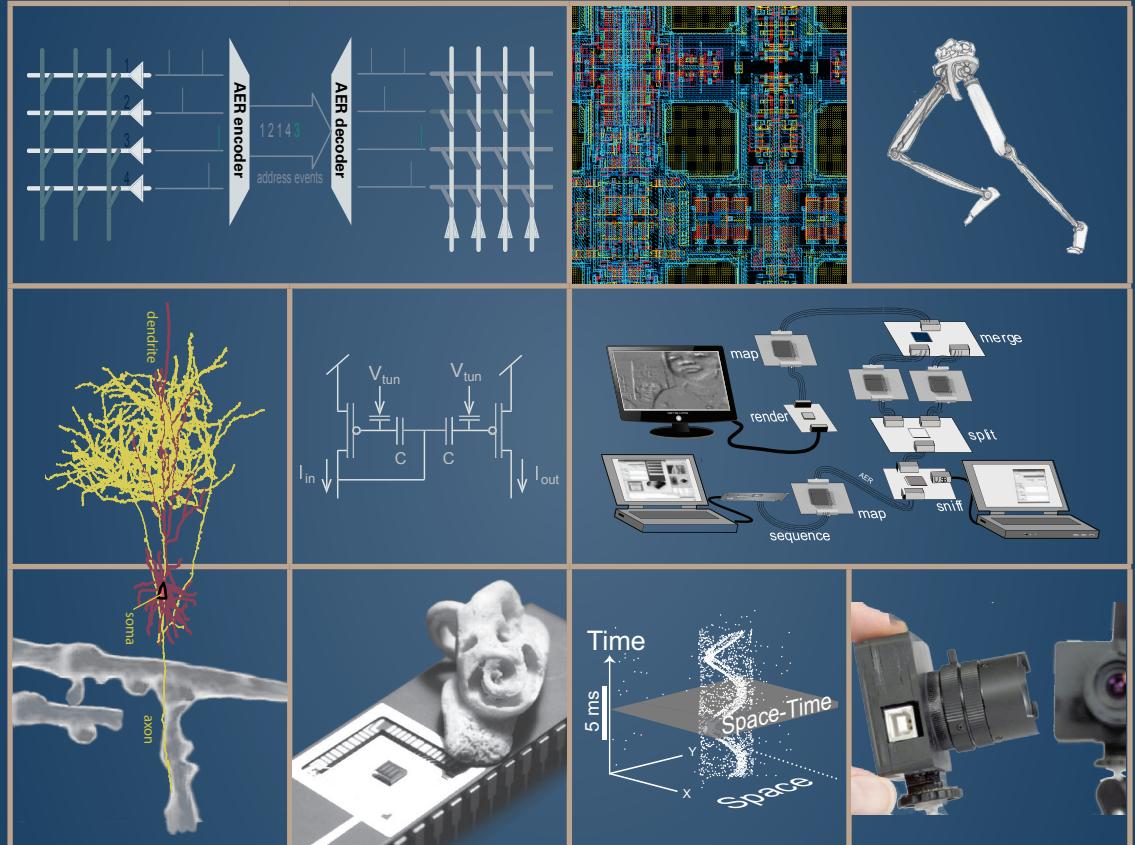


EVENT-BASED NEUROMORPHIC SYSTEMS



Edited by

Shih-Chii Liu
Tobi Delbrück
Giacomo Indiveri
Adrian Whatley
Rodney Douglas

WILEY

EVENT-BASED NEUROMORPHIC SYSTEMS

EVENT-BASED NEUROMORPHIC SYSTEMS

Edited by

Shih-Chii Liu
Tobi Delbrück
Giacomo Indiveri
Adrian Whatley
Rodney Douglas

*University of Zürich and ETH Zürich
Switzerland*

WILEY

This edition first published 2015
© 2015 John Wiley & Sons, Ltd

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. It is sold on the understanding that the publisher is not engaged in rendering professional services and neither the publisher nor the author shall be liable for damages arising herefrom. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data applied for.

ISBN: 9780470018491

Set in 10/12pt Times by Aptara Inc., New Delhi, India

*This book is dedicated to the memories of Misha Mahowald,
Jörg Kramer, and Paul Mueller.*

Contents

List of Contributors	xv
Foreword	xvii
Acknowledgments	xix
List of Abbreviations and Acronyms	xxi
1 Introduction	1
1.1 Origins and Historical Context	3
1.2 Building Useful Neuromorphic Systems	5
References	5
Part I UNDERSTANDING NEUROMORPHIC SYSTEMS	7
2 Communication	9
2.1 Introduction	9
2.2 Address-Event Representation	12
2.2.1 AER Encoders	13
2.2.2 Arbitration Mechanisms	13
2.2.3 Encoding Mechanisms	17
2.2.4 Multiple AER Endpoints	19
2.2.5 Address Mapping	19
2.2.6 Routing	19
2.3 Considerations for AER Link Design	20
2.3.1 Trade-off: Dynamic or Static Allocation	21
2.3.2 Trade-off: Arbitered Access or Collisions?	23
2.3.3 Trade-off: Queueing versus Dropping Spikes	24
2.3.4 Predicting Throughput Requirements	25
2.3.5 Design Trade-offs	27
2.4 The Evolution of AER Links	28
2.4.1 Single Sender, Single Receiver	28
2.4.2 Multiple Senders, Multiple Receivers	30
2.4.3 Parallel Signal Protocol	31

2.4.4	Word-Serial Addressing	32
2.4.5	Serial Differential Signaling	33
2.5	Discussion	34
	References	35
3	Silicon Retinas	37
3.1	Introduction	37
3.2	Biological Retinas	38
3.3	Silicon Retinas with Serial Analog Output	39
3.4	Asynchronous Event-Based Pixel Output Versus Synchronous Frames	40
3.5	AER Retinas	40
3.5.1	Dynamic Vision Sensor	41
3.5.2	Asynchronous Time-Based Image Sensor	46
3.5.3	Asynchronous Parvo-Magno Retina Model	46
3.5.4	Event-Based Intensity-Coding Imagers (Octopus and TTFS)	48
3.5.5	Spatial Contrast and Orientation Vision Sensor (VISe)	50
3.6	Silicon Retina Pixels	54
3.6.1	DVS Pixel	54
3.6.2	ATIS Pixel	56
3.6.3	VISe Pixel	58
3.6.4	Octopus Pixel	59
3.7	New Specifications for Silicon Retinas	60
3.7.1	DVS Response Uniformity	60
3.7.2	DVS Background Activity	62
3.7.3	DVS Dynamic Range	62
3.7.4	DVS Latency and Jitter	63
3.8	Discussion	64
	References	67
4	Silicon Cochleas	71
4.1	Introduction	72
4.2	Cochlea Architectures	75
4.2.1	Cascaded 1D	76
4.2.2	Basic 1D Silicon Cochlea	77
4.2.3	2D Architecture	78
4.2.4	The Resistive (Conductive) Network	79
4.2.5	The BM Resonators	80
4.2.6	The 2D Silicon Cochlea Model	80
4.2.7	Adding the Active Nonlinear Behavior of the OHCs	82
4.3	Spike-Based Cochleas	83
4.3.1	Q-control of AEREAR2 Filters	85
4.3.2	Applications: Spike-Based Auditory Processing	86
4.4	Tree Diagram	87
4.5	Discussion	87
	References	89

5	Locomotion Motor Control	91
5.1	Introduction	92
5.1.1	Determining Functional Biological Elements	92
5.1.2	Rhythmic Motor Patterns	93
5.2	Modeling Neural Circuits in Locomotor Control	95
5.2.1	Describing Locomotor Behavior	96
5.2.2	Fictive Analysis	97
5.2.3	Connection Models	99
5.2.4	Basic CPG Construction	100
5.2.5	Neuromorphic Architectures	102
5.3	Neuromorphic CPGs at Work	108
5.3.1	A Neuroprosthesis: Control of Locomotion <i>in Vivo</i>	109
5.3.2	Walking Robots	111
5.3.3	Modeling Intersegmental Coordination	112
5.4	Discussion	113
	References	115
6	Learning in Neuromorphic Systems	119
6.1	Introduction: Synaptic Connections, Memory, and Learning	120
6.2	Retaining Memories in Neuromorphic Hardware	121
6.2.1	The Problem of Memory Maintenance: Intuition	121
6.2.2	The Problem of Memory Maintenance: Quantitative Analysis	122
6.2.3	Solving the Problem of Memory Maintenance	124
6.3	Storing Memories in Neuromorphic Hardware	128
6.3.1	Synaptic Models for Learning	128
6.3.2	Implementing a Synaptic Model in Neuromorphic Hardware	132
6.4	Toward Associative Memories in Neuromorphic Hardware	136
6.4.1	Memory Retrieval in Attractor Neural Networks	137
6.4.2	Issues	142
6.5	Attractor States in a Neuromorphic Chip	143
6.5.1	Memory Retrieval	143
6.5.2	Learning Visual Stimuli in Real Time	145
6.6	Discussion	148
	References	149
Part II	BUILDING NEUROMORPHIC SYSTEMS	153
7	Silicon Neurons	155
7.1	Introduction	156
7.2	Silicon Neuron Circuit Blocks	158
7.2.1	Conductance Dynamics	158
7.2.2	Spike-Event Generation	159
7.2.3	Spiking Thresholds and Refractory Periods	161
7.2.4	Spike-Frequency Adaptation and Adaptive Thresholds	162

7.2.5	Axons and Dendritic Trees	164
7.2.6	Additional Useful Building Blocks	165
7.3	Silicon Neuron Implementations	166
7.3.1	Subthreshold Biophysically Realistic Models	166
7.3.2	Compact I&F Circuits for Event-Based Systems	169
7.3.3	Generalized I&F Neuron Circuits	170
7.3.4	Above Threshold, Accelerated-Time, and Switched-Capacitor Designs	174
7.4	Discussion	176
	References	180
8	Silicon Synapses	185
8.1	Introduction	186
8.2	Silicon Synapse Implementations	188
8.2.1	Non Conductance-Based Circuits	188
8.2.2	Conductance-Based Circuits	198
8.2.3	NMDA Synapse	200
8.3	Dynamic Plastic Synapses	201
8.3.1	Short-Term Plasticity	201
8.3.2	Long-Term Plasticity	203
8.4	Discussion	213
	References	215
9	Silicon Cochlea Building Blocks	219
9.1	Introduction	219
9.2	Voltage-Domain Second-Order Filter	220
9.2.1	Transconductance Amplifier	220
9.2.2	Second-Order Low-Pass Filter	222
9.2.3	Stability of the Filter	223
9.2.4	Stabilised Second-Order Low-Pass Filter	225
9.2.5	Differentiation	225
9.3	Current-Domain Second-Order Filter	227
9.3.1	The Translinear Loop	227
9.3.2	Second-Order Tau Cell Log-Domain Filter	229
9.4	Exponential Bias Generation	230
9.5	The Inner Hair Cell Model	233
9.6	Discussion	234
	References	234
10	Programmable and Configurable Analog Neuromorphic ICs	237
10.1	Introduction	238
10.2	Floating-Gate Circuit Basics	238
10.3	Floating-Gate Circuits Enabling Capacitive Circuits	238
10.4	Modifying Floating-Gate Charge	242
10.4.1	Electron Tunneling	242
10.4.2	pFET Hot-Electron Injection	242

10.5	Accurate Programming of Programmable Analog Devices	244
10.6	Scaling of Programmable Analog Approaches	246
10.7	Low-Power Analog Signal Processing	247
10.8	Low-Power Comparisons to Digital Approaches: Analog Computing in Memory	249
10.9	Analog Programming at Digital Complexity: Large-Scale Field Programmable Analog Arrays	251
10.10	Applications of Complex Analog Signal Processing	253
10.10.1	Analog Transform Imagers	253
10.10.2	Adaptive Filters and Classifiers	253
10.11	Discussion	256
	References	257
11	Bias Generator Circuits	261
11.1	Introduction	261
11.2	Bias Generator Circuits	263
11.2.1	Bootstrapped Current Mirror Master Bias Current Reference	263
11.2.2	Master Bias Power Supply Rejection Ratio (PSRR)	265
11.2.3	Stability of the Master Bias	265
11.2.4	Master Bias Startup and Power Control	266
11.2.5	Current Splitters: Obtaining a Digitally Controlled Fraction of the Master Current	267
11.2.6	Achieving Fine Monotonic Resolution of Bias Currents	271
11.2.7	Using Coarse–Fine Range Selection	273
11.2.8	Shifted-Source Biasing for Small Currents	274
11.2.9	Buffering and Bypass Decoupling of Individual Biases	275
11.2.10	A General Purpose Bias Buffer Circuit	278
11.2.11	Protecting Bias Splitter Currents from Parasitic Photocurrents	279
11.3	Overall Bias Generator Architecture Including External Controller	279
11.4	Typical Characteristics	280
11.5	Design Kits	281
11.6	Discussion	282
	References	282
12	On-Chip AER Communication Circuits	285
12.1	Introduction	286
12.1.1	Communication Cycle	286
12.1.2	Speedup in Communication	287
12.2	AER Transmitter Blocks	289
12.2.1	AER Circuits within a Pixel	289
12.2.2	Arbiter	290
12.2.3	Other AER Blocks	295
12.2.4	Combined Operation	297
12.3	AER Receiver Blocks	298
12.3.1	Chip-Level Handshaking Block	298
12.3.2	Decoder	299

12.3.3	Handshaking Circuits in Receiver Pixel	300
12.3.4	Pulse Extender Circuits	301
12.3.5	Receiver Array Peripheral Handshaking Circuits	301
12.4	Discussion	302
	References	303
13	Hardware Infrastructure	305
13.1	Introduction	306
13.1.1	Monitoring AER Events	307
13.1.2	Sequencing AER Events	311
13.1.3	Mapping AER Events	313
13.2	Hardware Infrastructure Boards for Small Systems	316
13.2.1	Silicon Cortex	316
13.2.2	Centralized Communication	317
13.2.3	Composable Architecture Solution	318
13.2.4	Daisy-Chain Architecture	324
13.2.5	Interfacing Boards using Serial AER	324
13.2.6	Reconfigurable Mesh-Grid Architecture	328
13.3	Medium-Scale Multichip Systems	329
13.3.1	Octopus Retina + IFAT	329
13.3.2	Multichip Orientation System	332
13.3.3	CAVIAR	335
13.4	FPGAs	340
13.5	Discussion	342
	References	345
14	Software Infrastructure	349
14.1	Introduction	349
14.1.1	Importance of Cross-Community Commonality	350
14.2	Chip and System Description Software	350
14.2.1	Extensible Markup Language	351
14.2.2	NeuroML	351
14.3	Configuration Software	352
14.4	Address Event Stream Handling Software	352
14.4.1	Field-Programmable Gate Arrays	353
14.4.2	Structure of AE Stream Handling Software	353
14.4.3	Bandwidth and Latency	353
14.4.4	Optimization	354
14.4.5	Application Programming Interface	355
14.4.6	Network Transport of AE Streams	355
14.5	Mapping Software	356
14.6	Software Examples	357
14.6.1	ChipDatabase – A System for Tuning Neuromorphic aVLSI Chips	357
14.6.2	Spike Toolbox	359
14.6.3	jAER	359
14.6.4	Python and PyNN	360

14.7	Discussion	363
	References	363
15	Algorithmic Processing of Event Streams	365
15.1	Introduction	365
15.2	Requirements for Software Infrastructure	367
15.2.1	Processing Latency	369
15.3	Embedded Implementations	369
15.4	Examples of Algorithms	370
15.4.1	Noise Reduction Filters	370
15.4.2	Time-Stamp Maps and Subsampling by Bit-Shifting Addresses	372
15.4.3	Event Labelers as Low-Level Feature Detectors	372
15.4.4	Visual Trackers	374
15.4.5	Event-Based Audio Processing	378
15.5	Discussion	379
	References	379
16	Towards Large-Scale Neuromorphic Systems	381
16.1	Introduction	381
16.2	Large-Scale System Examples	382
16.2.1	Spiking Neural Network Architecture	382
16.2.2	Hierarchical AER	384
16.2.3	Neurogrid	386
16.2.4	High Input Count Analog Neural Network System	388
16.3	Discussion	390
	References	391
17	The Brain as Potential Technology	393
17.1	Introduction	393
17.2	The Nature of Neuronal Computation: Principles of Brain Technology	395
17.3	Approaches to Understanding Brains	396
17.4	Some Principles of Brain Construction and Function	398
17.5	An Example Model of Neural Circuit Processing	400
17.6	Toward Neuromorphic Cognition	402
	References	404
Index		407

List of Contributors

Editors:

Shih-Chii Liu

Tobi Delbrück

Giacomo Indiveri

Adrian Whatley

Rodney Douglas

Institute of Neuroinformatics

University of Zürich and ETH Zürich

Zürich, Switzerland

Contributors:

Corey Ashby

Johns Hopkins University

Baltimore, MD, USA

Ralph Etienne-Cummings

Johns Hopkins University

Baltimore, MD, USA

Paolo Del Giudice

Department of Technologies and Health

Istituto Superiore di Sanità

Rome, Italy

Stefano Fusi

Center for Theoretical Neuroscience

Columbia University

New York, NY, USA

Tara Hamilton

The MARCS Institute

University of Western Sydney

Sydney, Australia

Jennifer Hasler
Georgia Tech
Atlanta, GA, USA

Alejandro Linares-Barranco
Universidad de Sevilla
Sevilla, Spain

Bernabe Linares-Barranco
National Microelectronics Center
(IMSE-CNM-CSIC)
Sevilla, Spain

Rajit Manohar
Cornell Tech
New York, NY, USA

Kevan Martin
Institute of Neuroinformatics
University of Zürich and ETH Zürich
Zürich, Switzerland

André van Schaik
The MARCS Institute
University of Western Sydney
Sydney, Australia

Jacob Vogelstein
Johns Hopkins University
Baltimore, MD, USA

Contributors by Chapter:

Chapter 1: Tobi Delbrück

Chapter 2: Rajit Manohar, Adrian Whatley, Shih-Chii Liu

Chapter 3: Tobi Delbrück, Bernabe Linares-Barranco

Chapter 4: André van Schaik, Tara Hamilton, Shih-Chii Liu

Chapter 5: Corey Ashby, Ralph Etienne-Cummings, Jacob Vogelstein

Chapter 6: Stefano Fusi, Paolo del Giudice

Chapter 7: Giacomo Indiveri

Chapter 8: Shih-Chii Liu, Giacomo Indiveri

Chapter 9: André van Schaik, Tara Hamilton

Chapter 10: Jennifer Hasler

Chapter 11: Tobi Delbrück, Bernabe Linares-Barranco

Chapter 12: Shih-Chii Liu

Chapter 13: Adrian Whatley, Alejandro Linares-Barranco, Shih-Chii Liu

Chapter 14: Adrian Whatley

Chapter 15: Tobi Delbrück

Chapter 16: Rajit Manohar

Chapter 17: Rodney Douglas, Kevan Martin

Foreword

The motivation for building neuromorphic systems has roots in engineering and neuroscience. On the engineering side, inspiration from how the brain solves complex problems has led to new computing algorithms; however, the goal of reverse engineering the brain is a difficult one because the brain is based on a biological technology that was evolved and not designed by human engineers. On the neuroscience side, the goal is to understand brain function, which is still at an early stage owing the extremely heterogeneous and compact nature of neural circuits. Neuromorphic systems are a bridge between these two ambitious enterprises. The lessons learned from building devices based on neural architectures are providing new engineering capabilities and new biological insights.

Building neuromorphic VLSI chips and perfecting asynchronous event-based communication between them has required a generation of talented engineering scientists. These people were inspired by Carver Mead and his 1989 landmark book on *Analog VLSI and Neural Systems*. I was a Wiersma Visiting Professor of Neurobiology at the California Institute of Technology in 1987 and attended ‘Carverland’ group meetings. Neuromorphic engineering was still in its infancy, but the strengths and weaknesses of the technology were already apparent. The promise of a new massively parallel, low-power, and inexpensive computing architecture was balanced by the technical challenges of working with the transistor mismatch and noise that plagued analog VLSI chips. The brain was an existence proof that these problems could be overcome, but it took much longer time than expected to find the practical solutions which are discussed in detail in *Event-Based Neuromorphic Systems*.

At about the same time that neuromorphic systems were introduced, the neural network revolution was getting underway based on simulations of simplified models of neurons. The two-volume 1986 book on *Parallel Distributed Processing*¹ had chapters on two new learning algorithms for multilayer network models, the backpropagation of errors and the Boltzmann machine. These networks were learned from examples, in contrast to engineered systems that were handcrafted. The increase in overall computing power by a factor of a million over the last 25 years and the large sizes of data sets now available on the Internet have made deep learning in hierarchies of simple model neurons both powerful and practical, at least when power is unlimited. The Neural Information Processing Systems (NIPS) meeting in 2013 had 2000 attendees and the applications of machine learning ranged from vision systems to advertisement recommender systems.

¹ Rumelhart DE and McClelland JL. 1986. *Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Systems neuroscience has made progress one neuron at a time since single cortical neurons were first recorded in 1959. In the last 10 years, new optical techniques have made it possible to record simultaneously from hundreds of neurons and allowed researchers to both selectively stimulate and suppress the spikes in subtypes of neurons. Analytic tools are being developed to explore the statistical structure of brain activity in large populations of neurons, and reconstruction of the connectivity of the brain from electron micrographs, aided by machine learning, is producing intricate wiring diagrams. The goal, which is far from yet realized, is to use these new tools to understand how activity in neural circuits generates behavior.

The next 25 years could be a golden period as these three interacting areas of research in neuromorphic electronic systems, artificial neural networks, and systems neuroscience reach maturity and fulfill their potential. Each has an important role to play in achieving the ultimate goal, to understand how the properties of neurons and communications systems in brains give rise to our ability to see, hear, plan, decide, and take action. Reaching this goal would give us a better understanding of who we are and create a new neurotechnology sector of the economy with far-reaching impact on our everyday lives. *Event-Based Neuromorphic Systems* is an essential resource for neuromorphic electrical engineers pursuing this goal.

Terrence Sejnowski

La Jolla, California

December 15, 2013

Acknowledgments

This book would never taken form without the consistent support of the US National Science Foundation (NSF) in funding the Telluride Neuromorphic Cognition Engineering Workshop and the help of the EU FET (Future and Emerging Technologies) program in supporting the European CapoCaccia Cognitive Neuromorphic Engineering Workshop. These unique, hands-on workshops were the places where many of the ideas developed in this book were first discussed and prototyped. For many neuromorphic engineers, these workshops are a highlight of the year and a kind of working holiday time that other venues, such as IEEE conferences, cannot replace.

The editors and contributors to *Event-Based Neuromorphic Systems* acknowledge the following people for reading and commenting on various chapters in the book: Luca Longinotti, Bjorn Beyer, Michael Pfeiffer, Josep Maria Margarit Taulé, Diederik Moeyns, Bradley Minch, Sim Bamford, Min-Hao Yang, and Christoph Posch. They acknowledge Srinjoy Mitra for his contribution to the on-chip AER circuit chapter, Philipp Häfliger for his contribution to the synapse chapter, Raphael Berner for his contribution to the retina chapter, and Anton Civit for his contribution to the hardware infrastructure chapter. They acknowledge Kwabena Boahen for use of the material in the communications chapter; Diana Kasabov for proofreading and corrections; and Daniel Fasnacht for setting up the original DokuWiki at the start of the book project. The editors also acknowledge the students of the *Neuromorphic Engineering I* course at the Institute of Neuroinformatics, University of Zürich and ETH Zürich who gave feedback on Chapters 7 and 8.

They further acknowledge the Institute of Neuroinformatics, University of Zurich and ETH Zurich; the NSF Telluride Neuromorphic Cognition Engineering Workshop; and the CapoCaccia Cognitive Neuromorphic Engineering Workshop.

The figure at the head of the Silicon Cochleas chapter and reproduced on the cover is courtesy of Eric Fragnière. The figure at the head of the Learning in Neuromorphic Systems chapter is courtesy of Valentin Nägerl and Kevan Martin. The figures at the head of the Silicon Neurons and Silicon Synapses chapters are courtesy of Nuno Miguel Maçarico Amorim da Costa, John Anderson, and Kevan Martin.

