Toward a general-purpose analog VLSI neural network with on-chip learning," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 413–423, Mar. 1997.
[22] G. Cauwenburghs, "An analog VLSI recurrent neural network learning a continuous-time trajectory," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 346–361, Mar. 1997.
[23] B. W. Lee, B. J. Sheu, and H. Yang, "Analog floating-gate synapses for general-purpose VLSI neural computation," *IEEE Trans. Circuits Syst.*, vol. 38, no. 6, pp. 654–658, Jun. 1991.
[24] S. M. Gowda, B. J. Sheu, J. Choi, C. G. Hwang, and J. S. Cable, "Design and characterization analog VLSI neural network modules," *IEEE J. Solid-State Circuits*, vol. 28, no. 3, pp. 301–313, Mar. 1993.
[25] P. Hasler, "Continuous-time feedback in floating-gate CMOS circuits," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 1, pp. 56–64, Jan. 2001.
[26] C. Diorio, "A p-channel MOS synapse transistor with self-convergent memory writes," *IEEE Trans. Electron Devices*, vol. 47, no. 2, pp. 464–472, Feb. 2000.
[27] J. Dugger and P. Hasler, "A continuously-adapting analog node using floating-gate synapses," in *Proc. 43rd IEEE Midwest Symp. Circuits Systems*, 2000, pp. 1058–1061.
[28] A. Moopenn, T. Duong, and A. P. Thakoor, "Digital-analog hybrid synapse chips for electronic neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 2, pp. 769–776, 1990.
[29] H. Djahanshahi, M. Ahmadi, G. A. Jullien, and W. C. Miller, "Design and VLSI implementation of a unified neuron-synapse architecture," in *Proc. 6th Great Lakes Symp. VLSI*, 1996, pp. 228–233.
[30] B. Linares-Barranco, T. Serrano-Gotarredona, and R. Serrano-Gotarredona, "Compact low-power calibration mini-DACs for neural arrays with programmable weights," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1207–1216, Sep. 2003.
[31] R. J. Kier, R. R. Harrison, and R. D. Beer, "An MDAC synapse for analog neural networks," in *Proc. IEEE Int. Symp. Circuits Systems (ISCAS)*, 2004, vol. V, pp. 752–755.
[32] R. J. Kier, "Design of pattern generators in analog integrated circuits," master's thesis, Univ. Utah, , Aug. 2004.
[33] K. Bult and G. J. G. M. Geelen, "An inherently linear and compact MOST-only current division technique," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1730–1735, Dec. 1992.
[34] E. A. Vittoz and X. Arreguit, "Linear networks based on transistors," *Electron. Lett.*, vol. 20, pp. 297–299, 1993.
[35] L. Watts, D. A. Kerns, R. F. Lyon, and C. A. Mead, "Improved implementations of the silicon cochlea," *IEEE J. Solid-State Circuits*, vol. 27, no. 5, pp. 692–700, May 1992.
[36] R. Sarpeshkar, "Efficient precise computation with noisy components extrapolating from an electronic cochlea to the brain," Ph.D. dissertation, California Inst. Tech., Pasadena, CA, 1997.
[37] A. Hastings, *The Art of Analog Layout*. Englewood Cliffs, NJ: Prentice-Hall, 2001.
[38] A. G. Andreou and K. A. Boahen, "Translinear circuits in subthreshold MOS," *Analog Integr. Circuits Signal Process.*, vol. 9, pp. 141–166, 1996.
[39] J. C. Ames, "Design methods for pattern generation circuits," M.S. thesis, Case Western Reserve Univ., Cleveland, OH, 2003.

Ryan J. Kier (S'03) received the B.S. and M.S. degrees in electrical engineering from the University of Utah, Salt Lake City, in 2004, where he is currently pursuing the Ph.D. degree.

His thesis work involved the implementation of neurally inspired circuits in analog VLSI. His research interests include low-power VCO design and integrated inductor optimization.

Jeffrey C. Ames, photograph and biography not available at the time of publication.

Randall D. Beer received the B.S. degree in computer engineering and the M.S. and Ph.D. degrees in computer science from Case Western Reserve University, Cleveland, OH, in 1985, 1985, and 1989, respectively.

He is currently a Professor in the Electrical Engineering and Computer Science Department, Case Western Reserve University, with joint appointments in the Department of Biology and the Department of Cognitive Science. He spent the 1995–1996 academic year on sabbatical leave at the Santa Fe Institute, and served as an External Faculty Member there from 1997 to 2003. In July 2006, he will join the Cognitive Science Program and the Department of Computer Science at Indiana University. His primary research interest is in understanding how coordinated behavior arises from the dynamical interaction of an animal's nervous system, its body, and its environment. He has worked on the evolution and analysis of dynamical "nervous systems" for model agents, neuromechanical modeling of animals, biologically inspired robotics, and dynamical systems approaches to behavior and cognition.

Reid R. Harrison (S'98–M'00) received the B.S. degree in electrical engineering from the University of Florida, Gainesville, in 1994 and the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 2000.

He joined the University of Utah, Salt Lake City, in 2000, where he is now an Assistant Professor of electrical and computer engineering and an Adjunct Assistant Professor of bioengineering. He has more than 30 refereed publications since 1999 in the fields of low-power analog and mixed-signal CMOS circuit design, integrated electronics for neural interfaces and other biomedical devices, and hardware for biologically inspired computational systems.