List of Abbreviations and Acronyms

1D	one dimensional
2D	two dimensional
3D	three dimensional
ACA	analog computing arrays
ACK	acknowledge
A/D	analog–digital (converter)
ADC	analog–digital converter
AdEx	Adaptive exponential integrate-and-fire model
AE	address event
AEB	address-event bus
AER	address-event representation
AEX	AER extension board
AFGA	autozeroing floating-gate amplifier
AGC	automatic gain control
ALOHA	Not actually an abbreviation, ALOHA refers to a network media access protocol originally developed at the University of Hawaii
ANN	artificial neural network
ANNCORE	analog neural network core
API	Application Programming Interface
APS	active pixel sensor
AQC	automatic Q (quality factor) control
ARM	Acorn RISC Machine
ASIC	application-specific integrated circuit
ASIMO	Advanced Step in Innovative MObility (robot)
ASP	analog signal processor/processing
ATA	AT Attachment (also PATA: Parallel ATA); an interface standard for connecting mass storage devices (e.g., hard disks) in computers
ATIS	asynchronous time-based image sensor
ATLUM	Automatic Tape-collecting Lathe Ultra-Microtome
aVLSI	Analog very large scale integration
BB	bias buffer
BGA	ball grid array

BJT	bipolar junction transistor
BM	basilar membrane
BPF	band-pass filter
bps	bits per second
Bps	bytes per second
BSI	back-side illumination
C⁴	capacitively coupled current conveyor
CAB	computational analog block
CADSP	cooperative analog–digital signal processing
CAVIAR	Convolution AER Vision Architecture for Real-time
CCD	charge-coupled device
CCN	cooperative and competitive network
CCW	counter clockwise
CDS	correlated double sampling
CIS	CMOS image sensor
CLBT	compatible lateral bipolar transistor
CMI	current-mirror integrator
CMOS	complementary metal oxide semiconductor
CoP	center of pressure
CPG	central pattern generator
CPLD	complex programmable logic device
CPU	central processing unit
CSMA	carrier sense multiple access
CV	coefficient of variation
CW	clockwise
DAC	digital-to-analog converter
DAEB	domain address -event bus
DAVIS	Dynamic and Active-Pixel Vision Sensor
DC	direct current
DCT	discrete cosine transform
DDS	differential double sampling
DFA	deterministic finite automaton
DIY	do it yourself
DMA	direct memory access
DNC	digital network chip
DOF	degree(s) of freedom
DPE	dynamic parameter estimation
DPI	differential pair integrator
DPRAM	dual-ported RAM
DRAM	dynamic random access memory
DSP	digital signal processor/processing
DVS	dynamic vision sensor
EEPROM	electrically erasable programmable read only memory
EPSC	excitatory post-synaptic current
EPSP	excitatory post-synaptic potential
ESD	electrostatic discharge

ETH	Eidgenössische Technische Hochschule
EU	European Union
FACTS	Fast Analog Computing with Emergent Transient States
FE	frame events
FET	field effect transistor
FET	<i>also</i> Future and Emerging Technologies
FG	floating gate
FIFO	First-In First-Out (memory)
fMRI	functional magnetic resonance imaging
FPAA	field-programmable analog array
FPGA	field-programmable gate array
FPN	fixed pattern noise
FPS	frames per second
FSI	front side illumination
FSM	finite state machine
FX2LP	A highly integrated USB 2.0 microcontroller from Cypress Semiconductor Corporation
GALS	globally asynchronous, locally synchronous
GB	gigabyte, 2^{30} bytes
Gbps	gigabits per second
Geps	giga events per second
GPL	general public license
GPS	global positioning system
GPU	graphics processing unit
GUI	graphical user interface
HCO	half-center oscillator
HDL	Hardware Description Language
HEI	hot electron injection
HH	Hodgkin–Huxley
HiAER	hierarchical AER
HICANN	high input count analog neural network
HMAX	Hierarchical Model and X
HMM	Hidden Markov Model
HTML	Hyper-Text Markup Language
HW	hardware
HWR	half-wave rectifier
hWTA	hard winner-take-all
I&F	integrate-and-fire
IC	integrated circuit
IDC	insulation displacement connector
IEEE	Institute of Electrical and Electronics Engineers
IFAT	integrate-and-fire array transceiver
IHC	inner hair cell
IMS	intramuscular stimulation
IMU	inertial or intensity measurement unit
INCF	International Neuroinformatics Coordinating Facility

INE	Institute of Neuromorphic Engineering
I/O	input/output
IP	intellectual property
IPSC	inhibitory post-synaptic current
ISI	inter-spike interval
ISMS	intraspinal micro stimulation
ITD	interaural time difference
JPEG	Joint Photographic Experts Group
KB	kilobyte, 2^{10} bytes
keps	kilo events per second
LAEB	local address-event bus
LFSR	linear feedback shift register
LIF	leaky integrate-and-fire
LLN	log-domain LPF neuron
LMS	least mean squares
LPF	low-pass filter
LSM	liquid-state machine
LTD	long-term depression
LTI	linear time-invariant
LTN	linear threshold neuron
LTP	long-term potentiation
LTU	linear threshold unit
LUT	look-up table
LVDS	low voltage differential signaling
MACs	multiplyand accumulate operations
MB	megabyte, 2^{20} bytes
MEMs	microelectromechanical systems
Meps	mega events per second
MIM	metal insulator metal (capacitor)
MIPS	microprocessor without interlocked pipeline stages (a microprocessor architecture)
MIPS	<i>also</i> millions of instructions per second
MLR	mesencephalic locomotor region
MMAC	millions of multiply accumulate operations
MMC/SD	Multimedia card/secure digital
MNC	multi-neuron chip
MOSFET	metal oxide semiconductor field effect transistor
MUX	multiplex; multiplexer
NE	neuromorphic engineering
NEF	neural engineering framework
nFET	n-channel FET
NMDA	N-Methyl-d-Aspartate
NoC	Network on Chip
NSF	National Science Foundation
OHC	outer hair cell
OR	Octopus Retina

ORISYS	orientation system
OS	operating system
OTA	operational transconductance amplifier
PC	personal computer
PCB	printed circuit board
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PDR	phase dependent response
pFET	p-channel FET
PFM	pulse frequency modulation
PLD	programmable logic device
PRNG	pseudo-random number generator
PSC	post-synaptic current
PSRR	power supply rejection ratio
PSTH	peri-stimulus time histogram
PTAT	proportional to absolute temperature
PVT	process, voltage, temperature
PWM	pulse width modulation
PyNN	Python for Neural Networks
Q	quality factor of filter
QE	quantum efficiency
QIF	quadratic integrate-and-fire
QVGA	Quarter Video Graphics Array; 320 × 240 pixel array
RAM	random access memory
REQ	request
RF	radio frequency
RF	<i>also</i> receptive field
RFC	Request for Comments(a publication of the Internet Engineering Task Force and the Internet Society)
RISC	reduced instruction set computing
RMS	root mean square
RNN	recurrent neural network
ROI	region of interest
SAC	selective attention chip
SAER	serial AER
SAM	spatial acuity modulation
SATA	serial ATA (an interface standard for connecting mass storage devices (e.g., hard disks) to computers, designed to replace ATA)
SATD	sum of absolute timestamp differences
SC	spatial contrast
S-C	switched-capacitor
SCX	Silicon Cortex
SDRAM	synchronous dynamic random access memory
SerDes	serializer/deserializer
SFA	spike-frequency adaptation
SiCPG	silicon CPG

SIE	serial interface engine
SiN	silicon neuron
SNR	signal-to-noise ratio
SOS	second-order section
SpiNNaker	spiking neural network architecture
SRAM	static random access memory
SS	shifted source
SSI	stacked silicon interconnect
STD	short-term depression
STDP	spike timing-dependent plasticity
STRF	spatiotemporal receptive field
SW	software
sWTA	soft winner-take-all
TC	temporal contrast
TCAM	ternary content-addressable memory
TCDS	time correlated double sampling
TCP	Transport Control Protocol
TN	TrueNorth
TTFS	time to first spike
UCSD	University of California at San Diego
UDP	User Datagram Protocol
USB	universal serial bus
USO	unit segmental oscillators
V1	primary visual cortex
VGA	Video Graphics Array; 640×480 pixel array
VHDL	Verilog Hardware Description Language
VISe	VIision Sensor
VLSI	very large scale integration
VME	VERSAbus Eurocard bus standard
VMM	vector-matrix multiplication/multiplier
WABIAN-2R	WAseda BIpedal humANoid No. 2 Refined (robot)
WKB	Wentzel–Kramers–Brillouin
WR_OTA	wide-linear range OTA
WTA	winner-take-all
XML	Extensible Markup Language
ZMP	zero moment point

1

Introduction

The effortless ability of animal brains to engage with their world provides a constant challenge for technology. Despite vast progress in digital computer hardware, software, and system concepts, it remains true that brains far outperform technological computers across a wide spectrum of tasks, particularly when these are considered in the light of power consumption. For example, the honeybee demonstrates remarkable task, navigational, and social intelligence while foraging for nectar, and achieves this performance using less than a million neurons, burning less than a milliwatt, using ionic device physics with a bulk mobility that is about 10 million times lower than that of electronics. This performance is many orders of magnitude more task-competent and power-efficient than current neuronal simulations or autonomous robots. For example, a 2009 ‘cat-scale’ neural simulation on a supercomputer simulated 10^{13} synaptic connections at 700 times slower than real time, while burning about 2 MW (Ananthanarayanan et al. 2009); and the DARPA Grand Challenge robotic cars drove along a densely GPS-defined path, carrying over a kilowatt of sensing and computing power (Thrun et al. 2007).

Although we do not yet grasp completely nature’s principles for generating intelligent behavior at such low cost, neuroscience has made substantial progress toward describing the components, connection architectures, and computational processes of brains. All of these are remarkably different from current technology. Processing is distributed across billions of elementary units, the neurons. Each neuron is wired to thousands of others, receiving input through specialized modifiable connections, the synapses. The neuron collects and transforms this input via its tree-like dendrites, and distributes its output via tree-like axons. Memory instantiated through the synaptic connections between neurons is co-localized with processing through their spatial arrangements and analog interactions on the neurons’ input dendritic trees. Synaptic plasticity is wonderfully complex, yet allows animals to retain important memories over a lifetime while learning on the time scale of milliseconds. The output axons convey asynchronous spike events to their many targets via complex arborizations. In the neocortex the majority of the targets are close to the source neuron, indicating that network processing is strongly localized, with relatively smaller bandwidth devoted to long-range integration.

The various perceptual, cognitive, and behavioral functions of the brain are systematically organized across the space of the brain. Nevertheless at least some aspects of these various

processes can be discerned within each specialized area, and their organization suggests a coalition of richly intercommunicating specialists. Overall then, the brain is characterized by vast numbers of processors, with asynchronous message passing on a vast point-to-point wired communication infrastructure. Constraints on the construction and maintenance of this wiring enforce a strategy of local collective specialization, with longer range coordination.

For the past two decades neuromorphic engineers have grappled with the implementation of these principles in integrated circuits and systems. The opportunity of this challenge is the realization of a technology for computing that combines the organizing principles of the nervous system with the superior charge carrier mobility of electronics. This book provides some insights and many practical details into the ongoing work toward this goal. These results become ever more important for more mainstream computing, as limits on component density force ever more distributed processing models.

The origin of this neuromorphic approach dates from the 1980s, when Carver Mead's group at Caltech came to understand that they would have to emulate the brain's style of communication if they were to emulate its style of computation. These early developments continued in a handful of laboratories around the world, but more recently there has been an increase of development both in academic and industrial labs across North America, Europe, and Asia. The relevance of the neuromorphic approach to the broader challenges of computation is now clearly recognized (Hof 2014). Progress in neuromorphic methods has been facilitated by the strongly cooperative community of neuroscientists and engineers interested in this field. That cooperation has been promoted by practical workshops such as the Telluride Neuromorphic Cognition Engineering Workshop in the United States, and the CapoCaccia Cognitive Neuromorphic Engineering Workshop in Europe.

Event-Based Neuromorphic Systems arose from this community's wish to disseminate state-of-the-art techniques for building neuromorphic electronic systems that sense, communicate, compute, and learn using asynchronous event-based communication. This book complements the introductory textbook (Liu et al. 2002) that explained the basic circuit building blocks for neuromorphic engineering systems. *Event-Based Neuromorphic Systems* now shows how those building blocks can be used to construct complete systems, with a primary focus on the hot field of event-based neuromorphic systems. The systems described in this book include sensors and neuronal processing circuits that implement models of the nervous systems. Communication between the modules is based on the crucial asynchronous event-driven protocol called the address-event representation (AER), which transposes the communication of spike events on slow point-to-point axons, into digital communication of small data packets on fast buses (see, for example, Chapter 2). The book as a whole describes the state of the art in the field of neuromorphic engineering, including the building blocks necessary for constructing complete neuromorphic chips and for solving the technological challenges necessary to make multi-chip scalable systems. A glance at the index shows the wide breadth of topics, for example, next to 'Moore's law' is 'motion artifact' and next to 'bistable synapse' is 'bootstrapped mirror.'

The book is organized into two parts: Part I (Chapters 2–6) is accessible to readers from a wider range of backgrounds. It describes the range of AER communication architectures, AER sensors, and electronic neural models that are being constructed without delving exhaustively into the underlying technological details. Several of these chapters also include a historical tree that helps relate the architectures and circuits to each other, and that guides readers to the extensive literature. It also includes the largely theoretical Chapter 6 on learning in event-based systems.

Part II (Chapters 7–16) is addressed to readers who intend to construct neuromorphic electronic systems. These readers are assumed to be familiar with transistor physics (particularly subthreshold operation), and in general to be comfortable with reasoning about analog CMOS circuits. A mixed-signal CMOS designer should be comfortable reading these more specialized topics, while an application engineer would be able easily to follow the chapters on hardware and software infrastructure. This part of the book provides information about the various approaches used to construct the building blocks for the sensors and computational units modeling the nervous system, including details of silicon neurons, silicon synapses, silicon cochlea circuits, floating-gate circuits, and programmable digital bias generators. It also includes chapters on hardware and software communication infrastructure and algorithmic processing of event-based sensor output.

The book concludes with Chapter 17, which considers differences between current computers and nervous systems in the ways that computational processing is implemented, and discusses the long-term route toward more cognitive neuromorphic systems.

1.1 Origins and Historical Context

Many of the authors of *Event-Based Neuromorphic Systems* were strongly influenced by *Analog VLSI and Neural Systems* (Mead 1989). Carver Mead's book was the story of an extended effort to apply the subthreshold transistor operating region of CMOS electronics to realize a neural style and scale of computation. The book was written at a time when automatically compiled synchronous logic circuits were just beginning to dominate silicon production, a field that Mead was central in creating. Much like the famous Mead and Conway (1980) book on logic design, which was focused toward instilling a set of methodologies for practical realization of logic chips in digital designers, *Analog VLSI and Neural Systems* was focused on providing a set of organizing principles for neuromorphic designers. These ideas were centered around the name of Mead's group at Caltech, the Physics of Computation group, and emphasized notions such as signal aggregation by current summing on wires, multiplication by summed exponentials, and relations between the fundamental Boltzmann physics of energy barriers and the physics of activation of voltage-sensitive nerve channels.

However, at that time the field was so new that there were many practical aspects that did not work out in the long run, mainly because they suffered from transistor mismatch effects. So the early systems were good for demonstration but not for real-world application and mass production. The fact that current copying in CMOS is the least precise operation possible to implement in practice, was barely mentioned in the book. This omission led to designs that worked ideally in simulation but functioned poorly in practice. In relation to *Event-Based Neuromorphic Systems*, the central importance of communication of information was not realized until after the book was completed, and so none of the systems described in the book had an AER output; rather the analog information was scanned out serially from the systems described there. Even a later collection of chapters (Mead and Ismail 1989) about Mead-lab systems and Mead's review paper in Proceedings of the IEEE (1990) barely touched on communication aspects.

Since 1989 there has been a continued drive to improve the technology of neuromorphic engineering. But to place the progress of neuromorphic engineering in context, we can consider logic, that is, digital chip design. Around 1990, a high-end personal computer had about 8 MB

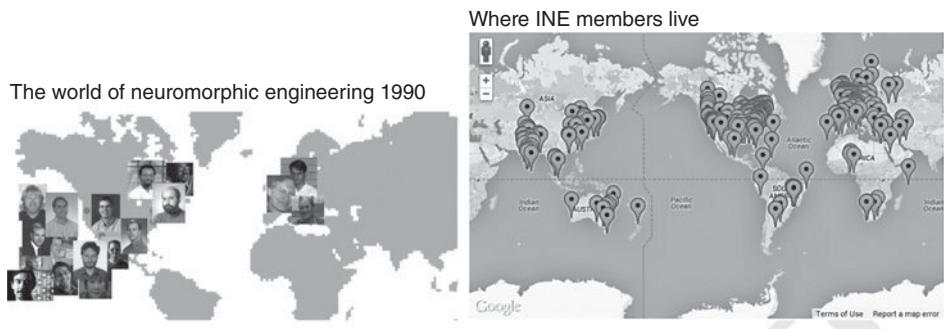


Figure 1.1 Maps of the neuromorphic electrical engineering community in 1990 (left) and 2013 (right), © 2013 Google)

of RAM and about 25 MHz clock speed (one of the authors remembers being a proud owner of a personal CAD station that could be used to work on chip design at home). As of 2013, a state-of-the-art personal computer has about 16 GB of memory and 3 GHz clock speed. So in about 20 years we have seen approximately a 1000-fold increase in memory capacity and a 100-fold increase in clock speed. These of course are reflections of Moore's law and investments of hundreds of billions of dollars. But the basic organizing principles used in computation have hardly changed at all. Most advances have come about because of the availability of more raw memory and computing power, not by fundamental advances in architectures.

During this period the neuromorphic engineering community has expanded considerably from its origins at Caltech, Johns Hopkins, and EPFL (Figure 1.1). At first only a few modest, rather unconvincing lab prototypes could be shown in a couple of labs, and these barely made it off the lab bench. But, after 20 years, neuromorphic engineering has scaled the number of spiking neurons in a system from a few hundred up to about a million (Chapter 16), neuromorphic sensors are available as high-performance computer peripherals (Liu and Delbrück 2010), and these components can be used by people at neuromorphic engineering workshops who know little about transistor-level circuit design (Cap n.d.; Tel n.d.). The literature shows a steady exponential growth in papers with the keywords 'neuromorphic' or 'address-event representation' (Figure 1.2), which is a higher growth rate than for the term 'synchronous logic.' Although the slopes of these exponentials tend to flatten over time, the number of

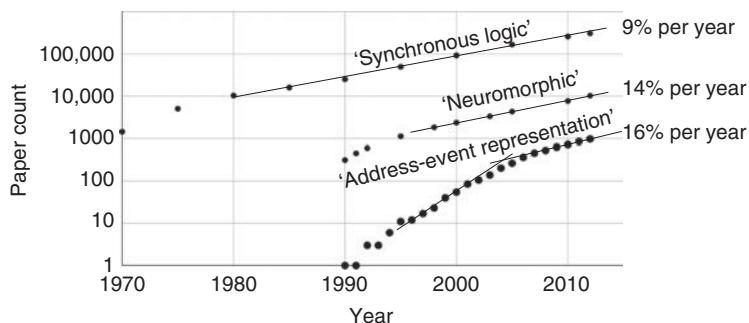


Figure 1.2 Growth of literature over time. From Google Scholar

papers mentioning ‘address-event representation’ has increased for the last 5 years at the rate of about 16% per year. If this growth is considered as resulting from perhaps 15 labs working for an average of 15 years at an investment of \$200,000 per year, then this progress has been achieved at a total financial investment of perhaps 50 million dollars, a tiny fraction of the hundreds of billions spent on developing conventional electronics during this period.

1.2 Building Useful Neuromorphic Systems

To be adopted as mainstream technology and have the financial support and competitive environment of an active industrial market there are some obvious requirements. Neuromorphic systems must function robustly and repeatably across chips, across temperature, with noisy power supplies, must have interfaces that allow easy development of applications, and need to be portable for use in the field without specialized equipment. *Event-Based Neuromorphic Systems* teaches the knowledge of the required technologies built up over the past 20 years of effort.

These features of a neuromorphic electronic system are necessary but not sufficient. A neuromorphic system must outperform conventional technology, or at least justify the investment of effort based on the belief that it could outperform conventional approaches when scaled up or when silicon technology can no longer scale down to smaller feature sizes or power supply voltages. And this last point has been a weakness: Proposals have not convincingly shown that the neuromorphic approach is better than simply scaling logic and making it more parallel. Many grants have been funded nonetheless, but the proposals are too vague to be very credible. One could say that funders are simply hopeful; there is no alternative that offers anything but a new device technology (e.g., graphene) to enable yet more clock and feature size scaling.

The scaling problem brings up the importance of communication: To scale up systems requires growing not only smaller in feature size and cost but also larger in system capabilities. To be neuromorphic, these systems must emulate something like the hybrid data-driven computation and communication architecture used in brains with their massive numbers of connections. One can see the evidence for this requirement from the direction of conventional electronics as well, with logic systems becoming more parallel and distributed. This requirement for communication is why the neuromorphic community has focused its efforts on event-based architectures and it is why this book is aimed at teaching the state-of-the-art techniques for building such systems. Chapter 2 will begin by outlining the principles of event-based communication architectures for neuromorphic systems.

References

- Ananthanarayanan R, Esser SK, Simon HD, and Modha DS. 2009. The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, OR, November 14– 20, 2009. IEEE. pp. 1–12.
- Cap. n.d. Capo Caccia Cognitive Neuromorphic Engineering Workshop, <http://capocaccia.ethz.ch/> (accessed July 16, 2014).
- Hof RD. 2014. Qualcomm’s neuromorphic chips could make robots and phones more astute about the world. *MIT Technology Review*. <http://www.technologyreview.com/featuredstory/526506/neuromorphic-chips/>.
- Liu SC and Delbrück T. 2010. Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* **20**(3), 288–295.
- Liu SC, Kramer J, Indiveri G, Delbrück T, and Douglas R. 2002. *Analog VLSI: Circuits and Principles*. MIT Press.

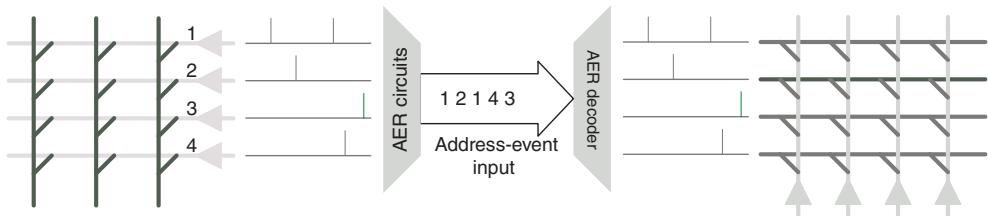
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Mead CA. 1990. Neuromorphic electronic systems. *Proc. IEEE* **78**(10), 1629–1636.
- Mead CA and Conway L. 1980. *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA.
- Mead CA and Ismail M (eds). 1989. *Analog VLSI Implementation of Neural Systems*. Kluwer Academic Publishers, Norwell, MA.
- Tel. n.d. Telluride Neuromorphic Cognition Engineering Workshop, www.ine-web.org/ (accessed July 16, 2014).
- Thrun S, Montemerlo M, Dahlkamp H, Stavens D, Aron A, Diebel J, Fong P, Gale J, Halpenny M, Hoffmann G, Lau K, Oakley C, Palatucci M, Pratt V, Stang P, Strohband S, Dupont C, Jendrossek LE, Koelen C, Markey C, Rummel C, Niekerk J, Jensen E, Alessandrini P, Bradski G, Davies B, Ettinger S, Kaehler A, Nefian A, and Mahoney P. 2007. Stanley: the robot that won the DARPA grand challenge. In: *The 2005 DARPA Grand Challenge* (eds Buehler M, Iagnemma K, and Singh S). Vol. 36: Springer Tracts in Advanced Robotics. Springer, Berlin Heidelberg. pp. 1–43.

Part I

Understanding Neuromorphic Systems

2

Communication



This chapter focuses on the fundamentals of communication in event-based neuromorphic electronic systems. Overall considerations on requirements for communication and circuit- versus packet-switched systems are followed by an introduction to Address-Event Representation (AER), asynchronous handshake protocols, address encoders, and address decoders. There follows a section on considerations regarding trade-offs in the design of AER links, and a section describing the details of the implementation of such links and how these have evolved.

2.1 Introduction

In evolution, the great expansion of computational power of brains seems to be implemented by expansion of cortex, a sheet of tissue surrounding older structures. Cortex is a layered structure divided into what is known as the gray matter and white matter. Figure 2.1 shows a cross section of a small chunk of visual cortex of a cat brain. The many types of neurons, of which there are about $10^5/\text{mm}^3$, make long-range connections through axons (their output branches) in the white matter (with a wiring density of 9 m of axon per mm^3) while the gray matter is composed mostly of dendrites (the neurons' input branches) with a wiring density of an amazing $4 \text{ km}/\text{mm}^3$ (Braitenberg and Schüz 1991). The long-range white matter connections occupy much more volume because they are myelinated, that is, thickly sheathed in a material called myelin. The myelin acts as an insulator, reducing the capacitance and

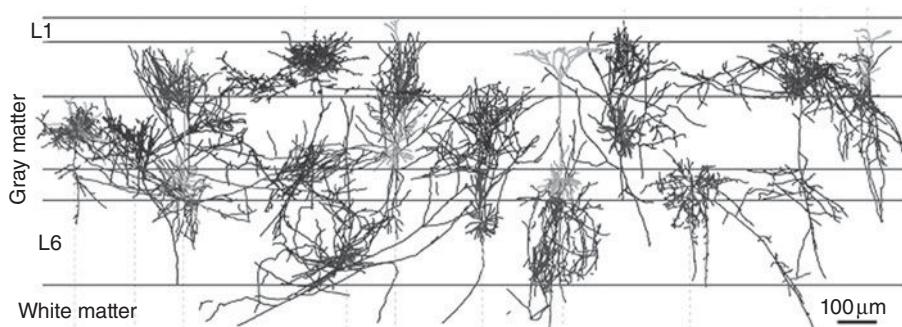


Figure 2.1 Cross-sectional view of layered cortical cells with their dendrites within the gray matter and axons projecting out to white matter. Only a few cells are shown, but gray matter is completely filled with neurons and some other supporting cells. Adapted from Binzegger et al. (2004). Reproduced with permission of the Society for Neuroscience

increasing the resistance to the outside of the cell to reduce decay of the action potential impulses by which the neurons communicate, as these impulses travel along the axons.

Action potentials, or so-called spikes from the shapes of their waveforms, whether traveling along axons in white matter or unmyelinated axons in gray matter are stereotypical (Gerstner and Kistler 2002). Although their amplitudes, durations (around 1 ms), and precise shapes can vary somewhat, they can be treated as all-or-none, essentially digital events.

Neuromorphic electronic systems must embed complex networks of neurons, axons, and synapses, which nature builds in three dimensions (3D), into an essentially 2D silicon substrate. Unlike standard digital logic where the output of a gate, on average, is connected to the input of three to four other gates, a neuron typically delivers a spike to thousands of destinations. Hence there is a fundamental *physical* mismatch between the logic-optimized, 2D silicon technology and the interconnectivity requirements for implementing biological networks of neurons. This mismatch is overcome using time-multiplexed communication.

Modern digital gates have switching delays that are on the order of tens of picoseconds, which is many orders of magnitude faster than the time constant of the output spiking activity in a neuron. Since the communication fabric is only carrying spikes from one neuron to another, the high connectivity problem as well as the 3D/2D mismatch can be resolved by using a time-multiplexed fabric for interneuron communication. Such a network is very similar to a packet-switched communication network used in on-chip, chip-to-chip, and system-to-system communication (e.g., the Internet) today.

Communication networks can be broadly divided into two major categories: circuit switched and packet switched. In a circuit-switched network, two end points communicate by setting up a *virtual circuit* – a path through the network that is dedicated for the communication between the two end points. Once the path is setup, data can be communicated over that path. Once the communication ends, the virtual circuit is destroyed and the hardware resources used by the circuit are released for use by another virtual circuit. This approach was used in the original telephone network. Communication in a circuit-switched network has a setup/teardown cost for the virtual circuit, after which the dedicated communication circuit can be used with very

low overhead. Such networks are efficient when the communication between two end points occurs for a very long duration.

Packet-switched networks, on the other hand, operate by time-multiplexing individual segments of the network. Instead of creating an end-to-end path up front, each item being communicated (*a packet*) requests access to shared resources on the fly. Hence, each packet must contain sufficient information that allows each step in the communication network to determine the appropriate next step in the path taken by the packet. Large messages in packet-switched networks are typically *packetized* – converted into a sequence of packets, where each packet contains replicated path information. However, there is no overhead in sending the first packet through the network. Such networks are efficient when the communication between two end points is in bursts of small amounts of information.

A neuron spiking event is only used to convey a small amount of information in typical neuromorphic electronic systems. In the extreme case, the only information conveyed by a spike is the fact that the spike occurred at all – that is, the *time* at which the spike occurred relative to other spikes in the system. A very sophisticated model might attempt to model the spike waveform directly, and convey a certain number of bits required to reconstruct the waveform to a certain degree of precision. Most large-scale neuromorphic electronic systems model spikes with a small number of bits of precision. Hence interneuron communication is typically implemented using packet-switched networks as they use hardware resources more effectively when small amounts of information are exchanged.

Representation of time is a nontrivial task, as temporal resolution is an important factor in the design of communication networks for spiking neurons. There are two major approaches to representing time. (i) Discrete time: In this approach, time is discretized into global *ticks*, and a communication network is designed to deliver time-stamped spikes at the appropriate global time step. (ii) Continuous time: In this approach, the entire system operates in continuous time, and spike delays are also modeled with continuous time electronics. This is a challenging design problem, and practitioners of this approach typically make use of the following set of observations:

- Neuron spiking rates are very low (tens of Hz) compared to the speed of digital electronics (GHz). This means that a fast communication fabric operating in the tens or hundreds of MHz regime would be idle almost all the time.
- Axonal delays are also on the order of milliseconds compared to the switching delay of gates (tens of picoseconds).
- A very small (<0.1%) variation in spike arrival time should not have a significant impact on overall system behavior, because biological systems are known to be very robust and should be able to adapt to a small variation in spike arrival time.

Combining these three observations leads to the conclusion that we can ignore the uncertainty in spike delivery time if it can be kept in the order of microseconds, since the dominant delay term is the one introduced by the time constant of the neuron itself (tens of milliseconds) or the axonal delay model (milliseconds). For this approach to be successful, it is important for the communication fabric to be *over-engineered* so that the network is never heavily loaded. This philosophy is sometimes described by stating that ‘time represents itself’ – that is, the arrival time of spikes itself represents the time at which the spikes should be delivered. This relies on real-time behavior of spiking networks and their silicon implementation.

2.2 Address-Event Representation

Typical neuron firing rates are in the regime of 1–10 Hz. Hence, thousands or even millions of neurons combined have spiking rates in the KHz to low MHz regime. This data rate can be easily supported by modern digital systems. Hence, instead of creating a network of individual neurons, neuromorphic electronic systems have end points that correspond to clusters of neurons, where a cluster can correspond to a specific processing layer. The circuits used to multiplex communication for a cluster of neurons into an individual communication channel are referred to as *address event representation* (AER) circuits. AER was first proposed in 1991 by Mead's lab at Caltech (Lazzaro et al. 1993; Lazzaro and Wawrynek 1995; Mahowald 1992, 1994; Sivilotti 1991), and has been used since then by a wide community of hardware engineers.

The function of AER circuits is to provide multiplexing/demultiplexing functionality for spikes that are generated by/delivered to an array of individual neurons. Figure 2.2 shows an example of how the spiking behavior of four neurons is encoded onto a single output channel.

Spikes are generated asynchronously, and the AER circuits accept spikes as they are generated and multiplex them onto a single output channel. The sequence of values produced on the output channel indicates which neuron fired. The time at which the neuron identifier is generated corresponds to the time at which the neuron produced a spike, plus a small delay due to the encoding process. As long as spikes are sufficiently separated in time, the encoding process ensures that the neuron identifiers are correctly ordered.

If each neuron in the cluster were guaranteed to only produce a spike when no other neuron in the same cluster was spiking, then the multiplexing circuits would correspond to a standard asynchronous encoder circuit. However, this is not a valid constraint as groups of neurons in a cluster could have similar/overlapping firing times. AER encoders therefore generally use arbitration logic to handle potentially simultaneous spike arrival times from multiple neurons.

The demultiplexing circuits are easier to design, and are illustrated in Figure 2.3. In the demultiplexing process, an input value received from an AER channel specifies the axon/dendrite (depending on your perspective) identifier to which the spike is to be delivered. The dendrite number is decoded using an asynchronous decoder, and the spike is delivered to the appropriate destination. These decoder circuits are identical to those found in self-timed memory structures. In the example shown in Figure 2.3, the address-event sequence is decoded into spikes that are delivered to individual dendrites that in turn are connected to the appropriate neurons. If the delays between adjacent dendrite identifiers in the AER input are sufficiently large, then the output of the AER decoder delivers a spike to the dendrite at the time that corresponds to the arrival time of the AER input plus a small delay corresponding

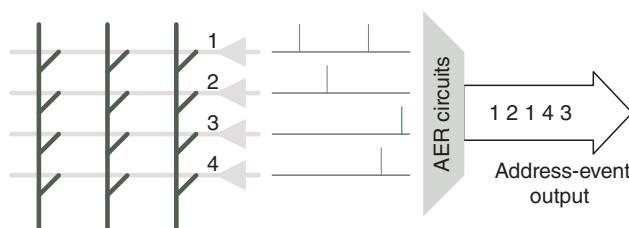


Figure 2.2 Multiplexing different neurons onto a single communication channel

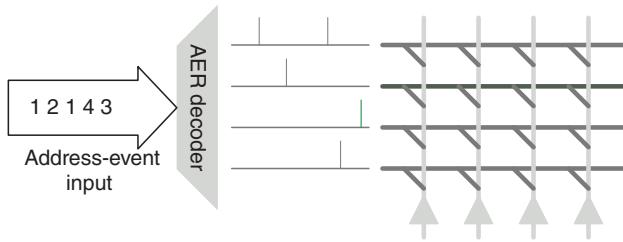


Figure 2.3 Demultiplexing an AER input into individual spikes for each dendrite

to the decoder circuit. The combination of a self-timed AER encoder and decoder results in spikes being delivered from source neurons to destination neurons in a manner that preserves the inter-arrival delay among spikes.

In the simplest possible AER scheme, neuron identifiers generated by the AER encoder directly correspond to the addresses of dendrites at the destination. For example, if we directly connect the output of Figure 2.2 to the input in Figure 2.3 as shown in the figure at the head of this chapter, that provides a direct connection between the axon and dendrite at the source and destination. This is useful if, for example, the source and destination are on separate chips, or if the encoding/decoding process makes the wiring between the axons and dendrites more manageable. In more complex schemes, the neuron identifiers are translated into the appropriate individual or set of destination identifiers for spike delivery.

2.2.1 AER Encoders

AER encoders have many individual axons as input, and they have to implement two functions. First, the encoder must determine which is the next spike to be communicated on the output channel. This corresponds to the traditional arbitration problem in asynchronous systems. Once the spiking axon has been selected, the circuit must encode the axon identifier and produce an output data value. This is a traditional encoder, where one of N different integers is encoded using $\log N$ bits. There is a variety of schemes in the literature for AER encoders. These schemes differ in the mechanisms they use for resolving conflicts between multiple simultaneous spikes, and how the neuron identifiers are encoded. Each scheme has its strengths and weaknesses, and is appropriate for different types of neuromorphic systems.

There are two common methods to organize AER encoders. In the simplest mechanism, the neurons are logically organized in a linear array, and a single encoder is used to collect the spikes into an output channel. This approach was used in the implementation of cochlear circuits (Lazzaro et al. 1993). When the number of neurons becomes very large, a common practice is to arrange the neurons in a 2D array and use two separate encoders to encode the x -address and y -address of the neuron that spiked. This is especially popular in retina models where neurons are naturally organized in a 2D spatial grid.

2.2.2 Arbitration Mechanisms

Arbitration refers to the process of selecting the order of access to a shared resource among asynchronous requests. In AER systems the shared resource is the bus that carries the address

events. If we provide random access to a shared communication channel or bus, we have to deal with contention for channel access, which occurs when two or more elements (in our case neurons) attempt to transmit their addresses simultaneously. We can introduce an arbitration mechanism to resolve contention and a queuing mechanism to allow nodes to wait for their turn. The queues introduced can be at the source (per neuron), shared (per AER encoder), or a combination of the two.

Bus Sensing

The most straightforward mechanism to implement an arbitration scheme resembles the carrier sense multiple access (CSMA) protocols that are used in wireless networking as well as in the original Ethernet protocol (IEEE 802.3 working group 1985). This approach involves circuitry to detect if the AER bus is occupied; a spike is transmitted only if the bus is free. If the bus is busy, then the spike is discarded (Abusland et al. 1996). This approach has the advantage that spikes on the AER bus are transmitted at a predictable time relative to when the spike occurred. However, spikes may be lost due to bus contention.

Tree Arbitration

The most popular arbitration method is to use the classical arbiter circuit to determine the order in which active on-chip elements communicate their address. The addresses of spiking elements are effectively in a queue and are transmitted off-chip as bus occupancy permits (Boahen 2000). No events will be lost, but spike timing information will not be preserved as soon as multiple elements are queuing for access to the bus. Spikes that do not win the arbitration process are queued at the neuron. This approach was first proposed in (Mahowald 1992). If we imagine that the permission to access the shared AER output channel is visualized as a *token* (shown as a dot in Figure 2.4), then the process of requesting access to the output channel can be viewed as sending a request for the token to the root of a tree of two-way

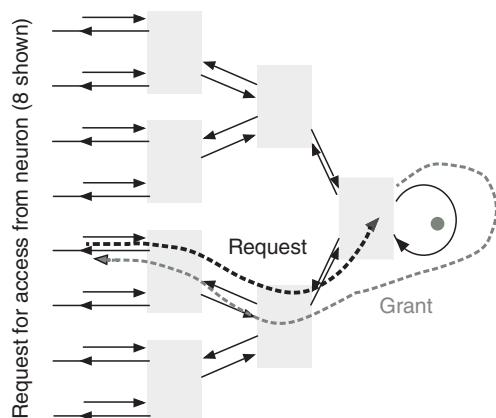


Figure 2.4 Tree arbitration scheme for AER

arbiter elements. The token then moves down the tree to one of the requesting neurons. This is illustrated in Figure 2.4.

In the standard scheme shown in Figure 2.4, each arbiter handles an input request by first sending a request for the token up the tree, and then responding to the input request after it has received the token. After the input has been handled, the token is returned to the root of the tree. When the number of neurons being handled becomes large, this introduces a penalty of $O(\log N)$ stages of arbitration per request. Note that this delay impacts the *throughput* of spike communication, since the shared AER bus cannot be used until permission to access it is obtained through the arbitration mechanism.

Greedy Tree Arbitration

An optimized version of the tree arbiter scheme uses the notion of a *greedy arbiter* (Boahen 2000). The greedy arbiter is a modified arbiter circuit that supports an optimized mechanism to handle simultaneous requests. If both inputs to an arbiter block request access to the token within a short interval of each other, then after one of the inputs is handled the token is propagated to the other input before returning it to the root of the tree. This can be viewed conceptually as providing a short-circuit path for token propagation at each level of the tree, and is illustrated in Figure 2.5.

If there is very little spike activity, then the greedy approach has similar performance to the tree arbitration approach. If there is significant spatially correlated spike activity, then the greedy path is activated and spikes can be serviced without having the token reach the root of the arbitration tree.

Ring Arbitration

A third approach is to construct a token-ring scheme for arbitration. In this scheme, illustrated in Figure 2.6, a ring of arbitration elements is constructed with the token staying at a fixed

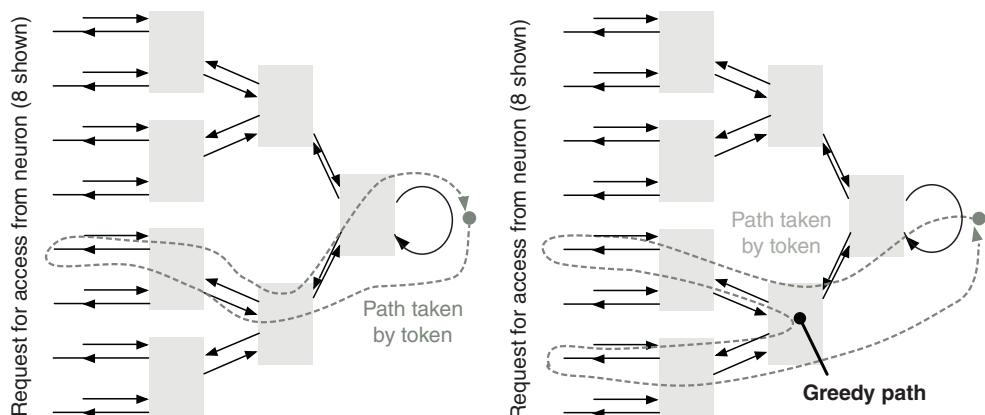


Figure 2.5 Token movement in basic arbiter scheme versus the greedy arbiter scheme

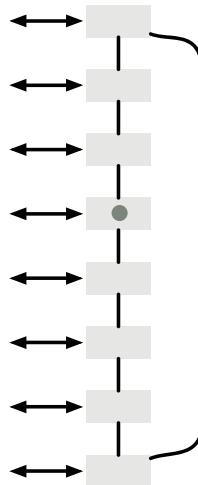


Figure 2.6 Token ring arbitration scheme

location until a request for the token travels around the ring until the token is found. It then moves to the requested location. This mechanism has benefits if the average distance traveled by the token is small, but has drawbacks if the token has to travel a long distance between spikes (Imam and Manohar 2011).

Multidimensional Arbitration

The arbitration mechanisms described so far are suitable for selecting one out of N neurons that need to access the AER bus. If neurons are organized in a $\sqrt{N} \times \sqrt{N}$ array (e.g., in a silicon retina), then each row of neurons would require $O(\sqrt{N})$ wires (one per neuron in the row). This is not a scalable approach as N becomes large. Instead, an alternative is to adopt a *2D arbitration* scheme.

2D arbitration schemes provide another opportunity for optimization. If we imagine the neurons organized in a 2D array, then there are two levels of arbitration necessary to enforce mutual exclusion among all the neurons: (i) row arbitration, which selects a row where a spike was produced; (ii) column arbitration, which selects one of the columns in the currently selected row where a spike occurred. The combination of row and column arbitration uniquely selects the neuron, and requires fewer wires than a 1D arbitration approach (Mahowald 1992).

An optimization for the 2D arbitration scheme that has also been used is called *burst-mode* operation. In burst-mode operation, row arbitration is followed by saving the entire state of the selected row into a set of latches. After this, all the spikes in the selected row are scanned out one at a time. This, combined with an efficient encoding scheme that shares the row address, can lead to an efficient AER scheme when bursts of spikes in an individual row are expected (Boahen 2004c).

We return to the issue of arbitration in Section 12.2.2.

Eliminating Arbitration

Arbitration lengthens the communication cycle period, reducing the channel capacity, and queuing causes temporal dispersion with the loss of timing information. An alternative is to provide no arbitration. Mortara et al. (1995) allowed any active element to immediately place its address on a common bus. The elements must be assigned addresses in such a way that not all numerically possible addresses are valid and such that when two or more elements place their addresses on the bus simultaneously (this is referred to as a collision) this results in an invalid address. These must then be ignored by the receiver. Allowing collisions to occur, and discarding the invalid addresses so generated, has the advantage of achieving a shorter cycle period and reducing dispersion (spike timing is preserved when no collision occurs), but events will be lost and this loss will increase as the load increases.

2.2.3 Encoding Mechanisms

Once the appropriate neuron has been selected, the neuron address has to be encoded into a compact representation for the AER output channel. The standard mechanism to do this uses the fact that the grant lines for each neuron are a one-hot encoding of the neuron address. Therefore, a conventional logarithmic encoder can be designed where the grant lines are encoded into $\log N$ wires to constitute the neuron address (Mahowald 1992; Sivilotti 1991). This is illustrated in Figure 2.7.

Each grant line connects to $O(\log N)$ transistors, so the structure uses $O(N \log N)$ transistors to compute the encoded output value. An alternative is to use the fact that the choice made by arbiters at each level of the tree corresponds to selecting the value of each bit in the encoded output. For example, the leaves of the arbiter tree determine if the least significant bit of the encoded output is 0 or 1. Hence, if we encode each bit of the output at each level of the tree, then the encoded output can be constructed with fewer transistors. Each arbiter connects to

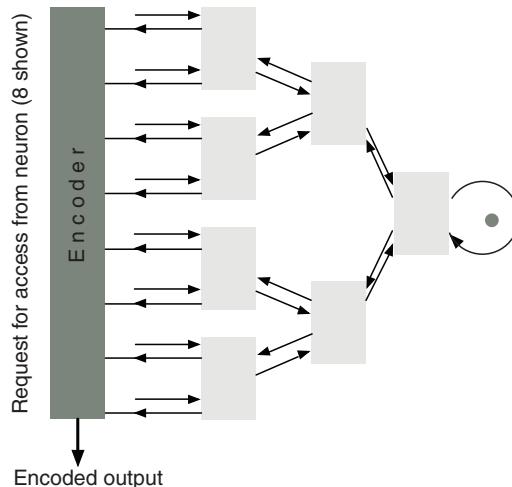


Figure 2.7 Basic encoder structure

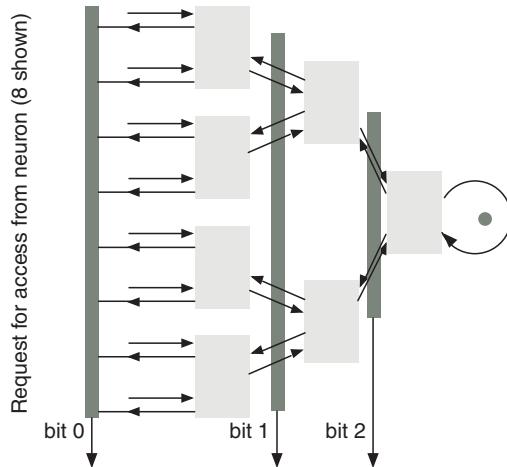


Figure 2.8 Hierarchical encoder structure

two different transistors only, making the total transistor count for this variant $O(N)$ (Georgiou and Andreou 2006). This method is illustrated in Figure 2.8.

A third approach maintains a counter that always tracks the current location of the token. Every time the token moves, the value of the counter is updated to reflect the current token position. For a linear arbitration structure, this corresponds to incrementing or decrementing a simple counter. For a tree-based arbitration structure, each bit in the counter is updated by a different level of the tree. This method is illustrated in Figure 2.9.

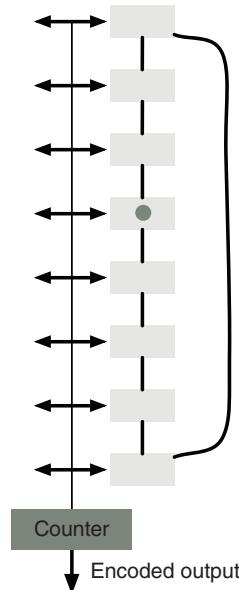


Figure 2.9 Counter-based encoder structure

2.2.4 Multiple AER Endpoints

So far we have described how spikes from a set of neurons can be multiplexed onto a shared AER channel, and how spikes encoded in this manner can be delivered to another set of neurons. There are two additional components necessary to complete an AER communication system: (i) a mapping from source neuron address to destination neuron/axon address; (ii) a routing architecture for multiple clusters of neurons.

2.2.5 Address Mapping

The AER encoder produces a sequence of spikes that are encoded via the source neuron address. The output of a particular source neuron n is connected to a specific destination axon, identified by its axon address a . Therefore, spikes with neuron identifier n must be remapped to address a so that they are delivered to the appropriate destination axon.

Sometimes this mapping function is quite simple – for example, if a chip with an array of neurons creates spikes that are delivered to corresponding neurons in another chip, then the mapping function would be the identity function and no translation/mapping is necessary.

In more general implementations, spikes from neuron n might have to be delivered to an arbitrary destination axon a . In such situations where the connectivity is programmed via software rather than hardcoded, a mapping table that implements the appropriate address translation is required. Programmable neuromorphic systems have implemented such tables using both on-chip (Lin et al. 2006) and off-chip memories.

Probably the most challenging aspect of implementing communication in neuromorphic systems is the fact that a spike created by a neuron is typically delivered to a very large number of destination neurons. Mechanisms that support high spike fan-out are discussed through case studies in Chapter 16.

2.2.6 Routing

A general neuromorphic system is organized with clusters of neurons, with each cluster typically corresponding to a set of neurons that are physically proximate. Since the entire system consists of many such clusters, a mechanism is needed to route spikes between clusters.

Many different routing topologies are possible, and here we provide a brief overview of some of the options available with a more in-depth description of architectures used for large-scale neuromorphic systems in Chapter 16. A detailed discussion of different routing architectures can be found in a number of text books (e.g., Dally and Towles 2004).

Point-to-Point Communication

The simplest topology for communication is a point-to-point communication architecture. In such a system, spikes from one cluster of neurons are simply transmitted to corresponding neurons in another cluster. This permits chaining of spike communication from one chip to the next, with each cluster of neurons performing additional processing before propagating spikes.

Rings and 1D Arrays

Rings and 1D arrays are linear structures with nearest-neighbor communication. Two commonly used addressing mechanisms include distance-based addressing and chip identifier-based addressing. In distance-based addressing, a spike from one chip is routed to a chip a certain distance away (e.g., routed to a chip three hops to the right). Each hop decrements the distance count, and a spike is accepted when the hop count is zero and delivered to the local cluster. In chip identifier-based addressing, an incoming spike identifier is matched against a local identifier (or, more generally, a table of potential identifiers) and accepted if the spike identifier matches the local chip information.

Meshes

Meshes extend the 1D topology to 2D. Routing is typically performed using dimension-ordered routing (sometimes called ‘XY’ routing or ‘X first’ routing). The standard approach to mesh routing is to specify a spike to be delivered to a $(\Delta x, \Delta y)$ offset relative to the current cluster, with one dimension being routed before the other.

Trees

Tree routing can be used as an alternative to deliver spikes. The route can be specified as a sequence of *turns* in the tree. In the simplest case, a packet traveling along an edge of the binary tree always has two options for the next hop, and a sequence of bits can be used to identify the complete path from source cluster to destination cluster.

2.3 Considerations for AER Link Design¹

The bandwidth of the communication links between the communicating processes or blocks is a critical specification. Initial analysis of performance figures for these links was explored by Mortara and Vittoz (1994) and subsequently extended by Boahen (2000).

The performance of event-driven communication links can be measured by five criteria: capacity, throughput, latency, integrity, and dispersion (see Table 2.1). Capacity is defined as the reciprocal of the minimum transmission time; this is the maximum rate at which events can be transmitted and received on the link. Throughput is defined as the usable fraction of capacity; the maximum rate is rarely sustainable in practice. Latency is defined as the mean delay; this wait time may be several transmission slots. Latency depends on the fraction of transmission slots that are filled. The fraction of the link capacity that is actually being used is called the load. Integrity is the fraction of spikes that are correctly delivered to the destination. Integrity is used to model the notion of networks that are allowed to drop spikes. Dispersion is defined as the standard deviation of the latency distribution. This metric determines how well spike timing properties are preserved.

Link designers strive not only to maximize throughput, but to minimize latency as well. High throughput allows large numbers of event generators operating over a broad range of rates to be serviced, while low latency preserves the timing of each individual event. Throughput is

¹ Most of the text in this section is © 2000 IEEE. Reprinted, with permission, from Boahen (2000).

Table 2.1 Time-multiplexed communication channel design options. © 2000 IEEE. Reprinted, with permission, from Boahen (2000)

Feature	Approaches	Remarks
Latency	Polling	\propto Number of neurons
	Event driven	\propto Active fraction
Integrity	Rejection	Collisions increase exponentially
	Arbitration	Queue events
Dispersion	Dumping	No waiting
	Queuing	\propto^{-1} surplus capacity
Capacity	Hardwired	Simple \Rightarrow Short cycle time
	Pipelined	\propto^{-1} slowest stage

optimized if collisions are prevented through arbitration; it is then limited only by the increase in latency with activity due to queuing (Boahen 2000; Deiss et al. 1999; Sivilotti 1991). To achieve a specified timing error, defined as the percentage error in a cell's inter-event interval, throughput must be capped at a level below the maximum link, or channel, capacity.

Several papers by Mortara and by Boahen analyze the trade-off between latency and throughput in these asynchronous parallel, read–write links and in some cases, these analytical results are validated using measurements from fabricated chips. Some of the original numbers have improved with each generation of chips through a combination of clear circuit design techniques, the use of modern faster CMOS processes, and the introduction of new communication protocols that speed up the transmission.

Given an information coding strategy, the communication channel designer faces several trade-offs. Should they preallocate the channel capacity, giving a fixed amount to each user, or allocate capacity dynamically, matching each user's allocation to his or her current needs? Should they allow users to transmit at will, or implement elaborate mechanisms to regulate access to the channel? And how does the distribution of activity over time and over space impact these choices? Can they assume that users act randomly, or are there significant correlations between their activities?

2.3.1 Trade-off: Dynamic or Static Allocation

Consider a scenario where a neuron is adaptive – namely, the neurons sample at f_{Nyq} when the signal is changing, and sample at f_{Nyq}/Z when the signal is static, where Z is a prespecified attenuation factor. Let the probability that a given neuron samples at f_{Nyq} be a : That is, a is the active fraction of the population. Then, each quantizer generates bits at the rate

$$f_{\text{bits}} = f_{Nyq}(a + (1 - a)/Z)\log_2 N; \quad (2.1)$$

because for the fraction a of the time, it samples at f_{Nyq} ; the remaining fraction $(1 - a)$ of the time, it samples at f_{Nyq}/Z . Furthermore, $\log_2 N$ bits are used to encode the neuron's location for AER, where N is the number of neurons.

On the other hand, we may use conventional quantizers that sample every location at f_{Nyq} and do not locally adapt their sampling rate. In that case, there is no need to encode location explicitly. We simply poll all N locations, according to a fixed sequence, and infer the origin of each sample from its temporal location. This is similar to scanning all the neurons. As the sampling rate is constant, the bit-rate per quantizer is simply f_{Nyq} . The multiple bits required to encode identity are offset by the reduced sampling rates produced by local adaptation when activity is sparse. In fact, adaptive sampling produces a lower bit rate than fixed sampling if

$$a < (Z/(Z - 1))(1/\log_2 N - 1/Z). \quad (2.2)$$

For example, in a 64×64 array of neurons with sampling rate attenuation $Z = 40$, the active fraction, a , must be less than 6.1%.

It may be more important to minimize the number of samples produced per second instead of minimizing the bit rate as there are usually sufficient I/O pins to transmit all the address bits in parallel. In that case, it is the number of samples per second that is fixed by the channel capacity. Given a certain fixed throughput F_{ch} , in samples per second, we may compare the effective sampling rates, f_{Nyq} , achieved by various sampling strategies. Adaptive neurons allocate channel throughput dynamically in the ratio $a:(1-a) = Z$ between active and passive fractions of the population. Hence

$$f_{Nyq} = f_{ch}/(a + (1 - a)/Z), \quad (2.3)$$

where $f_{ch} \equiv F_{ch}/N$ is the throughput per neuron. The average neuronal ensemble size determines the active fraction, a , and frequency adaptation and synchronicity determine the attenuation factor, Z , assuming neurons that are not part of the ensemble have adapted. Figure 2.10

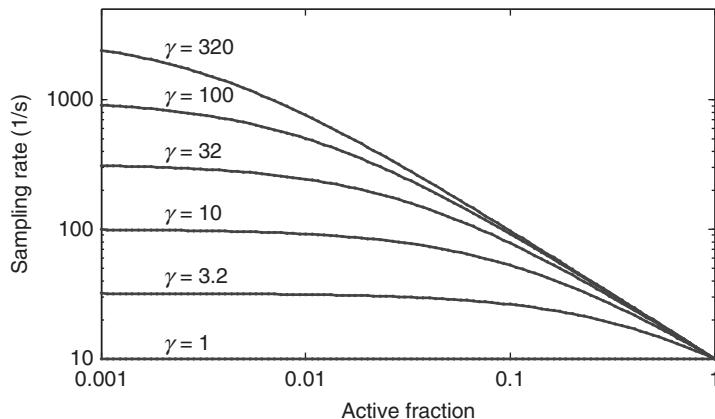


Figure 2.10 Effective Nyquist sampling rate versus active fraction plotted for various frequency adaptation factors (γ), with throughput fixed at 10 spikes/s/neuron. As the active fraction increases, the channel capacity must be shared by a larger number of neurons, and hence, the sampling rate decreases. It falls precipitously when the active fraction equals the reciprocal of the adaptation factor. © 2000 IEEE. Reprinted, with permission, from Boahen (2000)

shows how the sampling rate changes with the active fraction for various frequency adaptation factors, $Z = \gamma$. For small a and $Z > 1/a$, the sampling rate may be increased by a factor of at least $1/2a$.

2.3.2 Trade-off: Arbitered Access or Collisions?

Contention occurs if two or more neurons attempt to transmit simultaneously when we provide random access to the shared communication channel. We can simply detect and discard samples corrupted by collision (Mortara et al. 1995). Or we can introduce an arbiter to resolve contention and a queue to hold waiting neurons. Unfettered access shortens the cycle time, but collisions increase rapidly as the load increases, whereas arbitration lengthens the cycle time, reducing the channel capacity, and queuing causes temporal dispersion, degrading timing information.

Assuming the spiking neurons are described by independent, identically distributed, Poisson point processes, the probability of k spikes being generated during a single communication cycle is given by

$$P(k, G) = G^k e^{-G} / k!, \quad (2.4)$$

where G is the expected number of spikes. $G = T_{\text{ch}}/T_{\text{spk}}$, where T_{ch} is the cycle time and T_{spk} is the mean interval between spikes. By substituting $1/F_{\text{ch}}$ for T_{ch} , where F_{ch} is the channel capacity, and $1/(Nf_v)$ for T_{spk} , where f_v is the mean spike rate per neuron and N is the number of neurons, we find that $G = Nf_v = F_{\text{ch}}$. Hence, G is equal to the offered load.

We may derive an expression for the collision probability, a well-known result from communications theory, using the probability distribution $P(k, G)$. To transmit a spike without a collision, the previous spike must occur at least T_{ch} seconds earlier, and the next spike must occur at least T_{ch} seconds later. Hence, spikes are forbidden in a $2T_{\text{ch}}$ time interval, centered around the time that transmission starts. Therefore, the probability of the spike making it through is $P(0, 2G) = e^{-2G}$, and the probability of a collision is

$$p_{\text{col}} = 1 - P(0, 2G) = 1 - e^{-2G}. \quad (2.5)$$

The unfettered channel must operate at high error rates to maximize channel utilization. The throughput is $S = Ge^{-2G}$, since the probability of a successful transmission (i.e., no collision) is e^{-2G} . Throughput may be expressed in terms of the collision probability

$$S = \frac{1 - p_{\text{col}}}{2} \ln \left(\frac{1}{1 - p_{\text{col}}} \right). \quad (2.6)$$

This expression is plotted in Figure 2.11. The collision probability exceeds 0.1 when throughput reaches 5.3%. Indeed, the unfettered channel utilizes a maximum of only 18% of its capacity. Therefore, it offers higher transmission rates than the arbitered channel only if it is more than five times faster since, as we shall show next, the arbitered channel continues to operate in a useful regime at 95% capacity.

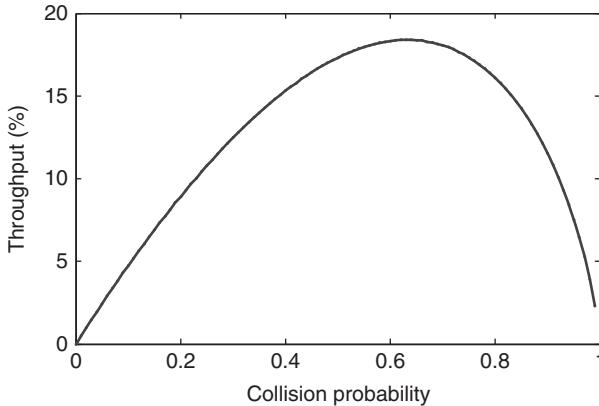


Figure 2.11 Throughput versus collision probability throughput attains a maximum value of 18% when the collision probability is 0.64, and the load is 50%. Increasing the load beyond this level lowers throughput because collisions increase more rapidly than the load does. © 2000 IEEE. Reprinted, with permission, from Boahen (2000)

2.3.3 Trade-off: Queueing versus Dropping Spikes

What about the timing errors introduced by queuing in the arbitrated channel? For an offered load of 95%, the collision probability is 0.85. Hence, collisions occur frequently and neurons are most likely to spend some time in the queue. By expressing these timing errors as percentages of the neuronal latency and temporal dispersion, we can quantify the trade-off between queuing new spikes, to avoid losing old spikes, versus dumping old spikes, to preserve the timing of new spikes.

To find the latency and temporal dispersion introduced by the queue, we use well-known results from queuing theory, which give moments of the waiting time, $\overline{w^n}$; as a function of moments of the service time, $\overline{x^n}$:

$$\overline{w} = \frac{\lambda \overline{x^2}}{2(1 - G)}; \overline{w^2} = 2\overline{w}^2 + \frac{\lambda \overline{x^3}}{3(1 - G)}; \quad (2.7)$$

where λ is the arrival rate of spikes. These results hold when spikes arrive according to a Poisson process. With $x = T_{ch}$ and $\lambda = G/T_{ch}$, the mean and the variance of the cycles spent waiting are given by

$$\overline{m} \equiv \frac{\overline{w}}{T_{ch}} = \frac{G}{2(1 - G)}; \sigma_m^2 \equiv \frac{\overline{w^2} - \overline{w}^2}{T_{ch}^2} = \overline{m}^2 + \frac{2}{3}\overline{m}. \quad (2.8)$$

We have assumed that the service time, x , always equals T_{ch} ; and therefore $\overline{x^n} = T_{ch}^n$.

We find that at 95% capacity, for example, a sample spends 9.5 cycles in the queue, on average. This result agrees with intuition: As every twentieth slot is empty, one must wait anywhere from 0 to 19 cycles to be serviced, which averages out to 9.5. Hence the latency is

10.5 cycles, including the additional cycle required for service. The standard deviation is 9.8 cycles – virtually equal to the latency. In general, this is the case whenever the latency is much more than one cycle, resulting in a Poisson-like distribution for the wait times. We can express the cycle time, T_{ch} , in terms of the neuronal latency, μ , by assuming that T_{ch} is short enough to transmit half the spikes in an ensemble in that time. That is, if the ensemble has N_e spikes and its latency is μ , the cycle time must satisfy $\mu/T_{\text{ch}} = (N_e/2)(1/G)$ since $1/G$ cycles are used to transmit each spike, on average, and half of them must be transmitted in μ seconds. Using this relationship, we can express the wait time as a fraction of the neuronal latency:

$$e_\mu = \frac{(\bar{m} + 1)T_{\text{ch}}}{\mu} = \frac{G}{N_e} \left(\frac{2 - G}{1 - G} \right). \quad (2.9)$$

The timing error is inversely proportional to the number of neurons because the channel capacity grows with population size. Therefore, the cycle time decreases, and there is a proportionate decrease in queuing time – even when the number of cycles spent queuing remains the same.

Conversely, given a timing error specification, we can invert our result to find out how heavily we can load the channel. The throughput, S , will be equal to the offered load, G , since every spike is transmitted eventually. Hence, the throughput is related to channel latency and population size by

$$S = N \left(\frac{e_\mu}{2} + \frac{1}{N_e} - \sqrt{\left(\frac{e_\mu}{2} \right)^2 + \frac{1}{N_e^2}} \right), \quad (2.10)$$

when the channel capacity grows linearly with the number of neurons. Figure 2.12 shows how the throughput changes with the channel latency. It approaches 100% for large timing errors and drops precipitously for low timing errors, going below 95% when the normalized error becomes less than $20/N_e$. As $N_e = aN$, the error is $400/N$ if the active fraction, a , is 5%. Therefore, the arbitrated channel can operate close to capacity with timing errors of a few % when the population size exceeds several tens of thousands.

2.3.4 Predicting Throughput Requirements

Given a neuron's firing rate immediately after a step change in its input, f_a , we can calculate the peak spike rate of active neurons and add the firing rate of passive neurons to obtain the maximum spike rate. Active neurons fire at a peak rate of ϵf_a , where ϵ is the synchronicity. And passive neurons fire at f_a/γ (assuming they have adapted), where γ is the frequency adaptation. Hence, we have

$$F_{\text{max}} = aN_e f_a + (1 - a)N f_a / \gamma, \quad (2.11)$$

where N is the total number of neurons and a is the active fraction of the population, which form a neuronal ensemble. We can express the maximum spike rate in terms of the neuronal latency by assuming that spikes from the ensemble arrive at the peak rate. In this case, all aN

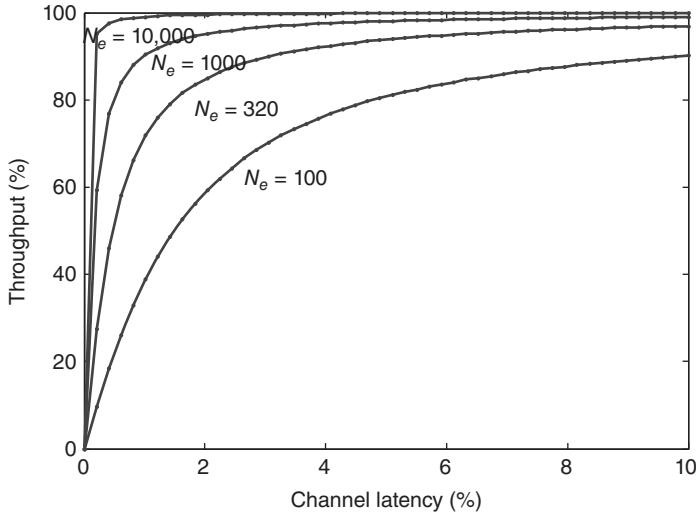


Figure 2.12 Throughput versus normalized channel latency plotted for different neuronal ensemble sizes (N_e). Higher throughput is achieved at the expense of latency because queue occupancy goes up as the load increases. These wait cycles become a smaller fraction of the neuronal latency as the population size increases, because cycle time decreases proportionately. © 2000 IEEE. Reprinted, with permission, from Boahen (2000)

neurons will spike in the time interval $1/(ef_a)$. Hence, the minimum latency is $\mu_{\min} = 1/(2ef_a)$. Thus, we can rewrite our expression for F_{\max} as

$$F_{\max} = \frac{N}{2\mu_{\min}} \left(a + \frac{1-a}{\epsilon\gamma} \right). \quad (2.12)$$

Intuitively, μ_{\min} is the neurons' timing precision and $N(a + (1-a)/(\epsilon\gamma))/2$ is the number of neurons that fire during this time. The throughput must be equal to F_{\max} , and there must be some surplus capacity to minimize collision rates in the unfettered channel and minimize queuing time in the arbitrated one. This overhead is over 455% (i.e., $(1 - 0.18)/0.18$) for the unfettered channel, but only 5.3% (i.e., $(1 - 0.95)/0.95$) for the arbitrated one.

In summary, arbitration is the best choice for neuromorphic systems whose activity is sparse in space and in time, because we trade an exponential increase in collisions for a linear increase in temporal dispersion. Furthermore, holding utilization constant (i.e., throughput expressed as a percentage of the channel capacity), temporal dispersion decreases as technology advances and we build larger networks with shorter cycle times, even though the collision probability remains the same. The downside of arbitration is that it takes up area and time, reducing the number of neurons that can be integrated onto a chip and the maximum rate at which they can fire. Several effective strategies for reducing the overhead imposed by arbitration have been developed; they are the subject of the next section.

2.3.5 Design Trade-offs

For a random-access, time-multiplexed channel, the multiple bits required to encode identity are offset by the reduced sampling rates produced by local adaptation when activity is sparse. The payoff is even better when there are sufficient I/O pins to transmit all the address bits in parallel. In this case, frequency and time-constant adaptation allocate bandwidth dynamically in the ratio $a: (1-a)/Z$ between active and passive fractions of the population. For low values of the active fraction, a , and sampling-rate attenuation factors, Z , larger than $1/a$, the effective Nyquist sampling rate may be increased by a factor of $1/2a$.

Contention occurs when two or more neurons spike simultaneously, and we must dump old spikes to preserve the timing of new spikes or queue new spikes to avoid losing old spikes. An unfettered design, which discards spikes clobbered by collisions, offers higher throughput if high spike loss rates are tolerable. In contrast, an arbitrated design, which makes neurons wait their turn, offers higher throughput when low spike loss rates are desired. Indeed, the unfettered channel utilizes only 18% of its capacity, at the most. Therefore, the arbitrated design offers more throughput if its cycle time is no more than five times longer than that of the unfettered channel.

The inefficiency of the unfettered channel design, also known as ALOHA, has been long recognized, and more efficient protocols have been developed (Schwartz 1987). One popular approach is CSMA (carrier sense, multiple access), where each user monitors the channel and does not transmit if it is busy. This channel is prone to collisions only during the time it takes to update its state. Hence, the collision rate drops if the round trip delay is much shorter than the packet-transmission time, as in bit-serial transmission of several bytes. Its performance is no better than ALOHA's, however, if the round trip delay is comparable to the packet-transmission time (Schwartz 1987), as in bit-parallel transmission of one or two bytes. Consequently, it is unlikely that CSMA will prove useful for neuromorphic systems (some preliminary results were reported in Abusland et al. 1996).

As technology improves and we build denser arrays with shorter cycle times, the unfettered channel's collision probability remains unchanged for the same normalized load, whereas the arbitrated channel's normalized timing error decreases. This desirable scaling arises because timing error is the product of the number of wait cycles and the cycle time. Consequently, queuing time decreases due to the shorter cycle times, even though the number of cycles spent waiting remains the same. Indeed, as the cycle time must be inversely proportional to the number of neurons, N , the normalized timing error is less than $400/N$ for loads below 95% capacity and active fractions above 5%. For population sizes of several tens of thousands, the timing errors make up just a few percentage points.

For neurons whose timing precision is much better than their inter-spike interval, we may estimate throughput requirements by measuring frequency adaptation and synchronicity. Frequency adaptation, γ , gives the spike rate for neurons that are not part of the neuronal ensemble. And synchronicity, ϵ , gives the peak spike rate for neurons in the ensemble. These firing rates are obtained from the spike frequency at stimulus onset by dividing by γ and multiplying by ϵ , respectively. The throughput must exceed the sum of these two rates if we wish to transmit the activity of the ensemble without adding latency or temporal dispersion. The surplus capacity must be at least 455% to account for collisions in the unfettered channel, but may be as low as 5.3% in the arbitrated channel, with subpercent timing errors due to queuing.

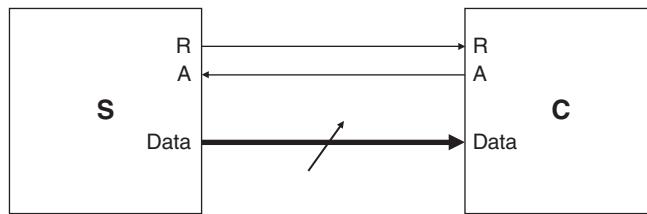


Figure 2.13 The system model, with sender **S**, receiver **C**, control signals request (**R**) and acknowledge (**A**), and implementation-dependent data wires

2.4 The Evolution of AER Links

Different neuromorphic chips or modules that communicate with each other using the AER representation must not only agree on the logical description of the protocol to communicate spikes, but also the physical and electrical protocol for communication. There are a number of standard approaches that have been used for this purpose, and we describe the evolution of various AER link implementations.

2.4.1 Single Sender, Single Receiver²

One of the earliest standards to emerge for point-to-point AER communication, that is, from a single sender to a single receiver, is dubbed ‘AER 0.02’ after the subtitle of the technical report in which it was first described (AER, 1993). This document described the logical and electrical requirements for signals used in AER communication.

AER 0.02 only concerns the point-to-point, unidirectional communication of asynchronous data from a sender **S** to a receiver **C**, as shown in Figure 2.13. The connection between **S** and **C** consists of two types of wires, control wires and data wires.

The data wires are exclusively driven by **S**, and exclusively sensed by **C**. The encoding used by the data wires is not the subject of AER 0.02; the number of wires, the number of states on each wire, and the number representation of the data wires are all considered implementation dependent. Only the validity of the data wires is specified by AER 0.02. If **S** is driving a stable high signal on the data wires, suitable for robust sensing by **C**, the data lines are considered valid; if this condition does not hold, the data lines are considered invalid. For those familiar with asynchronous communication protocols, AER 0.02 uses a *bundled-data protocol* for communication with a four-phase handshake on the wires **R** and **A**.

The control wires use a delay-insensitive protocol for communication. Figure 2.14 shows the control sequence, a four-phase handshake, that communicates data from **S** to **C**. Initially both **R** and **A** are logic zero, and the data wires are considered invalid. To begin a transaction, **S** first drives valid signals on the data wires, and then drives **R** to logic 1. The receiver detects that **R** is a logic one, and can now sample the data wires. The *bundling* timing constraint is that the data wires are stable at the receiver when the receiver detects that **R** is a logic one. Once **C** has received the data, it responds by setting the acknowledge wire **A** to a logic one. As

² A large part of the text in this section is adapted from AER (1993).

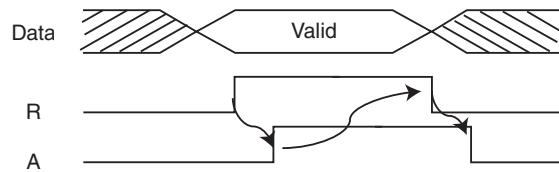


Figure 2.14 Single sender, single receiver, four-phase handshake. R is an active-high request signal; A is an active-high acknowledge signal

soon as the sender **S** detects this, the data wires no longer need to remain valid. The protocol concludes with **S** setting R to zero, in response to which **C** also resets the acknowledge wire A to zero and the entire transaction can repeat.

The minimum timing requirements in AER 0.02 are specified with an empirical test. A sender compatible with AER 0.02 must supply a free-running stream of signals on the data wires, if the R and A control wires of the sender are connected together. In addition, the signals on the data wires must be valid whenever R is at logic level 1. A receiver compatible with AER 0.02 must sense a stream of valid addresses, if an inverted version of the A signal of the receiver is connected to the R of the receiver, and a valid address is present on the data wires. It was expected that future versions of the standard might augment these empirical requirements with more traditional specifications.

If an implementation used voltages as the signal variable, AER 0.02 strongly suggested that, in the default mode of operation for an implementation, the logic 1 level for A and R is more positive than the logic 0 level. In addition, AER 0.02 strongly suggests implementations support multiple modes of signal polarity; ideally, the electrical polarity of A and R can be changed, either by physical contact (reconfiguring signals or jumpers) or under programmed control. AER 0.02 was a minimal communications standard, and it was expected that research groups would augment the standard to add new functionality, and it was expected that future versions of the standard would largely be the codification of enhancements that had been successfully implemented.

Guidelines for extensions to AER 0.02 were also prescribed in that it was stated that in an extended AER 0.02 implementation, it must be possible to nondestructively disable all extensions, for example, through reconfiguring switches or jumpers, or via programmed control. In this ‘backward-compatibility’ mode, an implementation must supply A and R control signals and data wires that perform as required by AER 0.02.

In practice, the AER 0.02 standard was so loosely defined that it was of limited utility. The basic four-phase handshake with request and acknowledge lines became established. But AER 0.02 did not specify voltages, bus width, signal polarities, traditional signal setup and hold times, or any kind of connector standard. Indeed AER 0.02 implementations with nontraditional electrical signaling methods were encouraged! This all meant that any two devices, both of which conformed to AER 0.02, were likely to be incompatible, particularly if they were developed in different labs. In order for two such devices to work together, there would normally need to be some glue logic device built to connect them, and this would introduce new possible points of failure due to anything from timing violations to bad solder joints. Building general purpose interfaces that satisfied the AER 0.02 ideal of supporting either electrical polarity for A and R introduced further complexity to the design and configuration.

Although never codified as AER 0.03 (or some higher revision number), more tightly defined de facto standards did emerge. Voltages were defined to be 0 V/5 V TTL, later 3.3 V, the bus width was 16 bits, data lines were positive logic; request and acknowledge lines negative logic, and insulation displacement connectors of various widths became standard (cf. Dante 2004; Deiss et al. 1999; Häfliger 2003). This made it easier to construct multichip systems more reliably, for example, the system described in Serrano-Gotarredona et al. (2009), which we discuss in Section 13.3.3.

Timing issues often remained because the senders violated the requirement that the signals on the data wires must be valid whenever the request signal is at logic level 1. The request signal was often driven by the same circuitry that drove the data wires and changed state at the same time as the data wires. In such a system there is no guarantee that the data seen by a receiver will be valid when the receiver sees the request signal.

2.4.2 Multiple Senders, Multiple Receivers

A possible idealized design for an address-event system consists of multiple address-event sending and receiving blocks that place addresses on an address-event bus to send, and simply monitor the bus to listen for addresses which they are interested in to receive those addresses when they occur on the bus.

This design uses *source* addresses on its bus. As a fan-out factor of 10 to 10,000 might be expected in neural networks, that is, one source address might be of interest to up to $O(10^4)$ destinations, using source addresses helps to keep down the bandwidth requirements. The idealized address-event receiver blocks are expected to perform the required fan-out and conversion to *destination* addresses internally.

A design with multiple senders and receivers (not all senders need also be receivers and *vice versa*) on a single bus needs a somewhat different protocol to that used in the point-to-point case described in Section 2.4.1 above. To avoid collisions on the shared data lines, a sender may not drive them at the same time as generating a request R but must wait until it has received the acknowledge signal A. This is illustrated in Figure 2.15.

For the SCX project (Deiss et al. 1999) a multiple sender, multiple receiver protocol was defined (illustrated in Figure 2.16) in which each device i connected to its AE bus has its own dedicated pair of request (\overline{REQ}_i) and acknowledge (\overline{ACK}_i) lines. In addition, a collection of shared data lines are used. A device that wishes to transmit a spike on the bus asserts its request line (de-asserts \overline{REQ}_i). A centralized bus arbiter monitors all the request lines, and selects one of the devices requesting bus access by asserting the corresponding acknowledge line (de-asserting \overline{ACK}_i). When a requesting device detects that it has been selected it drives

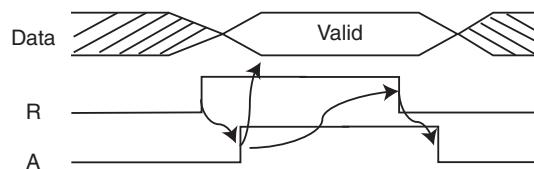


Figure 2.15 Multiple sender, multiple receiver four-phase handshake

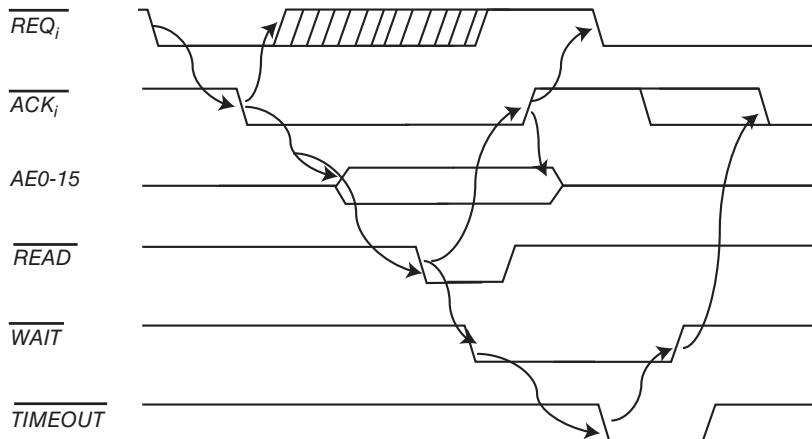


Figure 2.16 SCX project multiple sender, multiple receiver protocol. Adapted from Deiss (1994). Reproduced with permission of Stephen R. Deiss, ANDt (Applied Neurodynamics)

data on the shared bus, after which it de-asserts its request line (i.e., asserts \overline{REQ}_i). At this point all devices can listen on the bus and determine if the spike on the bus should be received locally. To simplify this process, the bus arbiter generates a global \overline{READ} signal. A certain amount of time is budgeted for receiving devices to accept the data off the shared wires, and after this time has elapsed the \overline{ACK}_i is returned to its initial state thereby permitting the next bus transaction.

Passive listener devices may exist on the bus that do not generate any request signals. These devices monitor the AE activity on the bus by latching the address bits on either edge of the of the \overline{READ} signal. Signals called \overline{WAIT} and $\overline{TIMEOUT}$ were specified to implement a mechanism whereby a slow acting device may delay the bus arbiter from continuing with a further bus cycle (by granting an acknowledge to some other device) for up to 1 μ s. To cause such a delay, a device might assert \overline{WAIT} (low) after the low going edge of \overline{READ} , and de-assert it either when sufficient delay for its purposes has elapsed, or in any event when a low going edge occurs on $\overline{TIMEOUT}$. The $\overline{TIMEOUT}$ signal would be driven low by the bus arbiter if \overline{WAIT} were still low 1 μ s after being first asserted. The use of this \overline{WAIT} and $\overline{TIMEOUT}$ feature was however discouraged, and no device ever made use of it.

2.4.3 Parallel Signal Protocol

The Single Sender, Single Receiver (Mahowald 1992; Sivilotti 1991) and Multiple Sender, Multiple Receiver (Deiss et al. 1999) protocols described above employ (or at least imply) straightforward parallel buses. The SCX project adopted this approach, defining an optional connector standard for peripheral AE devices based on 26-way ribbon cable and IDC connectors (Baker and Whatley 1997). The first 16 pins carry the signals AE0 to AE15 in order, then come two ground pins followed by the request signal on pin 19 and the acknowledge signal on pin 20. This arrangement of the signals on the first 20 pins has been followed by several projects

since then, including those using only the single sender, single receiver model, although the remaining 6 pins have been dropped so that only 20-pin cables and connectors are required.

The CAVIAR project decided to use an easy ‘off-the-shelf’ solution for their cable and connector that was widely used for what at that time were considered fast digital signals, namely the ATA/133 standard normally used at that time to connect storage devices to motherboards inside PCs (Häfliger 2003). This standard uses a special 40-pin IDC connector with an 80-way ribbon cable. The connector remains compatible with older 40-pin ATA headers, but has internal wiring that routes the signals onto the 80-way ribbon cable in such a way that the signal wires are well interspersed with ground wires. Using this system provided a cheap, readily available interconnection solution with good electrical characteristics, but the pinout of the connectors is not compatible with the simple 20-pin layout described above. The signals are arranged for compatibility with the ATA/133 standard with AE0 to AE7 on pins 17 to 3 and AE8 to AE15 on pins 4 to 18; the request signal appears on pin 21 and the acknowledge on pin 29.

CAVIAR also defined the use of variants of the single-sender/single-receiver and single-sender/multiple-receiver protocols described above, which used 3.3 V signals and in which the various transition-to-transition bus timings were specified in the traditional manner, albeit with some of these timings being specified with a minimum of 0 and a maximum of infinity.

2.4.4 Word-Serial Addressing

Many address-event senders and receivers, particularly senders like so-called retina chips, are constructed as 2D arrays of address-event sender elements, for example, pixels. In this case, the address space used by the device is usually organized such that a certain number of address bits are used to indicate the row and a certain number of address bits are used to indicate the column in which an AE is generated. (For example a 128×128 pixel square array might produce AEs with addresses of the form $y_6y_5y_4y_3y_2y_1y_0x_6x_5x_4x_3x_2x_1x_0$, where each y_i and x_i represent a binary digit.) For every doubling in the size of the array, a further address bit must be generated and transmitted. This may be highly disadvantageous for a chip design: as more wires must be routed across the chip, there may remain less area to do useful computation (the fill factor becomes worse); also more pads, which are usually a scarce resource, are required to communicate to the outside world. It may also become highly inconvenient when the associated AEs enter the realm of conventional digital electronics and software where it is expected that data words have widths of a power of two, for example, an 18-bit AE does not fit conveniently into a 16-bit word.

One way to alleviate this problem would be to always transmit one AE over the course of two separate parts of a bus cycle; in our example above, first the y (row) address and then the x (column) address could be transmitted over a 7-bit bus. This would, however, be to the detriment of the time required to transmit the AE and the utilization of the available bus bandwidth.

As devices become larger, and also as the frequency with which the individual elements (pixels) wish to send events gets higher, the more likely it becomes that multiple elements on one row are waiting to send an event on the bus simultaneously. This can be exploited by servicing all waiting elements on one row transmitting the y (row) address just once followed by the x (column) addresses of all of the waiting elements in that row before moving on to

another row. Such a group of one y address and one or more x addresses forms a burst in a similar way that bursts are used on memory buses. In a memory bus burst there is typically one address value followed by multiple data values from consecutive memory locations; in an AE bus burst, however, all words transmitted are partial addresses, and not necessarily from adjacent locations, merely from the same row. One extra signaling wire is required to distinguish the sending of an x address from the sending of a y address or equivalently to signal the beginning and end of a burst. This scheme, known as word-serial, achieves a more efficient utilization of the available bus bandwidth (Boahen 2004a, 2004b, 2004c).

2.4.5 Serial Differential Signaling³

With the speeds that AER chips and systems have recently reached, the parallel (including so-called Word-Serial) AER approach in board-to-board communication has become a limiting factor at the system level. With the frequencies on parallel AER in the order of tens of megahertz, the wavelength of those frequencies has shrunk to about the order of magnitude of the lengths involved in the experimental setups, or only slightly larger. One rule of thumb in electrical engineering says that if the signal wavelength is not at least one to two orders of magnitude greater than the physical size of the system, then the radio frequency (RF) properties of the signals have to be taken into account: wires can no longer be assumed to be perfect conductors with the same potential at every point, but have to be treated as transmission lines. If these problems are not taken into account, issues such as RF sensitivity, cross talk and ground bounce arise, especially in parallel AER links using ribbon cables. These issues can best be solved by resorting to serial differential signaling.

These issues with the parallel approach have also played a major role in industrial and consumer electronics in general. The solution has been to use even faster, but differential links, and to carefully control the line impedance at every point between the sender and receiver. In such a differential signaling scheme there is always a pair of wires that carry signals of opposite sense. The absolute value of the voltages on the signal wires does not have any meaning, only the voltage difference between the two wires of the pair has. These so called differential pairs are then usually shielded, thus avoiding the problems of RF sensitivity and cross talk to other signal wires. Because of the differential signaling, the ground bounce problem is also solved. A differential driver always pushes as much charge into one wire as it pulls from the other. Thus the net charge flow is always zero.

Parallel to Serial and Serial to Parallel

The data rates that can be achieved using differential signaling are orders of magnitude higher than with traditional single-ended signaling. Therefore less (but better) wires are nowadays used to achieve the same or better bandwidth than with the many parallel wires in traditional bus links.

For example IDE / parallel ATA can achieve up to 1 Gbps using 16 single-ended data signals, but only in one direction at a time (half-duplex). Serial ATA (SATA) has 2 differential pairs

³ A large part of the text in this section is © 2008 IEEE. Reprinted, with permission, from Fasnacht et al. (2008).

(and thus four signal wires), one pair to send, and one to receive (SATA n.d.). Each pair can transmit up to 3 Gbps.

An AER communication infrastructure using serial differential signaling over SATA cables for inter-board communication was implemented by Fasnacht et al. (2008) following an approach similar to the one proposed in (Berge and Häfliger 2007).

In order to take advantage of the low latency and high bandwidth available by using such high-speed serial links, the notion of using a handshake must be abandoned, as there is no time for an acknowledge signal to return to a sender before the next AE can be sent. It is, however, possible and advantageous to implement a back channel for a flow control signal; see Section 13.2.5 and Fasnacht et al. (2008) for details.

2.5 Discussion

We have seen that communication in neuromorphic systems is typically implemented with a form of time-multiplexed, packet-switched communication known as AER, in which *time represents itself*. This communication relies on the observations that neurons operate orders of magnitude slower than digital electronics, and that the small amounts of delay and jitter imposed by passing spikes through digital electronics are negligible in terms of the time constants in neurons.

AER communication requires multiplexing and encoder circuits at the sender, almost invariably some form of arbitration, and demultiplexing or decoding circuits at the receiver. We return to look at the design of these on-chip circuits in Chapter 12. As soon as there are multiple endpoints possible in a system, the issues of address mapping and routing arise. These issues are addressed further in Chapters 13 and 16.

We have looked at how in designing an AER link, there are five performance criteria to consider: capacity, throughput, latency, integrity, and dispersion. We have seen how these criteria are interrelated, and that there are trade-offs to be made between various features of a link. At the end of the Introduction to this chapter (Section 2.1) and the beginning of Section 2.2 we made the observation that the relevant timescales for the operation of neurons are very slow in comparison to the speed of the electronic circuits used to implement communication links. Formalizing these observations leads to the conclusion that arbitrated channels are much more efficient for AER communication than unfettered (nonarbitrated) ones.

We have also seen how the electrical and physical characteristics of communication links must be defined and preferably standardized in order to achieve practical communication between AER-based devices.

Parallel AER generally uses a form of four-phase handshake. With the move to faster serial differential signaling, handshaking must be abandoned, since there is no time for an acknowledge signal to be returned to the sender.

Over time there has been an increasing trend to use off-the-shelf interconnection technology, from the original simple use of IDC (Insulation Displacement Connectors), through the use of the then readily available ATA/133 cables, to the use of SATA cables in newer serial links.

The hardware infrastructure for implementing AER communication between AER chips will be examined further in Chapter 13. Before then, the intervening chapters discuss the individual building blocks of neuromorphic systems.

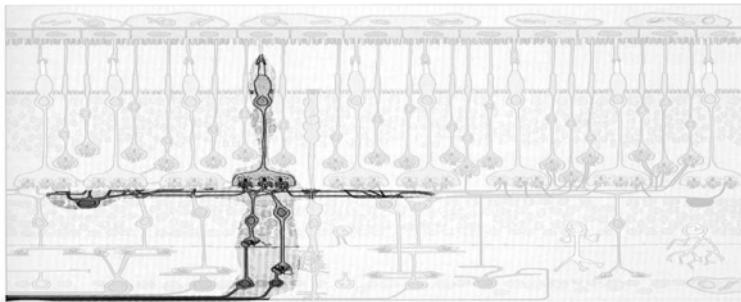
References

- Abusland AA, Lande TS, and Høvin M. 1996. A VLSI communication architecture for stochastically pulse-encoded analog signals. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* III, pp. 401–404.
- AER. 1993. The address-event representation communication protocol [sic], AER 0.02.
- Baker B and Whatley AM. 1997. Silicon cortex daughter board 1, <http://www.ini.uzh.ch/amw/scx/daughter.htmlSCXDB1> (accessed July 28, 2014)
- Berge HKO and Häfliger P. 2007. High-speed serial AER on FPGA. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 857–860.
- Binzegger T, Douglas RJ, and Martin KAC. 2004. A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci.* **24**(39), 8441–8453.
- Boahen KA. 2000. Point-to-point connectivity between neuromorphic chips using address-events. *IEEE Trans. Circuits and Syst. II* **47**(5), 416–434.
- Boahen KA. 2004a. A burst-mode word-serial address-event link—I: transmitter design. *IEEE Trans. Circuits Syst. I, Reg. Papers* **51**(7), 1269–1280.
- Boahen KA. 2004b. A burst-mode word-serial address-event link—II: receiver design. *IEEE Trans. Circuits Syst. I, Reg. Papers* **51**(7), 1281–1291.
- Boahen KA. 2004c. A burst-mode word-serial address-event link—III: analysis and test results. *IEEE Trans. Circuits Syst. I, Reg. Papers* **51**(7), 1292–1300.
- Braitenberg V and Schüz A. 1991. *Anatomy of the Cortex*. Springer, Berlin.
- Dally WJ and Towles B. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.
- Dante V. 2004. *PCI-AER Adapter board User Manual*, 1.1 edn. Istituto Superiore di Sanità, Rome, Italy, http://www.ini.uzh.ch/~amw/pcaer/user_manual.pdf (accessed July 28, 2014).
- Deiss SR. 1994. Address-event asynchronous local broadcast protocol. Applied Neurodynamics (ANDt), 062894 2e, <http://appliedneuro.com/> (July 28, 2014).
- Deiss SR, Douglas RJ, and Whatley AM. 1999. A pulse-coded communications infrastructure for neuromorphic systems [Chapter 6]. In: *Pulsed Neural Networks* (eds Maass W and Bishop CM). MIT Press, Cambridge, MA, pp. 157–178.
- Fasnacht DB, Whatley AM, and Indiveri G. 2008. A serial communication infrastructure for multi-chip address event systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 648–651.
- Georgiou J and Andreou A. 2006. High-speed, address-encoding arbiter architecture. *Electron. Lett.* **42**(3), 170–171.
- Gerstner W and Kistler WM. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.
- Häfliger P. 2003. CAVIAR Hardware Interface Standards, Version 2.0, Deliverable D_WP7.1b, <http://www.imse-cnm.csic.es/caviar/download/ConsortiumStandards.pdf> (accessed July 28, 2014).
- IEEE 802.3 working group. 1985. *IEEE Std 802.3-1985, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. IEEE Computer Society, New York.
- Imam N and Manohar R. 2011. Address-event communication using token-ring mutual exclusion. *Proc. 17th IEEE Int. Symp. on Asynchronous Circuits and Syst. (ASYNC)*, pp. 99–108.
- Lazzaro J, Wawrzynek J, Mahowald M, Sivilotti M, and Gillespie D. 1993. Silicon auditory processors as computer peripherals. *IEEE Trans. Neural Netw.* **4**(3), 523–528.
- Lazzaro JP and Wawrzynek J. 1995. A multi-sender asynchronous extension to the address-event protocol. In: *Proceedings of the 16th Conference on Advanced Research in VLSI* (eds Dally WJ, Poulton JW, and Ishii AT). IEEE Computer Society, pp. 158–169.
- Lin J, Merolla P, Arthur J, and Boahen K. 2006. Programmable connections in neuromorphic grids. *Proc. 49th IEEE Int. Midwest Symp. Circuits Syst.* **1**, pp. 80–84.
- Mahowald M. 1992. VLSI analogs of neural visual processing: a synthesis of form and function. PhD thesis. California Institute of Technology, Pasadena, CA.
- Mahowald M. 1994. *An Analog VLSI System for Stereoscopic Vision*. Kluwer Academic, Boston, MA.
- Mortara A and Vittoz EA. 1994. A communication architecture tailored for analog VLSI artificial neural networks: intrinsic performance and limitations. *IEEE Trans. Neural Netw.* **5**(3), 459–466.
- Mortara A, Vittoz EA, and Venier P. 1995. A communication scheme for analog VLSI perceptive systems. *IEEE J. Solid-State Circuits* **30**(6), 660–669.
- SATA. n.d. SATA – Serial ATA, <http://www.sata-io.org/> (accessed July 28, 2014).

- Schwartz M. 1987. *Telecommunication Networks: Protocols, Modeling, and Analysis*. Addison-Wesley, Reading, MA.
- Serrano-Gotarredona R, Oster M, Lichtsteiner P, Linares-Barranco A, Paz-Vicente R, Gomez-Rodriguez F, Camunas-Mesa L, Berner R, Rivas M, Delbrück T, Liu SC, Douglas R, Häfliger P, Jimenez-Moreno G, Civit A, Serrano-Gotarredona T, Acosta-Jimenez A, and Linares-Barranco B. 2009. CAVIAR: A 45 K-neuron, 5 M-synapse, 12 G-connects/sec AER hardware sensory-processing-learning-actuating system for high speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* **20**(9), 1417–1438.
- Sivilotti M. 1991. Wiring considerations in analog VLSI systems with application to field-programmable networks. PhD thesis. California Institute of Technology Pasadena, CA.

3

Silicon Retinas



This chapter introduces biological and silicon retinas, focusing on recently developed silicon retina vision sensors with an asynchronous address-event output. The first part of the chapter introduces biological retinas and four examples of Address-Event Representation (AER) retinas. The second part of the chapter discusses the details of some of the pixel designs and the specifications of these sensors. The discussion focuses on future goals for improvements.

3.1 Introduction

Chapter 2 discussed the event-based communication architectures for neuromorphic systems, and this chapter focuses on state-of-the-art electronic models of biological retinas that use this communication fabric to transmit their output data. Electronically emulating the retina has been a major target of neuromorphic electronic engineers since the field's earliest times. When

The figure shows a cross section of mammalian retina, with input photoreceptors at top and output ganglion cells axons exiting at bottom left. The highlighted parts show the prototypical path from photoreceptors to ON and OFF bipolar cells (also involving the lateral horizontal cell) to ON and OFF ganglion cells. Rodieck (1988). Reproduced with permission of Sinauer Associates.

Fukushima et al. (1970) demonstrated his discrete model of the retina, it generated excitement although the practical industrial impact was minuscule. Nonetheless it probably inspired an entire generation of young Japanese engineers to work on cameras and vision sensors, which was evident in a Japanese domination of the electronic imager market which lasted for several decades. Likewise, Mahowald and Mead (1991) on the cover of *Scientific American* was an inspiration to a generation of American neuromorphic electronic engineers despite having raw performance capabilities that barely enabled the recognizable imaging of a cat. It has taken more than 20 years of effort to go from these early lab prototypes to devices that can now be commercially purchased and used in applications. This chapter is about the current state of electronic retinas, with a focus on event-based silicon retinas. It starts with a brief synopsis of biological retinal function and then reviews some existing silicon retina vision sensors. The chapter concludes with an extensive discussion of silicon retina performance specifications and their measurements.

3.2 Biological Retinas

As illustrated in Figure 3.1, the retina is a complex structure with three primary layers: the photoreceptor layer, the outer plexiform layer, and the inner plexiform layer (Kuffler 1953; Masland 2001; Rodieck 1988; Werblin and Dowling 1969). The photoreceptor layer consists of two classes of cells: cones and rods, which transform the incoming light into an electrical signal that affects neurotransmitter release in the photoreceptor output synapses. The photoreceptor cells in turn drive horizontal cells and bipolar cells in the outer plexiform layer. The two major classes of bipolar cells, the ON bipolar cells and OFF bipolar cells, separately code for bright spatiotemporal contrast and dark spatiotemporal contrast changes. They do this by comparing

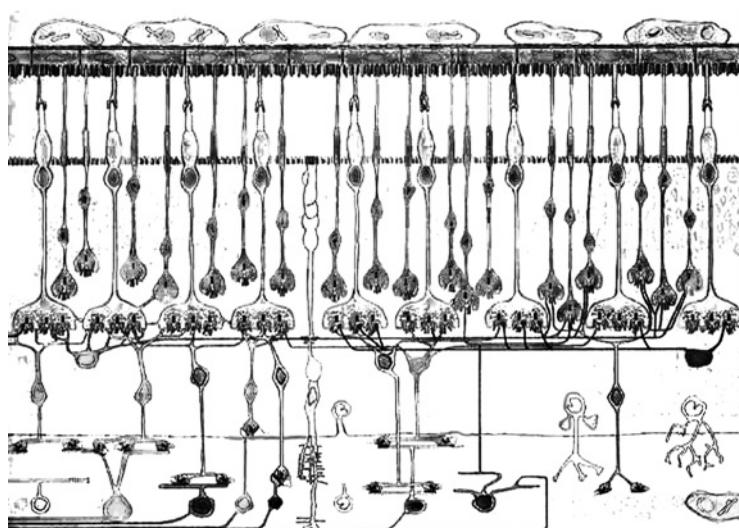


Figure 3.1 Cross section of the retina. Adapted from Rodieck (1988). Reprinted with permission of Sinauer Associates

the photoreceptor signals to spatiotemporal averages computed by the laterally connected layer of horizontal cells, which form a resistive mesh. The horizontal cells are connected to each other by conductive pores called gap junctions. Together with the input current produced at the photoreceptor synapses, this network computes spatiotemporal low-passed copies of the photoreceptor outputs. The horizontal cells and bipolar cell terminals come together in complex synapses at the rod and cone pedicles of photoreceptors. There the bipolar cells are effectively driven by differences between the photoreceptor and horizontal cell outputs, and in turn feed back onto the photoreceptors. In the even more complex outer plexiform layer, the ON and OFF bipolar cells synapse onto many types of amacrine cells and many types of ON and OFF ganglion cells in the inner plexiform layer. The horizontal and the amacrine cells mediate the signal transmission process between the photoreceptors and the bipolar cells, and the bipolar cells and the ganglion cells, respectively.

The bipolar and ganglion cells can be further divided into two different groups consisting of cells with more sustained responses and cells with more transient responses. These cells carry information along at least two parallel pathways in the retina, the magnocellular pathway, where cells are sensitive to temporal changes in the scene, and the parvocellular pathway where cells are sensitive to forms in the scene. This partition into sustained and transient pathways is too simplistic; in reality, there are many parallel pathways computing many views (probably at least 50 in the mammalian retina) of the visual input. However, for the purpose of this chapter, which is to discuss developments of silicon models of retinal processing, a simplified view of biological vision is sufficient.

Biological retinas have many desirable characteristics that are lacking in conventional silicon imagers, of which we mention two characteristics that have been incorporated in silicon retinas. First, eyes operate over a wide dynamic range (DR) of light intensity, allowing vision in lighting conditions over nine decades, from starlight to bright sunlight. Second, the cells in the outer plexiform and inner plexiform layers code for spatiotemporal contrast, thus removing redundant information and allowing the cells to encode the signals within their limited DR (Barlow 1961).

While conventional imagers are well suited to produce sequences of pictures, they do not have these biological properties of local gain control and redundancy reduction. Conventional imagers are constrained by economics and the infrastructure of conventional imaging technology. Economics has dictated that small pixels are cheaper than large ones, and that camera outputs should consist of a stroboscopic sequence of static pictures. The responses of the pixels are read out serially, leading to the generation of frames on which almost all of machine vision has been historically based. By contrast, biological vision does not depend on frames, so the availability of activity-driven, event-based silicon retinas can help to instigate research in new algorithms that are not dependent on frame information. These algorithms have advantages for fast, low-power vision systems.

3.3 Silicon Retinas with Serial Analog Output

It is interesting to see the historical evolution of silicon retinas in the diagram shown in Figure 3.19 (page 66). All the early retina designs had a serial output like conventional imagers; retina output values were repetitively sampled – or ‘scanned’ – to produce at the output a ‘picture’ of the activity. The first silicon VLSI retina implemented a model of the photoreceptor

cells, horizontal cells, and bipolar cells (Mahowald and Mead 1991). (Historically, an earlier electronic retina model by Fukushima et al. (1970) used discrete elements but in this chapter we focus on VLSI retinas.) Each silicon photoreceptor mimics a cone cell and contains both a continuous-time photosensor and adaptive circuitry that adjusts its response to cope with changing light levels (Delbrück and Mead 1994). A network of MOS variable resistors (the H-Res circuit discussed in Mead 1989) mimics the horizontal cell layer, furnishing feedback based on the average amount of light striking nearby photoreceptors; the bipolar cell circuitry amplifies the difference between the signal from the photoreceptor and the local average and rectifies this amplified signal into ON and OFF outputs. The response of the resulting retinal circuit approximates the behavior of the human retina (Mahowald 1994; Mead 1989). The ‘Parvo–Magno’ retina implementation discussed in Section 3.5.3 included both sustained and transient bipolar and ganglion cells and also amacrine cells. This design still comes closest to capturing the various cell classes of biological retinas. Besides these two retinas, many designers have produced silicon retina designs which implement usually a specific retina functionality but considered various circuit approaches to reduce mismatch in the responses of the chips so that the retinas can be used in applications. For example, the scanned silicon retina system of Kameda and Yagi (2003) uses a frame-based active pixel sensor (APS) as the phototransduction stage because of the better matching in the outputs of the pixel responses.

3.4 Asynchronous Event-Based Pixel Output Versus Synchronous Frames

Because of the frame-based nature of the output from cameras in most electronic systems, almost all computer vision algorithms are based on image frames, that is, on static pictures or sequences of pictures. This is natural given that frame-based devices have been dominant from the earliest days of electronic imaging. Although frame-based sensors have the advantages of small pixels and compatibility with standard machine vision, they also have clear drawbacks: the pixels are sampled repetitively even if their values are unchanged; short-latency vision problems require high frame rate and produce massive output data; DR is limited by the identical pixel gain, the finite pixel capacity for integrated photocharge, and the identical integration time. The high computational cost of machine vision with conventional imagers is largely the practical reason for the recent development of AER silicon retinas. These sensors are discussed next.

3.5 AER Retinas

The asynchronous nature of AER output is inspired by the way in which neurons communicate over long range. Rather than sampling pixel values, AER retina pixels asynchronously output address events when they detect a significant signal. The definition of ‘significant’ depends on the sensor, but in general, these events should be much sparser for typical visual input than repetitive samples of analog value. In other words, good retina pixels autonomously determine their own region of interest (ROI), a job that is left to the system level for conventional image sensors with ROI readout control.

The design and use of these AER vision sensors is still relatively novel. Even though the first integrated AER retina was built in 1992, there was not much progress in usable

AER vision sensors until recently. The main obstacles to advancement were the unfamiliarity with asynchronous logic and poor uniformity of pixel response characteristics. Industry is unfamiliar with frame-free vision sensors and understandably wary of large pixels with small fill factors. In recent years, the technology has progressed much faster mostly due to the increasing number of people working in this area. Recent AER retinas typically implement only one or two of the retina cell types so as to keep the fill factor still reasonably large and to have a low pixel array response variance. Table 3.1 compares the AER silicon retinas discussed here, along with a conventional CMOS camera for machine vision and the human eye. From even a cursory inspection we can see that cameras lag far behind the human eye, and that silicon retinas have far fewer pixels and consume much more power per pixel than a standard CMOS camera. Of course this low power consumption of a conventional camera leaves out the processing costs. If the silicon retina reduces redundancy or allows the vision processor to run at lower speed or bandwidth, then system-level power consumption can be greatly reduced by using it. In any case, we can still attempt to place existing sensors in classes in order to guide our thinking. AER retinas can be broadly divided into the following classes as listed in the row labeled ‘Class’ in the table:

- Spatial contrast (SC) sensors reduce spatial redundancy based on intensity ratios versus spatial difference sensors which use intensity differences. SC sensors are more useful for analyzing static scene content for the purpose of feature extraction and object classification.
- Temporal contrast (TC) sensors reduce temporal redundancy based on relative intensity change versus temporal difference sensors which use absolute intensity change. TC sensors are more useful for dynamic scenes with nonuniform scene illumination, for applications such as object tracking and navigation.

The exposure readout and pixel reset mechanisms can also be divided broadly into two classes:

- Frame event (FE) sensors which use synchronous exposure of all pixels and scheduled event readout.
- Asynchronous event (AE) sensors whose pixels continuously generate events based on a local decision about immediate relevance.

The next sections discuss examples of four different silicon retinas: the dynamic vision sensor (DVS) (Lichtsteiner et al. 2008), the asynchronous time-based image sensor (ATIS), the Magno–Parvo spatial and TC retina (Zaghlool and Boahen 2004a, 2004b), the biomimetic Octopus image sensor (Culurciello et al. 2003), and the SC and orientation sensor (ViSe) (Ruedi et al. 2003). Each of these vision sensors were designed with particular aims.

3.5.1 Dynamic Vision Sensor

The DVS responds asynchronously to relative temporal changes in intensity (Lichtsteiner et al. 2008). The sensor outputs an asynchronous stream of pixel address-events (AEs) that encode scene reflectance changes, thus reducing data redundancy while preserving precise timing information. These properties are achieved by modeling three key properties of

Table 3.1 Comparison of AER vision sensors with each other and with a conventional CMOS camera and the human eye. The CMOS camera example is of ‘global shutter’ type where all pixels are exposed synchronously, which is preferred for machine vision applications

Type	Parvo-Magno (Section 1.5.3)	VISe (Section 1.5.5)	DVS (Section 1.5.1)	ATIS (Section 1.5.2)	CMOS camera	Human eye
Year	2001	2003	2006	2010	2012	100 k BCE
Functionality	Asynch. spatial and temporal contrast	Frame-based spatial contrast and gradient direction, ordered output	Async. temporal contrast dynamic vision sensor	Machine vision; synchronous global shutter operation		>50 classes of cells
Class	SC TC AE	SC FE	TC AE	TC AE	Image sequence	Mammalian retina
Gray picture output	N	N	N	Y	Y	NA
Pixel size μm	34 \times 40	69 \times 69	40 \times 40	30 \times 30	5 \times 5	1 \times 1
Fill factor (%)	14%	9%	8.1%	10% (TC)/20% (gray)	60%	3% QE
Fabrication process	0.35 μm 4 M 2 P	0.5 μm 3 M 2 P	0.35 μm 4 M 2 P	180 nm 6 M 1 P MIM	0.13 μm	self-assembling
Pixel complexity	38 T	>50 T, 1 C	26 T, 3 C	77 T, 4 C, 2 PD	4 T/S T	~200 neurons
Array size	96 \times 60	128 \times 128	128 \times 128	304 \times 240	4 M pixels	100 M pixels
Power/pixel	10 μW	18 μW	400 nW	1.3 μW	25 nW	30 pW
Dynamic range	~50 dB	120 dB	120 dB	125 dB @30 FPS	~60 dB	180 dB
Response latency, frames/s (fps), events/s (eps)	~10 Meps	<2 ms 60 to 500 fps	15 us @1 klux	3.2 μs @1 klux 30 Meps peak, 6 Meps sustained	10 ms (100 FPS)	20 ms
FPN matching	1–2 decades	2% contrast	2.1% contrast		<0.3%	

See Section 3.7 for definitions of some of the specifications. (SC - spatial contrast, TC - temporal contrast, FE - frame events, AE - address events).

Source: Modified from Delbrück et al. (2010).

biological vision: sparse, event-based output, representation of relative luminance change (thus directly encoding scene reflectance change), and rectification of positive and negative signals into separate output channels. The DVS improves on prior frame-based temporal difference detection imagers (e.g., Mallik et al. 2005) by asynchronously responding to TC rather than absolute illumination, and on prior event-based imagers which do not reduce redundancy at all (Culurciello et al. 2003), reduce only spatial redundancy (Ruedi et al. 2003), have large fixed pattern noise (FPN), slow response, and limited DR (Zaghoul and Boahen 2004a), or have low contrast sensitivity (Lichtsteiner et al. 2004).

It would seem that by discarding all the DC information the DVS would be hard to use. However, for solving many dynamic vision problems this static information is not helpful. The DVS has been applied for various applications: high-speed ball tracking in a goalie robot (Delbrück and Lichtsteiner 2007), building an impressive pencil balancing robot (Conradt et al. 2009a, 2009b), highway traffic analysis (Litzenberger et al. 2006), stereo-based head tracking for people counting (Schraml et al. 2010), for tracking particle motion in fluid dynamics (Drazen et al. 2011), tracking microscopic particles (Ni et al. 2012), stereo vision based on temporal correlation (Rogister et al. 2012), and stereo-based gesture recognition (Lee et al. 2012). In other unpublished work (implementations of these unpublished projects is open-sourced in jaER (2007) as the classes `Info`, `BeeCounter`, `SlotCarRacer`, `FancyDriver`, and `LabyrinthGame`), the sensor has also been used for analyzing sleep-wake activity of mice for a week, counting bees entering and leaving a hive, building a self-driving slot car racer, driving a high-speed electric monster truck along a chalked line, and controlling a ball on a Labyrinth game table.

DVS Pixel

The DVS pixel design uses a combination of continuous and discrete time operation, where the timing is self-generated. The use of self-timed switched capacitor architecture leads to well-matched pixel response properties and fast, wide DR operation. To achieve these characteristics, the DVS pixel uses a fast logarithmic photoreceptor circuit, a differencing circuit that amplifies changes with high precision and simple comparators. Figure 3.2a shows how these three components are connected. The photoreceptor circuit automatically controls individual pixel gain (by its logarithmic response) while at the same time responding quickly to changes in illumination. The drawback of this photoreceptor circuit is that transistor threshold variation causes substantial DC mismatch between pixels, necessitating calibration when this output is used directly (Kavadias et al. 2000; Loose et al. 2001). The DC mismatch is removed by balancing the output of the differencing circuit to a reset level after the generation of an event. The gain of the change amplification is determined by the well-matched capacitor ratio C_1/C_2 . The effect of random comparator mismatch is reduced by the precise gain of the differencing circuit.

Due to the logarithmic conversion in the photoreceptor and the removal of DC by the differencing circuit, the pixel is sensitive to ‘temporal contrast’ TCON, which is defined as

$$\text{TCON} = \frac{1}{I(t)} \frac{dI(t)}{t} = \frac{d(\log(I(t)))}{dt}, \quad (3.1)$$

where I is the photocurrent. (The units of I do not affect $d(\log I)$.) The log here is the natural logarithm. Figure 3.2b illustrates the principle of operation of the pixel. The operation of some of the pixel components will be discussed in the latter part of this chapter.

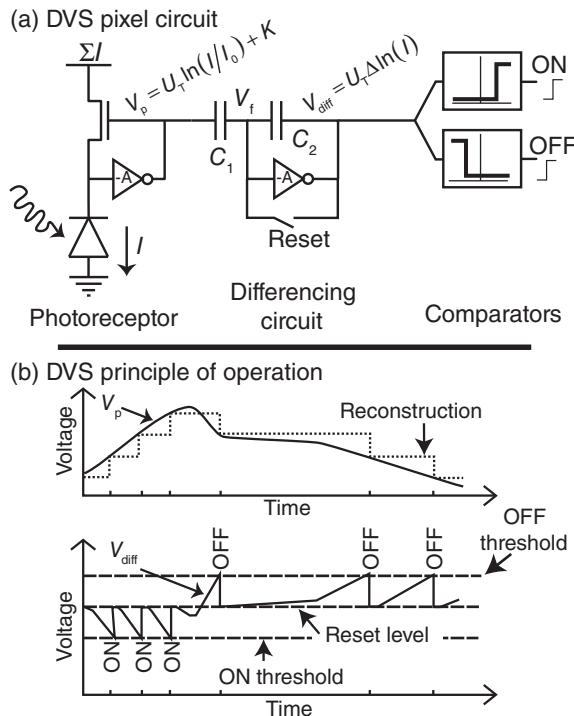


Figure 3.2 DVS pixel. (a) Simplified pixel schematic. (b) Principle of operation. In (a), the inverters are symbols for single-ended inverting amplifiers. © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

The DVS was originally developed as part of the CAVIAR multichip AER vision system (Serrano-Gotarredona et al. 2009) in Chapter 16 but it can be directly interfaced to other AER components that use the same word-parallel protocol or can be adapted to other AER protocols with glue logic. The DVS streams time-stamped address events to a host PC over a high-speed USB 2.0 interface. The camera architecture is discussed in more detail in Chapter 13. On the host side, there is a lot of complexity in acquiring, rendering, and processing the nonuniformly distributed, asynchronous retina events in real time on a hardware single-threaded platform like a PC or microcontroller. The infrastructure implemented through the open-source jAER project, consisting of several hundred Java classes, allows users to capture retina events, monitor them in real time, control the on-chip bias generators, and process them for various applications (jAER 2007).

DVS Example Data

Perhaps the most important section of any report for users (and paper reviewers) is the one displaying example data from real-world scenarios. As an example, the images in Figure 3.3 show DVS example data from Lichtsteiner et al. (2008). The dynamic properties are illustrated

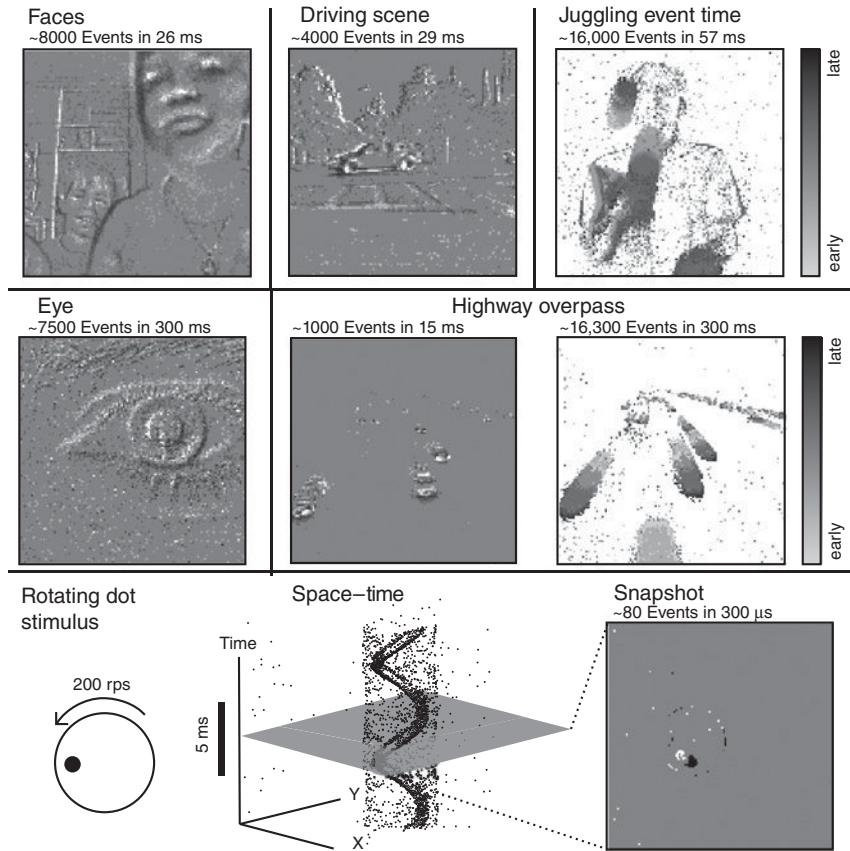


Figure 3.3 DVS data taken under natural lighting conditions with either object or camera motion. These are rendered as 2D histograms of collected events, either as contrast (gray scale represents reconstructed gray scale change), gray scale time (event time is shown as gray scale, black = young, gray = old, white = no event), or 3D space-time. © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

by using grayscale or 3D to show the time axis. The ‘Rotating Dot’ panel shows the events generated by a black dot drawn on a white disk rotating at 200 revolutions per second under indoor fluorescent office illumination of 300 lux. The events are rendered both in space-time over 10 ms and as a briefer snapshot image spanning 300 μ s. The spinning dot forms a helix in space time. The ‘Faces’ image was collected indoors at night with illumination from a 15-W fluorescent desk lamp. The ‘Driving Scene’ was collected outdoors under daylight from a position on a car dashboard when just starting to cross an intersection. The ‘Juggling Event Time’ image shows the event times as grayscale. The paths of the balls are shaded in gray showing the direction of movement. For example, the uppermost ball is rising because the top part of the path is darker, indicating later events in the time slice. The ‘Eye’ image shows events from a moving eye under indoor illumination. The ‘Highway Overpass’ images show

events produced by cars on a highway viewed from an overpass in late afternoon lighting, on the left displayed as ON and OFF events and on the right as relative time during the snapshot.

3.5.2 Asynchronous Time-Based Image Sensor

The ATIS (Posch et al. 2011) combines a DVS pixel and an intensity measurement unit (IMU) in each pixel. DVS events are used to trigger time-based intensity measurements. This impressive sensor, with a resolution of 304×240 $30\text{ }\mu\text{m}$ pixels in a 180 nm technology (see Table 3.1), combines in each pixel the notions of TC detection (Lichtsteiner et al. 2008) with pulse-width modulated (PWM) intensity encoding (Qi et al. 2004). It uses a new time-based correlated double sampling (CDS) circuit (Matolin et al. 2009) to output pixel gray level values only from pixels that change. This intensity readout scheme allows high DR exceeding 130 dB, which is especially valuable in surveillance scenarios. A good example is Belbachir et al. (2012).

The ATIS pixel asynchronously requests access to an output channel only when it has a new illumination value to communicate. The asynchronous operation avoids the time quantization of frame-based acquisition and scanning readout. The gray level is encoded by the time between two additional IMU events. This IMU measurement is started by a DVS event in the pixel. Example data from the ATIS is shown in Figure 3.4. The transistor circuits are discussed in the second part of this chapter in Section 3.6.2, with a focus on its effective time-based CDS.

3.5.3 Asynchronous Parvo–Magno Retina Model

The Parvo–Magno retina by Zaghloul and Boahen (2004a) is a very different type of silicon retina that was focused on modeling of both the outer and inner retina including sustained (Parvo) and transient (Magno) types of cells, based on histological and physiological findings. It is an improvement over the first retina by Mahowald and Mead which models only the

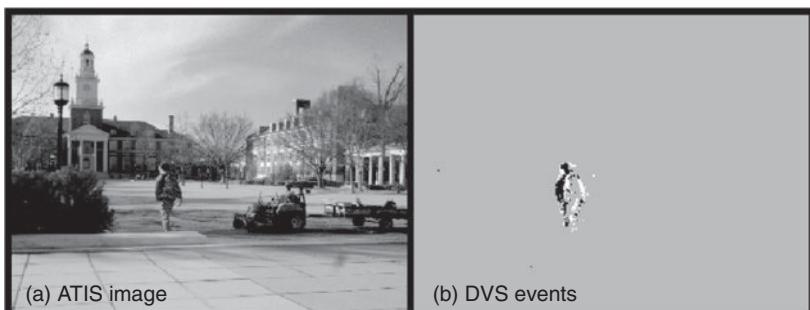


Figure 3.4 ATIS example data. (a) Capture of image of student walking to school is triggered externally by a rolling (row-wise) reset of all pixels. Subsequently, only intensity values of pixels that change (e.g., the moving student) are updated. (b) The sparse DVS events are generated in the ATIS pixels by the moving person. Reproduced with permission of Garrick Orchard

outer retina circuitry, that is, the cones, horizontal cells, and bipolar cells (Mahowald and Mead 1991). The use of the Parvo–Magno retina in an orientation feature extraction system is described in Section 13.3.2.

The Parvo–Magno design captures key adaptive features of biological retinas including light and contrast adaptation, and adaptive spatial and temporal filtering. By using small transistor log-domain circuits that are tightly coupled spatially by diffuser networks and single-transistor synapses, they were able to achieve 5760 phototransistors at a density of $722/\text{mm}^2$ and 3600 ganglion cells at a density of $461/\text{mm}^2$ tiled in $2 \times 48 \times 30$ and $2 \times 24 \times 15$ mosaics of sustained and transient ON and OFF ganglion cells in a $3.5 \times 3.3 \text{ mm}^2$ area, using a $0.25 \mu\text{m}$ process (see Table 3.1).

The overall Parvo–Magno model is shown in Figure 3.5. The cone outer segments (CO) supply photocurrent to cone terminals (CT), which excite horizontal cells (HC). Horizontal cells reciprocate with shunting inhibition. Both cones and horizontal cells are electrically coupled to their neighbors by gap junctions. Horizontal cells modulate cone to horizontal cell excitation and cone gap junctions. ON and OFF bipolar cells (BC) relay cone signals to ganglion cells (outputs) and excite narrow- and wide-field amacrine cells (NA, WA). They also excite amacrine cells that inhibit complementary bipolars and amacines. Narrow-field amacrine cells inhibit bipolar terminals and wide-field amacrine cells; their inhibition onto wide-field amacrine cells is shunting. They also inhibit transient ganglion cells (OnT, OffT), but not sustained ganglion cells (OnS, OffS). Wide-field amacrine cells modulate narrow-field amacrine cell pre-synaptic inhibition and spread their signals laterally through gap junctions.

The outer retina's synaptic interactions realize spatiotemporal bandpass filtering and local gain control. The model of the inner retina realizes contrast gain control (the control of sensitivity to TC) through modulatory effects of wide-field amacrine cell activity. As TC increases, their modulatory activity increases, the net effect of which is to make the transient

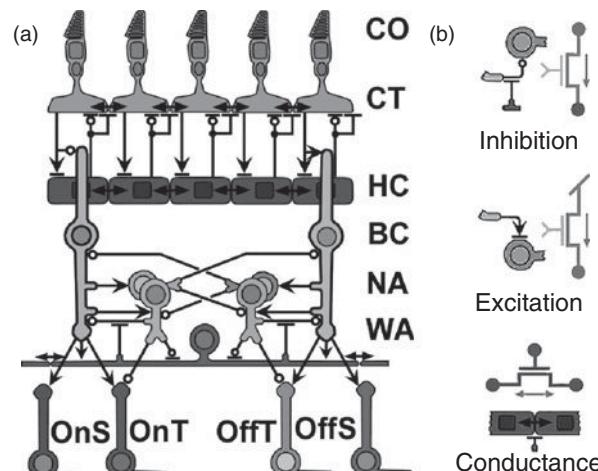


Figure 3.5 Parvo–Magno retina model. (a) Synaptic organization. (b) Single-transistor synapses. Adapted from Zaghoul and Boahen (2006). © IOP Publishing. Reproduced by permission of IOP Publishing. All rights reserved

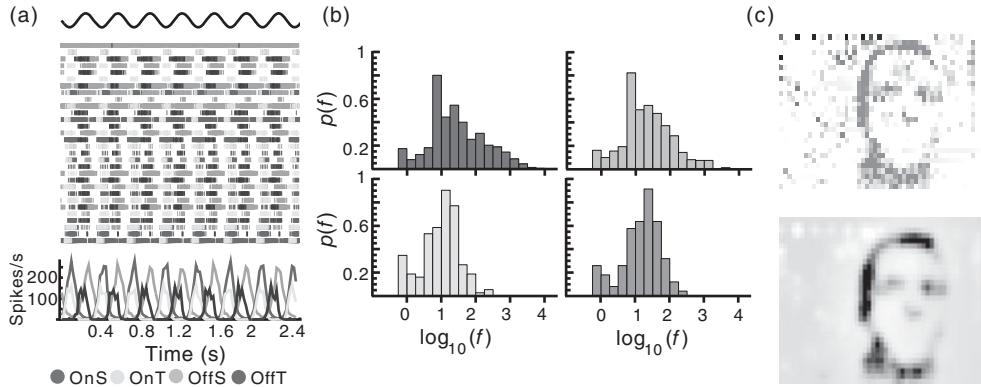


Figure 3.6 Parvo–Magno chip responses. (a) A raster plot of the spikes (top) and histogram (bottom, bin width = 20 ms) recorded from a single column of the chip array. The stimulus was a 3 Hz 50% – contrast drifting sinusoidal grating whose luminance varied horizontally across the screen and was constant in the vertical direction. The amplitude of the fundamental Fourier component of these histograms is plotted in all frequency responses presented here. (b) The distribution of firing rates for the four types of GC outputs, demonstrating the amount of response variability. Log of the firing rate f is plotted versus the probability density $p(f)$. Histograms are computed from all active cells of a given type in the array (40% of the sustained cells and 60% of the transient cells did not respond). (c) Top image shows that edges in a static scene are enhanced by sustained type GC outputs of the chip. Bottom image shows that an optimal mathematical reconstruction of the image from sustained GC activity demonstrates fidelity of retinal encoding despite the extreme variability. © 2004 IEEE. Reprinted, with permission, from Zaghloul and Boahen (2004b)

ganglion cells respond more quickly and more transiently. This silicon retina outputs spike trains that capture the behavior of ON- and OFF-center wide-field transient and narrow-field sustained ganglion cells, which provide 90% of the primate retina’s optic nerve fibers. These are the four major types that project, via thalamus, to the primary visual cortex.

Figure 3.6 shows sample data from the Parvo–Magno retina. Clearly this device has complex and interesting properties, but the extremely large mismatch between pixel response characteristics makes it quite hard to use.

3.5.4 Event-Based Intensity-Coding Imagers (Octopus and TTFS)

Rather than modeling the redundancy reduction and local computation done in the retina, some imagers simply use the AER representation to directly communicate pixel intensity values. These sensors encode the pixel intensity either in the spike frequency (Azadmehr et al. 2005; Culurciello et al. 2003) or in the time to first spike, referenced to a reset signal (Qi et al. 2004; Shoushun and Bermak 2007).

The so-called ‘Octopus retina’ by Culurciello et al. (2003), for example, communicates the pixel intensity through the frequency or inter-spike interval of the AER event outputs. (The Octopus retina is named after the animal because the octopus eye, and those of other cephalopods, is inverted compared to mammalian eyes – the photoreceptors are on the inside

of the eye rather than the outside – and some of the retinal processing is apparently done in the optic lobe. However, the octopus eye still has a complex architecture and is capable, for instance, of discriminating polarized light.) This imager is also used as a front end of the IFAT system described in Section 13.3.1.

This biologically inspired readout method offers pixel-parallel readout. In contrast, a serially scanned array allocates an equal portion of the bandwidth to all pixels independent of activity and continuously dissipates power because the scanner is always active. In the Octopus sensor, brighter pixels are favored because their integration threshold is reached faster than darker pixels, thus their AER events are communicated at a higher frequency. Consequently, brighter pixels request the output bus more often than darker ones and are updated more frequently. Another consequence of this asynchronous image readout is image lag behind moving objects. A moving object consisting of bright and dark parts will result in later intensity readings for the dark parts than for the bright parts, distorting the object's shape and creating motion artifacts similar to those seen from the ATIS (Section 3.6.2).

Octopus retina example images are shown in Figure 3.7. The Octopus scheme allows wide DR and that is why the shaded face and light bulb scenes are visible when the images are rendered using a logarithmic encoding. But the rather large FPN makes the sensor hard to sell for conventional imaging. This FPN is due mostly to threshold variation in the individual PFM circuits.

The Octopus imager is composed of 80×60 pixels of $32 \times 30 \text{ mm}^2$ fabricated in a $0.6 \mu\text{m}$ CMOS process. It has a DR of 120 dB (under uniform bright illumination and when no lower bound was placed on the update rate per pixel). The DR is 48.9 dB value at 30-pixel updates per second. Power consumption is 3.4 mW in uniform indoor light and a mean event rate of 200 kHz, which updates each pixel 41.6 times per second. The imager is capable of updating each pixel 8.3 K times per second under bright illumination.

The Octopus imager has the advantage of a small pixel size compared to the other AER retinas, but has the disadvantage that the bus bandwidth is allocated according to the local

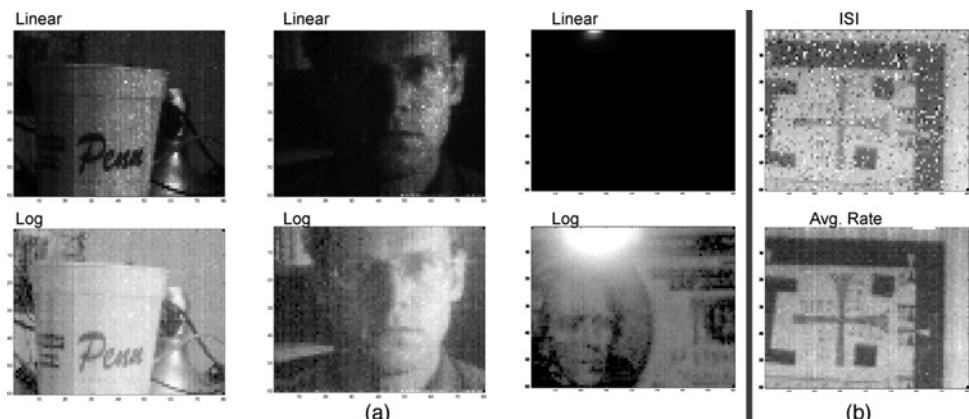


Figure 3.7 Example images from a fabricated Octopus retina of 80×60 pixels. (a) Linear intensity (top) and log (bottom) scales. The images have been reconstructed by counting large numbers of spike events. (b) Comparing instantaneous ISI with average spike count (here the average count over image is 200 spikes). © 2003 IEEE. Reprinted, with permission, from Culurciello et al. (2003)

scene luminance. Because there is no reset mechanism and the event interval directly encodes intensity, a dark pixel can take a long time to emit an event, and a single highlight in the scene can saturate the off-chip communication spike address bus. Another drawback of this approach is the complexity of the digital frame grabber required to count the spikes produced by the array. The buffer must either count events over some time interval, or hold the latest spike time and use this to compute the intensity value from the ISI to the current spike time. This however leads to a noisier image. The Octopus retina would probably be most useful for tracking small and bright light sources.

The alternate way of coding intensity information is the time to first spike (TTFS) coding implementation from the group of Harris and others (Luo and Harris 2004; Luo et al. 2006; Qi et al. 2004; Shoushun and Bermak 2007). In this encoding method, the system does not require the storage of large number of spikes, since every pixel only generates one spike per frame. A similar coding method was also suggested as a rank-order scheme used by neurons in the visual system to code information (Thorpe et al. 1996). The global threshold for generating a spike in each pixel can be reduced over the frame time so that dark pixels will still emit a spike in a reasonable amount of time. A disadvantage of the TTFS sensors is that uniform parts of the scene all try to emit their events at the same time, overwhelming the AER bus. An example of mitigating this problem would be by serially resetting rows of pixels, but then the problem remains that a particular image could still cause simultaneous emission of events.

The TTFS notion is used in the ATIS (Section 3.5.2) and in a related form in the VISe, discussed next. In the ATIS, the DVS events in individual pixels trigger a reading of intensity like the TTFS sensors have, but because the reading is local and triggered by intensity change, it overcomes some of the main drawbacks of the TTFS sensors.

3.5.5 Spatial Contrast and Orientation Vision Sensor (VISe)

The beautifully designed VISe (VISe stands for VIision Sensor) (Ruedi et al. 2003) has a complementary functionality to the DVS (Section 3.5.1). The VISe outputs events that code for spatial rather than TC. The VISe pixels compute the contrast magnitude and angular orientation. It differs from the previous sensors in that it outputs these events using a temporal ordering according to the contrast magnitude. This ordering can greatly reduce the amount of data delivered. The VISe design results in impressive performance, particularly in terms of low mismatch and high sensitivity.

A VISe cycle starts with a global frame exposure, for example, 10 ms. Next the VISe transmits address events in the order of high-to-low SC. Pixels that emit events with the earliest times correspond to pixels that see the highest local SC. Readout can be aborted early if limited processing time is available without losing information about high-contrast features. Each contrast event is followed by another event that encodes gradient orientation by the timing of the event relative to global steering functions, which are described later. The VISe has low 2% contrast mismatch, a contrast direction precision of 3 bits, and a large 120 dB illumination DR. The main limitation of this architecture is that it does not reduce temporal redundancy (compute temporal derivatives), and its temporal resolution is limited to the frame rate.

The traditional APS approach to acquire an image with a CMOS imager consists of integrating on a capacitor, in each pixel of an array, the photocurrent delivered by a photodiode for a fixed exposure time (Fossum 1995). Without saturation, the resulting voltages are then

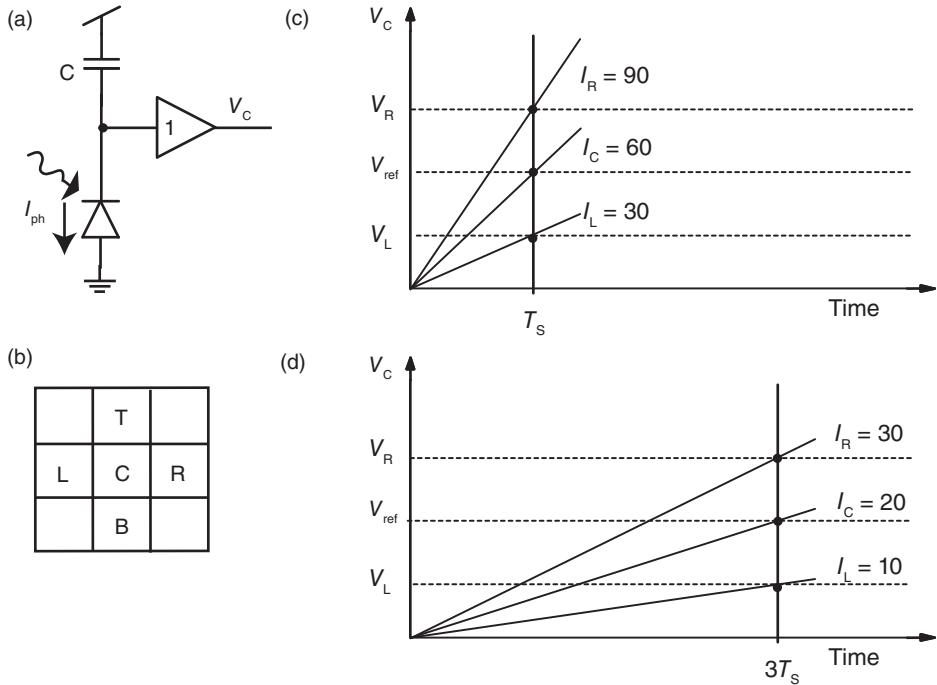


Figure 3.8 Contrast generation in the VISe. © 2003 IEEE. Reprinted, with permission, from Ruedi et al. (2003)

proportional to the photocurrents. In the VISe pixel, like the Octopus or ATIS retina, instead of integrating photocurrents for a fixed duration, photocurrents are integrated until a given reference voltage is reached. This principle as used in the VISe is illustrated in Figure 3.8, where a central pixel and its four neighbors are considered (left, right, top, and bottom). In each pixel, the photocurrent \$I_c\$ is integrated on a capacitor \$C\$. The resulting voltage is \$V_c\$ continuously compared to a reference voltage \$V_{ref}\$. Once \$V_c\$ reaches \$V_{ref}\$, the central pixel samples the voltages \$V_L\$, \$V_R\$, \$V_T\$, and \$V_B\$ of the four neighboring pixels and stores these voltages on capacitors.

The local integration time \$T_S\$ is given by \$T_S = \frac{CV_{ref}}{I_C}\$. If \$I_X\$ is the photocurrent in neighbor \$X\$, then the sampled voltage will be

$$V_X = \frac{I_X T_S}{C} = \frac{I_X}{I_C} V_{ref}. \quad (3.2)$$

\$V_X\$ depends only on the ratio of the photocurrents generated in pixels \$X\$ and \$C\$. With the assumption that the photocurrent of the central pixel corresponds to the average of its four neighbors – which is mostly fulfilled with the spatial low-pass characteristics of the optics – one obtains

$$I_C = \frac{I_R + I_L}{2} = \frac{I_T + I_B}{2}. \quad (3.3)$$

Combining Eqs. (3.2) and (3.3) and solving for $V_R - V_L$ gives

$$V_R - V_L = 2 \frac{I_R - I_L}{I_R + I_L} V_{\text{ref}}. \quad (3.4)$$

The same form of equation can be derived for $V_T - V_B$. Equation (3.4) is equivalent to the conventional definition of the Michelson contrast, which is the difference divided by the average.

Voltages V_X and V_Y can be considered the X and Y components of a local contrast vector. If the illumination level is decreased by a factor of three, as illustrated in Figure 3.8, the time elapsed between the start of the integration and the sampling of the four neighbors is three times longer so that the sampled voltages remain the same.

The computation of the contrast direction and magnitude of the contrast vector defined in Eqs. (3.3) and (3.4), and the readout of the combined contrast and orientation signal, are together illustrated in Figure 3.9. Each pixel continuously computes the scalar projection $F(t)$ of the local contrast vector on a rotating unit vector \vec{r} . When \vec{r} points in the same direction as the local contrast vector, $F(t)$ goes through a maximum. Thus, $F(t)$ is a sine wave whose amplitude and phase represent the magnitude and direction of the contrast vector. The projection is computed locally, in each pixel, by multiplying the local contrast components by globally distributed steering signals. The steering signals are sinusoidal voltages arranged in quadrature to represent the rotation of \vec{r} .

To illustrate these operations and the data output, a frame acquisition and the whole sequence of operations for two pixels a and b are shown in Figure 3.9. During a first phase, photocurrents

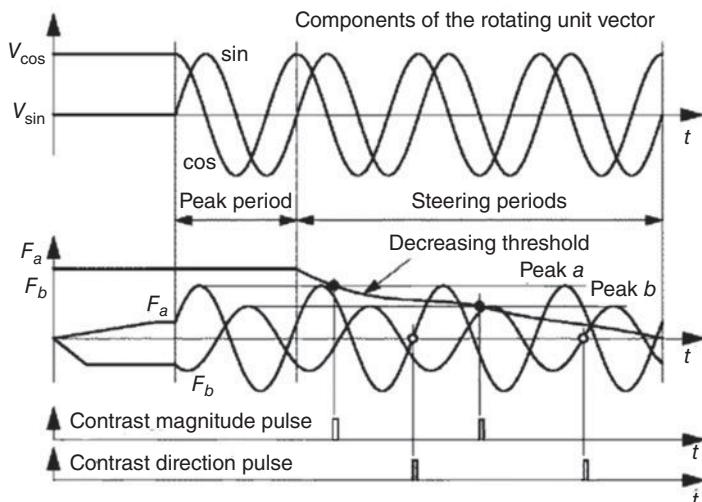


Figure 3.9 VISe computation and data output principle (two pixels a and b are represented). © 2003 IEEE. Reprinted, with permission, from Ruedi et al. (2003)

are integrated. The steering functions are then turned on. During the first steering period, $F_a(t)$ and $F_b(t)$ go through a maximum, which is detected and memorized by a simple maximum detector (peaks *a* and *b*). During the subsequent periods, a monotonously decreasing threshold function is distributed to all pixels in parallel. In each pixel, this threshold function is continuously compared to this maximum. To output the contrast magnitude, an event encoding the address (X, Y) of the pixel is emitted on the AER bus (Mortara et al. 1995) when the threshold function reaches the maximum of the individual pixel. The contrast present at each pixel is time encoded by the high contrasts preceding the lower ones. This scheme limits data transmission to pixels with a contrast higher than a given value. To dispatch the contrast direction, a second event is emitted at the first zero crossing with positive slope that follows. This way, the contrast direction is ordered in time according to the contrast magnitude. Without gating of the contrast direction by the contrast magnitude, each pixel would fire a pulse at the first occurrence of the zero crossing. Notice that the zero crossing is shifted by -90° with respect to the maximum of $F(t)$.

Figure 3.10 illustrates a number of images of the contrast representation output by the sensor. Even a high DR scene consisting of a person and an illuminated light bulb, or an indoor/outdoor scene, or a scene containing the sun, are still represented clearly. And even low contrast edges are cleanly output.

Although the VISE is a lovely and imaginative architecture, it has been replaced in the group's developments by a different approach based around a digital logarithmic image sensor tightly integrated with a custom DSP (Ruedi et al. 2009). The reason cited for abandoning the VISE approach is the desire to scale better to smaller digital process technologies and the desire to integrate tightly to a custom digital coprocessor on the same die.



Figure 3.10 VISE SC output. Each image shows the detected contrast magnitude as gray level. The magnitude of contrast is encoded by the time of the emission of the event relative to start of readout. © 2003 IEEE. Reprinted, with permission, from Ruedi et al. (2003)

3.6 Silicon Retina Pixels

The first part of this chapter presented an overview of the DVS, the ATIS, the Parvo–Magno retina, the Octopus, and the VISe vision sensors. The second part of this chapter discusses the pixel details of some of the retina designs.

3.6.1 DVS Pixel

Figure 3.2 shows an abstract view of the DVS pixel. This section discusses the actual implementation of the pixel, shown in Figure 3.11. The pixel consists of four parts: photoreceptor, buffer, differencing amplifier, comparators, plus the AER handshaking circuits.

The photoreceptor circuit uses a well-known transimpedance configuration (see, e.g., Delbrück and Mead 1994) that converts the photocurrent logarithmically into a voltage. The photodiode PD photocurrent is sourced by a saturated nFET M_{fb} . The gate of M_{fb} is connected to the output of an inverting amplifier (M_{pr} , M_{cas} , M_n) whose input is connected to the photodiode. Because this configuration holds the photodiode clamped at a virtual ground, the bandwidth of the photoreceptor is extended by the factor of the loop gain in comparison to a passive logarithmic photoreceptor circuit. This extended bandwidth is beneficial for high-speed applications, especially in low lighting conditions. The global sum of

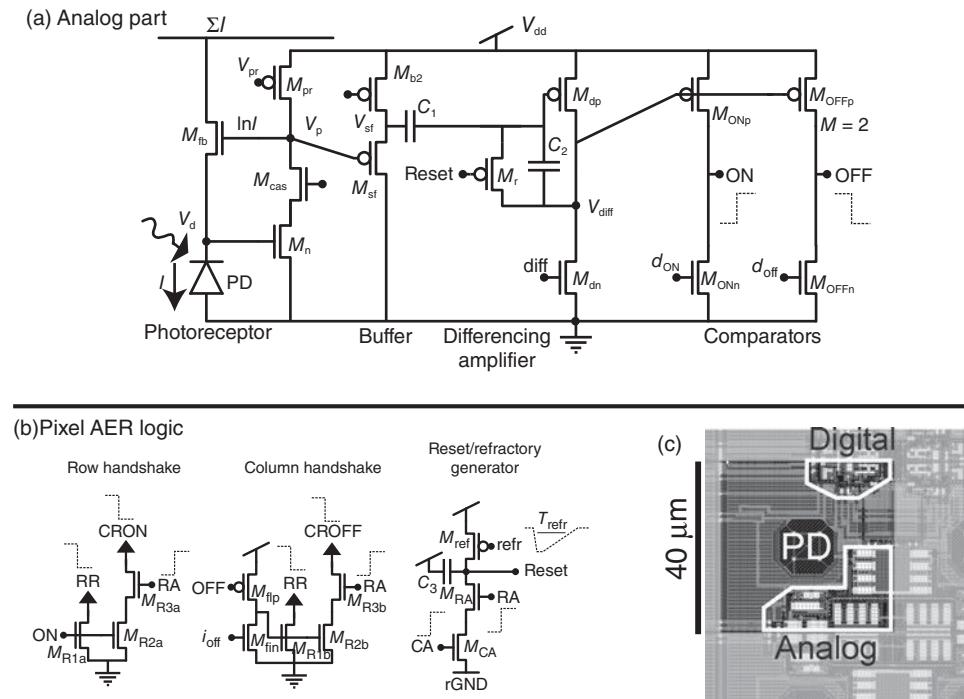


Figure 3.11 Complete DVS pixel circuit. (a) Analog part. (b) AER handshaking circuits. (c) Pixel layout. © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

all photocurrents is available at the ΣI node, where it is sometimes used for adaptive control of bias values.

The photoreceptor output V_p is buffered with a source follower to V_{sf} to isolate the sensitive photoreceptor from the rapid transients in the differencing circuit. (It is important in the layout to avoid any capacitive coupling to V_d because these can lead to false events and excessive noise.) The source follower drives the capacitive input of the differencing circuit.

Next the differencing amplifier amplifies changes in the buffer output. This inverting amplifier with capacitive feedback is balanced with a reset switch that shorts its input and output together, resulting in a reset voltage level that is about a diode drop from V_{dd} . Thus the change of V_{diff} from its reset level represents the amplified change of log intensity.

It is straightforward to see that the relation between the TC TCON defined in Eq. (3.1) and V_{diff} is given by

$$\Delta V_{diff} = -A \cdot \Delta V_{sf} = -A \cdot \kappa_{sf} \cdot \Delta V_p \quad (3.5)$$

$$= -A \cdot \frac{U_T \kappa_{sf}}{\kappa_{fb}} \ln \left(\frac{I(t + \Delta t)}{I(t)} \right) = -A \cdot \frac{U_T \kappa_{sf}}{\kappa_{fb}} \Delta \ln(I) \quad (3.6)$$

$$= -A \cdot \frac{U_T \kappa_{sf}}{\kappa_{fb}} \int_t^{t+\Delta t} \text{TCON}(t') dt', \quad (3.7)$$

where $A = C_1/C_2$ is the differencing circuit gain, U_T is the thermal voltage, and κ_x is the subthreshold slope factor of transistor M_X . This equation might make it appear that the amplifier somehow integrates, and yes, this is true, it does integrate TCON, but TCON is already the derivative of log intensity. Therefore, it boils down to simple amplification of the change in log intensity since the last reset was released.

The comparators ($M_{ONn}, M_{ONp}, M_{OFFn}, M_{OFFp}$) compare the output of the inverting amplifier against global thresholds that are offset from the reset voltage to detect increasing and decreasing changes. If the input of a comparator overcomes its threshold, an ON or OFF event is generated.

Replacing ΔV_{diff} in Eq. (3.7) by comparator input thresholds d_{on} and d_{off} and solving for $\Delta \ln(I)$ yield the threshold positive and negative TCs θ_{on} and θ_{off} that trigger ON or OFF events:

$$\theta_{on} = \Delta \ln(I)_{\min,ON} = -\frac{\kappa_{fb}}{\kappa_{sf} \kappa_{ONp} U_T A} \cdot (\kappa_{ONn}(\text{don} - \text{diff}) + U_T), \quad (3.8)$$

$$\theta_{off} = \Delta \ln(I)_{\min,OFF} = -\frac{\kappa_{fb}}{\kappa_{sf} \kappa_{OFFp} U_T A} \cdot (\kappa_{OFFn}(\text{doff} - \text{diff}) - U_T), \quad (3.9)$$

where $\text{don} - \text{diff}$ is the ON threshold and $\text{doff} - \text{diff}$ is the OFF threshold.

The threshold TC θ has dimensions of natural log of intensity and is hereafter called contrast threshold. Each event represents a change of log intensity of θ . For smoothly varying TCs, the rate $R(t)$ of generated ON and OFF events can be approximated with

$$R(t) \approx \frac{\text{TCON}(t)}{\theta} = \frac{1}{\theta} \frac{d}{dt} \ln(I). \quad (3.10)$$

ON and OFF events are communicated to the periphery by circuits that implement the 4-phase address-event handshaking (see Chapter 2) with the peripheral AE circuits discussed in detail in Chapter 12. The row and column ON and OFF request signals (RR, CRON, CROFF) are generated individually, while the acknowledge signals (RA, CA) are shared. They can be shared because the pixel makes either an ON or OFF event, never both simultaneously. The row signals RR and RA are shared by pixels along rows and the signals CRON, CROFF, and CA are shared along columns. The signals RR, CRON, and CROFF are pulled high by statically biased pFET row and column pull-ups. When either the ON or OFF comparator changes state from its reset condition, the communication cycle starts. The communication cycle ends by turning on the reset transistor M_r which removes the pixel request.

M_r resets the pixel circuit by balancing the differencing circuit. This transistor also has the important function of enabling an adjustable refractory period T_{refr} (implemented by the starved NAND gate consisting of M_{rof} , M_{RA} , and M_{CA}), during which the pixel cannot generate another event. This refractory period limits the maximum firing rate of individual pixels to prevent small groups of pixels from taking the entire bus capacity.

When M_r is turned off, positive charge is injected from its channel onto the differencing amplifier input. This is compensated by slightly adjusting the thresholds don and doff around diff .

The smallest threshold that can be set across an array of pixels is determined by mismatch: as the threshold is decreased, a pixel with a low threshold will eventually not stop generating an event even when the differencing amplifier is reset. Depending on the design of the AER communication circuits, this state will either hang the bus or causes a storm of events from this pixel that consumes bus bandwidth. Mismatch characterization is further discussed in Section 3.7.1. The contrast sensitivity of the original DVS can be enhanced by inserting an extra voltage preamplification stage at node V_{ph} in Figure 3.12a. This modified design from Leñero Bardallo et al. (2011b) uses a few extra transistors to implement a two-stage voltage amplification before the capacitive differentiator/amplifier. The photoreceptor circuit is also modified to the one in Figure 3.12b. The amplifying transistors are biased in the limit of strong to weak inversion. The extra amplifiers introduce a voltage gain of about 25, which allows reducing the capacitive gain stage (and area) by about 5, while still allowing overall gain improvement. As a result, the pixel area is reduced by about 30%, contrast sensitivity is improved, overall mismatch is only slightly degraded about a factor 2, but power consumption is significantly increased by a factor between 5 and 10. This technique is especially valuable when MIM capacitors are not available and poly capacitors occupy valuable transistor area. It has the possible drawback that now an AGC circuit must be implemented to center the DC operating point of the photoreceptor around the proper light intensity. Control of the gain can then globally generate events as though the illumination were changing. A more recent improvement to the preamplifier reduces the power consumption significantly (Serrano-Gotarredona and Linares-Barranco 2013).

3.6.2 ATIS Pixel

The ATIS pixel detects TC and uses this to locally trigger intensity measurements (Figure 3.13a). The change detector initiates the exposure of a new gray-level measurement. The asynchronous operation also avoids the time quantization of frame-based acquisition and scanning readout.

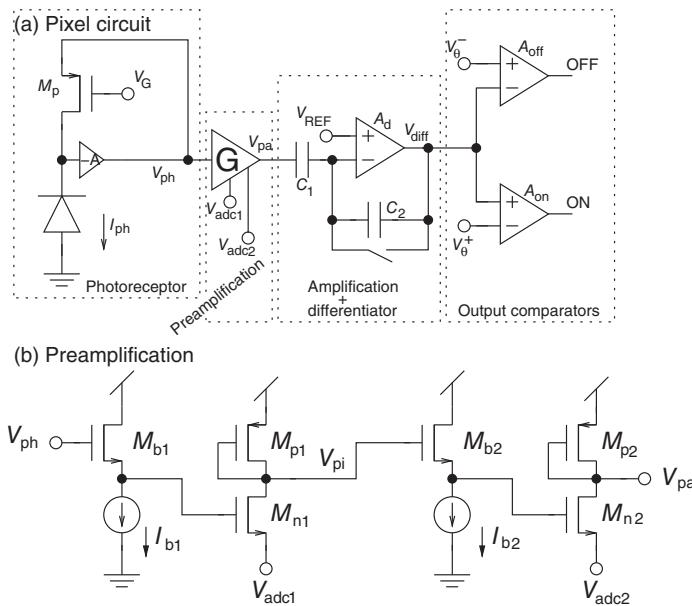


Figure 3.12 DVS pixel with higher sensitivity. (a) Pixel circuit, showing photoreceptor with feedback to gate of pFET that has fixed gate voltage V_G and preamplifier G. (b) Details of G preamplifier. © 2011 IEEE. Reprinted, with permission, from Leñero Bardallo et al. (2011b)

The temporal change detector consists of the pixel circuits in the DVS. For the IMU, a time-based PWM imaging technique was developed. In time-based or pulse modulation imaging, the incident light intensity is encoded in the timing of pulses or pulse edges. This approach automatically optimizes the integration time separately for each pixel instead of imposing a fixed integration time for the entire array, allowing for high DR and improved signal-to-noise-ratio (SNR). DR is no longer limited by the power supply rails as in conventional CMOS

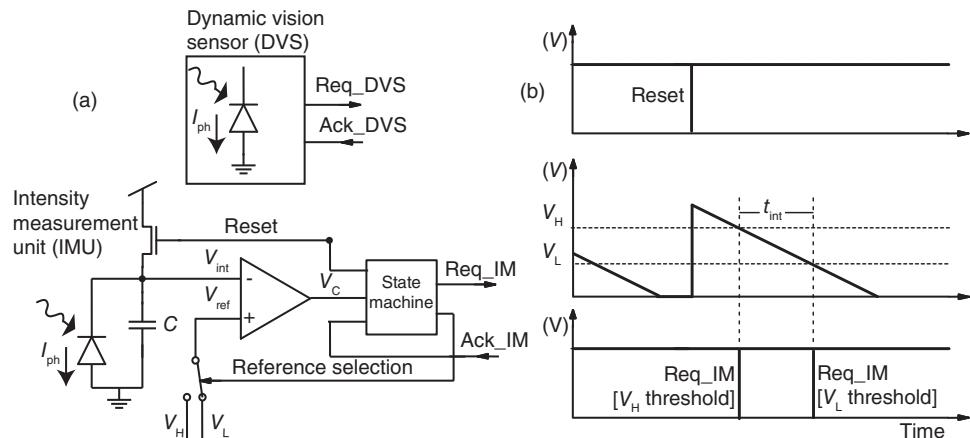


Figure 3.13 ATIS pixel (a) and examples of pixel signals (b)

APS, providing some immunity to the supply voltage scaling of modern CMOS technologies. Instead, DR is now only limited by the dark current and allowable integration time. As long as there is more photocurrent than the dark current and the user can wait for sample long enough to cover the integration voltage range, the intensity can still be captured. The photodiode is reset and subsequently discharged by the photocurrent. An in-pixel comparator triggers a digital pulse signal when a first reference voltage V_H is reached, and then another pulse when the second reference voltage V_L is crossed (Figure 3.13b). The resulting integration time t_{int} is inversely proportional to the photocurrent. Independent AER communication circuits on the four sides of the sensor read out the DVS and two IMU events.

Time-domain true correlated double sampling (TCDS) (Matolin et al. 2009) based on two adjustable integration thresholds (V_H and V_L) and intra-pixel state logic ('state machine') completes the pixel circuit (Figure 3.13a). The key feature of TCDS is the use of a single comparator where the reference is switched between the high and low thresholds. This way, comparator mismatch and kTC noise are both canceled out, improving both shot noise and FPN. $V_H - V_L$ can be set as small as 100 mV with acceptable image quality, allowing operation down to low intensities. However, IMU capture of fast moving objects requires high light intensity so that the object is not too blurred during the pixel exposure; significant drawback is that small or thin moving objects can disappear from the IMU output because the IMU exposure is restarted by the trailing edge of the object, resulting in an exposure of the background after the object has already passed. This effect is a specific instance of the motion artifacts that result from exposures that are not globally synchronous, and the need for so-called 'global shutter' imagers drives a large market for such sensors in machine vision applications.

The asynchronous change detector and the time-based photo-measurement approach complement each other nicely, mainly for two reasons: first, because both reach a $DR > 120$ dB – the former is able to detect small relative change over the full range, while the latter is able to resolve the associated gray levels independently of the initial light intensity; second, because both circuits are event based, namely the events of detecting brightness changes and the events of pixel integration voltages reaching thresholds.

3.6.3 VISe Pixel

The VISe sensor introduced in Section 3.5.5 measures SC and orientation. Here we will describe only the VISe front-end circuit to illustrate how it integrates over a fixed voltage range and then generates a sampling pulse. This sampling pulse is then used in the full VISe pixel to sample neighboring pixel photodetector voltages onto capacitors. These capacitors are the gates of the multiplier circuit used for contrast orientation measurement, which will not be described here. Figure 3.14 shows the schematic of the VISe photocurrent integration and sampling block. When signal RST is high, capacitor C is reset to voltage V_{black} . Transistor M_C together with OTA1 maintain a constant voltage across the photodiode so that the photocurrent is not integrated on the parasitic capacitance of the photodiode. The transconductance amplifier has an asymmetrical input differential pair to generate a voltage V_{ph} around 25 mV. The voltage follower made of M_{P1} and M_{P2} is a simple p-type source follower in a separate well to get a gain close to one and minimize the parasitic input capacitance. Voltage V_C is distributed to the four neighbors and compared to a reference voltage V_{ref} . At the beginning of the photocurrent integration, signal V_{out} is high. When V_C reaches V_{ref} , signal V_{out} , which is fed to the four-quadrant multiplier block, goes down.

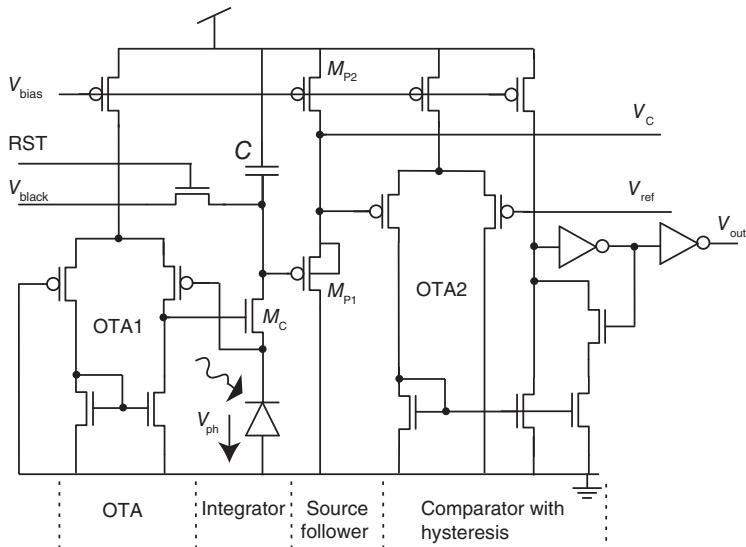


Figure 3.14 VISE pixel front end. © 2003 IEEE. Reprinted, with permission, from Ruedi et al. (2003)

3.6.4 Octopus Pixel

The Octopus retina pixel introduced in Section 3.5.4 reports intensity by measuring the time taken for the input photocurrent to be integrated on a capacitor to a fixed voltage threshold. When the threshold is crossed, an AER spike is generated by the pixel. The magnitude of the photocurrent is represented by the inter-event interval or frequency of spikes from a pixel. Although this representation does not reduce redundancy or compress DR and thus have much relation to biological retinal function, the circuit is interesting and forms the basis for some popular low-power spike generators used in silicon neurons as discussed in Chapter 7.

The pixel in the retina uses an event generator circuit that has been optimized so that an event is generated quickly and with little power consumption. A typical digital inverter using minimum size transistors in a 0.5 μm process and 3.3 V supply consumes only about 0.1 pJ capacitive charging energy per transition when the input signal is a perfect square wave. However, a linear photosensor slew rate (or a silicon neuron membrane potential) can be six orders of magnitude slower than typical digital signals (or 1 V/ms) in ambient lighting. In that case, a standard inverter operates in a region where both nFETs and pFETs are simultaneously conducting for a long time, and then an event generator like Mead's axon hillock consumes about 4 nJ. Therefore, the power consumption of the inverter as the event generator is about four to five orders of magnitude greater than that of a minimum-size inverter in a digital circuit. To solve this problem, the Octopus imager uses positive feedback to shorten the transition time in the circuit shown in Figure 3.15. Unlike the positive feedback circuit used in Mead's axon hillock circuit (Chapter 7) which uses capacitively coupled voltage feedback, the Octopus pixel uses current mirror-based positive feedback of current.

After reset, V_{in} is at a high voltage. As V_{in} is pulled down by photocurrent, it turns on M2, which eventually turns on M5, which accelerates the drop of V_{in} . Eventually the pixel enters positive feedback that results in a down-going spike on V_{out} . The circuit is cleverly arranged

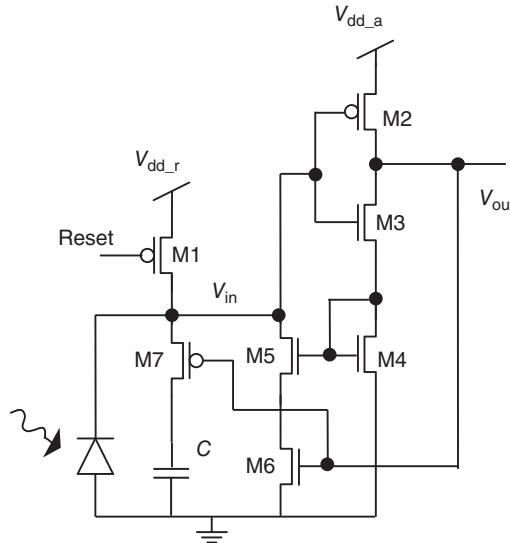


Figure 3.15 Octopus pixel schematic. © 2003 IEEE. Reprinted, with permission, from Culurciello et al. (2003)

so that the extra integration capacitor C is disconnected by M7 during the spike to reduce the load and power consumption. Once the input of the inverter V_{in} reaches ground, the inverter current goes to zero and so does the feedback, because M3 turns off M4. Thus at initial and final state there is no power-supply current. Measurements show that at an average spike rate of 40 Hz, each event requires about 500 pJ of energy, a factor of eight times less power than the axon hillock circuit. The improvement becomes larger as the light intensity decreases. An array of VGA size (640×480 pixels) at an average firing rate of 40 Hz would burn about 6 mW in the pixels and would produce events at a total rate of about 13 Meps, which despite all these tricks to reduce power still is a substantial amount of power consumption and data to process.

3.7 New Specifications for Silicon Retinas

Because of the nature of the event-based readout of AER silicon retinas, new specifications are needed to aid the end user and to enable quantifying improvements in sensor performance. As examples of these specifications, we use the DVS and discuss characteristics such as response uniformity, DR, pixel bandwidth, latency, and latency jitter, with more detail than reported in Lichtsteiner et al. (2008). A good recent publication on measurement of TC noise and uniformity characteristics by using flashing LED stimuli is Posch and Matolin (2011).

3.7.1 DVS Response Uniformity

For standard CMOS image sensors (CIS), FPN characterizes the uniformity of response across pixels of the array. For event-based vision sensors, the equivalent measure is the pixel-to-pixel variation in the event threshold, which is the threshold when the output of the comparator within

the pixel generates an event. Some pixels will generate more events and others will generate fewer events or even no events. This variation depends on the settings of the comparator thresholds and is due to pixel-to-pixel mismatch. For the DVS, which we use here as an example, the threshold θ is defined in terms of minimum log intensity change to generate an ON or OFF event. Because $d(\log I) = dI/I$, the threshold θ is the same as the minimum TC to generate an event. The contrast threshold mismatch is defined as σ_θ , the standard deviation of θ .

The dominant source of mismatch is expected to be found in the relative mismatch between differencing circuit reset level and comparator thresholds because device mismatch for transistors is in the order of 30% while capacitor mismatch is only in the order of 1% and the front-end photoreceptor steady-state mismatch is eliminated by differencing, and gain mismatch (κ mismatch) in the photoreceptor is expected to be in the order of 1%.

To measure the variation in event threshold, Lichtsteiner et al. (2008) used a black bar with linear gradient edges (which reduces the effects of the pixel refractory period) which was moved at a constant projected speed of about 1 pixel/10 ms through the visual field of the sensor. A pan-tilt unit smoothly rotated the sensor and a lens with long focal length minimized geometric distortion. Figure 3.16 shows the resulting histogram of events per pixel per stimulus edge for six different threshold settings. The threshold mismatch is measured from the width of the distributions combined with the known stimulus contrast of 15:1. Assuming an event threshold $\theta = \Delta \ln(I)$, (and assuming identical ON and OFF thresholds), and a threshold variation σ_θ , then an edge of log contrast $C = \ln(I_{\text{bright}}/I_{\text{dark}})$ will make $N \pm \sigma_N$ events:

$$N \pm \sigma_N = \frac{C}{\theta} \left(1 \pm \frac{\sigma_\theta}{\theta} \right). \quad (3.11)$$

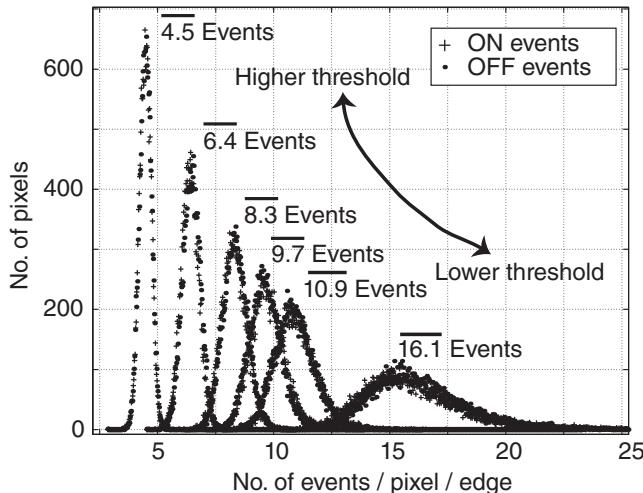


Figure 3.16 Distributions in the number of DVS events recorded per pass of a dark bar for 40 repetitions of the 15 : 1 contrast bar sweeping over the pixel array, for example, for the highest threshold setting there are an average of 4.5 ON and 4.5 OFF events per ON or OFF edge per pixel. © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

From Eq. (3.11) come the computed expressions for θ and σ_θ :

$$\theta = \frac{C}{N}; \quad \sigma_\theta = \theta \frac{\sigma_N}{N}, \quad (3.12)$$

where C is measured from the stimulus using a spot photometer under uniform lighting; N and σ_N are measured from the histograms.

Posch and Matolin (2011) summarizes many DVS pixel characteristics and reports a useful method for measuring DVS threshold matching using a flashing LED. This method characterizes the response by analyzing the *s*-shaped logistic function probabilistic DVS event response to step changes of LED intensity. The LED stimulus has the advantage of being easily controlled but the disadvantage of synchronously stimulating all the illuminated pixels, limiting its practical application to a few hundred pixels and even less if response timing jitter is being measured. Making a small LED flashing spot on the array is not so simple however, especially if the surrounding pixels must be uniformly illuminated by steady light intensity to reduce their dark-current noise-generated activity, which can be significant especially when the DVS thresholds are set low. Therefore, this method is probably best used with sensors that can limit the pixel activity to a controlled ROI.

It may also be possible to use a display to generate the stimulus, but caution should be used to ensure the screen is not flickering, because most screens control backlight intensity using duty-cycle modulation of the backlight. Turning up the display brightness to maximum often turns off the PWM backlight brightness control, reducing the display flicker significantly. The display latency and pixel synchronicity should also be carefully investigated before quantitative conclusions are drawn from using a computer display for stimulating pixels.

3.7.2 DVS Background Activity

Event-based sensors may produce some level of background activity that is not related to the scene. As an example from the DVS pixel, the reset switch junction leakage also produces background events, in this design only ON events. These background events become significant at low frequencies and contribute to the source of noise. The DVS of Lichtsteiner et al. (2008) has a background rate of about 0.05 Hz at room temperature. Since charge input to the floating input node V_f of the differencing amplifier in Figure 3.2 appears at the output V_{diff} as a voltage across C_2 , the rate of background activity events R_{leak} is related to leakage current I_{leak} , threshold voltage θ , and feedback capacitor C_2 by Eq. (3.13):

$$R_{\text{leak}} = \frac{I_{\text{leak}}}{\theta C_2}. \quad (3.13)$$

In most applications this background activity is easily filtered out by the simple digital filter described in Section 15.4.1 that discards events that are not correlated either in space or time with past events.

3.7.3 DVS Dynamic Range

Another important metric for all vision sensors is DR, defined as the ratio of maximum to minimum illumination over which the sensor generates useful output. As an example, for the

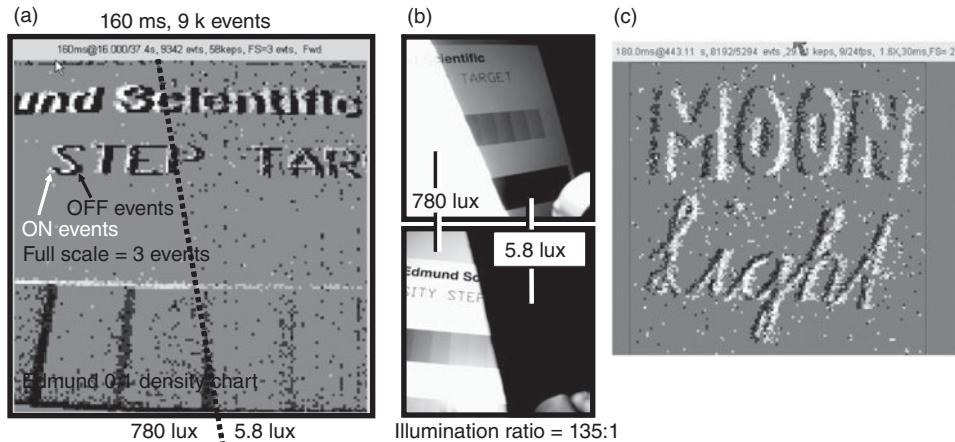


Figure 3.17 Illustration of DVS DR capabilities. (a) Histogrammed output from the vision sensor viewing an Edmund density step chart with a high contrast shadow cast on it. (b) The same scene as photographed by a Nikon 995 digital camera to expose the two halves of the scene. (c) Moving black text on white background under 3/4 moon (<0.1 lux) illumination (180 ms, 8000 events). © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

DVS the DR is defined as the ratio of maximum to minimum scene illumination at which events can be generated by high contrast stimuli. This range is illustrated in Figures 3.17a–3.17c. In the DVS, this large range arises from the logarithmic compression in the front-end photoreceptor circuit and the local event-based quantization. Photodiode dark current of 4 fA at room temperature limits the lower end of the range. (In the DVS, it is possible to measure the dark current from the global ΣI current divided by the number of pixels.) Events are generated down to less than 0.1 lux scene illumination using a fast f/1.2 lens (Figure 3.17c). At this illumination level, the signal (photocurrent induced by photons from the scene) is only a small fraction of the noise (background dark current). Operation at this low SNR is possible because the low threshold mismatch allows setting a threshold that is low enough to detect the reduced photocurrent TC. The sensor also operates up to bright sunlight scene illumination of 100 klux; the total DR amounts to about 6 decades, or 120 dB. The vision sensor is usable for typical scene contrast under nighttime street lighting of a few lux. The DR is halved approximately every 8°C increase in temperature. Another relevant metric is the intra-scene DR, which is defined as the range of illumination within a scene over which the sensor is usable. This range could be lower than the total DR if the sensor uses global gain control to center the photodetector operating point.

3.7.4 DVS Latency and Jitter

The latency and precision of timing of event-based sensor output are important metrics because they define speed of response and the analog quantity (inter-event time) being represented. The pixel event latency depends on the pixel latency and the latency of the peripheral AER

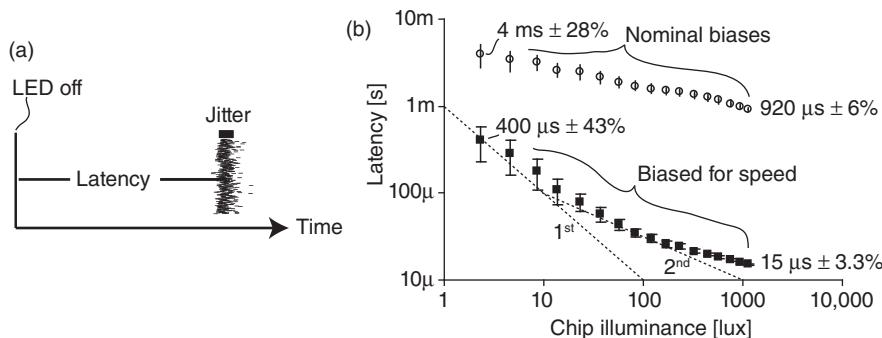


Figure 3.18 DVS latency and latency jitter (error bars) versus illumination in response to a 30% contrast periodic 10 Hz step stimulus of a single pixel illumination. (a) Measurement of repeated single OFF event responses to the step. (b) Results with two bias settings as a function of chip illuminance. © 2008 IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

circuits. The latency jitter depends on the jitter of the event generation within the pixel and the variance in the transmission time of the event.

As an example from the DVS, a basic prediction in Lichtsteiner et al. (2008) is that the increase in latency should be proportional to reciprocal illumination. The latency is typically measured using a small spot covering a few pixels of a low-contrast periodic step stimulus while varying DC illuminance (Figure 3.18). The thresholds are set to produce about one event of each polarity per stimulus cycle. The overall latency is plotted versus stimulus chip illuminance for two sets of measurements, one at the nominal biases used for many applications, the other at higher current levels for the front-end photoreceptor biases. The plots show the measured latency and the $1-\sigma$ response jitter. The dashed lines show a reciprocal (first-order) and reciprocal-square-root (second-order) relationship between latency and illumination. As can be seen, latencies are on the order of milliseconds and follow either inverse linear or inverse square root characteristics with intensity depending on photoreceptor biasing. An often quoted metric (with unproven relevance) is the minimum latency, which is obtained under very high illumination and biasing conditions.

3.8 Discussion

This chapter focused on discussion of recent silicon retina designs and their specifications. But there are many areas of possible improvement and innovation in the design of silicon retinas.

The event-based silicon retinas in this chapter offer either spatial or temporal processing and none of them offer both in a form usable for real-world applications. Although the ATIS discussed here offers an intensity output, this output has its own problems, because it has large motion artifacts. Perhaps a new generation of so-called DAVIS silicon retinas that combine DVS and APS technologies in the same pixels (Berner et al. 2013) can offer output that combines the strengths of conventional machine vision based on tiny APS pixels with the strengths of low-latency, sparse output neuromorphic event-based vision. But here the challenge will

be to improve the performance of the DAVIS APS output to keep up with conventional APS technology, which is continually being evolved in a competitive commercial market.

Another example is that no one has thus far built a high-performance color silicon retina, although color is a basic feature of biological vision in all diurnal animals. A few recent attempts in this direction have all attempted to use buried double (Berner and Delbrück 2011; Berner et al. 2008; Fasnacht and Delbrück 2007) or triple (Leñero Bardallo et al. 2011a) junctions to separate the colors, as was pioneered by Dick Merrill and others at Foveon Inc. (Gilder 2005). But the usability and performance of these neuromorphic prototypes fall far short of those afforded by commodity CMOS color cameras. This is likely due mostly to the poor color separation properties of standard CMOS implants; Foveon used a modified process with special implants to create buried junctions at optimal depths. Mostly, however, the development of color silicon retinas has been held back by the expense of using the process technologies that provide integrated color filters.

Another example is that implementations of vision sensors with space-varying resolution are in their infancy (Azadmehr et al. 2005) and have not been convincingly demonstrated as useful, although all animals vary the spatial acuity and characteristics of their visual system with eccentricity from the fovea. This could be the symptom of fundamental technological differences between electronics and the ionic basis for biological neural computation. The ability to electronically steer a focus of attention over a sensor with a large number of inexpensive pixels much more cheaply than building a mechanical eye could mean that foveated cameras would find a limited niche commercially.

The rather poor quantum efficiencies and fill factors of silicon retinas are also a consequence of using standard CMOS technologies. One solution is the use of integrated microlenses, which concentrate light onto the photodiode. However, standard microlenses offered in CIS processes are optimized for pixels smaller than 5 μm , so for 20 μm retina pixels they do not offer any benefit. Another possible improvement could come from back-side illumination (BSI). Normally a vision sensor is illuminated from the top of the wafer. However, for tiny-pixel CMOS imagers front-side illumination (FSI) is a big problem, because the photodiode sits at the bottom of a tunnel through all the overlaying metal and insulator layers, making it difficult to capture light, particularly at the edges of the sensor when using a wide angle lens. This problem led the development of BSI, where the wafer is thinned down to less than 20 μm and is illuminated from the back rather than the front. Now all the silicon area can receive light and if properly designed, most of the photocharge is collected by the photodiode. Intense development of BSI image sensors by industry may shortly make this technology more accessible for prototyping. But now new problems can arise, such as unwanted ‘parasitic photocurrents’ in junctions other than the photodiode. These currents can disturb the pixel operation, particularly when the pixel stores a charge on a capacitor like a global-shutter CMOS imager or the DVS pixel.

There are other areas of improvement toward mimicking the features of biological retinas. Mammalian retinas have tens of parallel output channels with different and complex nonlinear signal-processing characteristics. We hope that biologists will understand more about the functional roles of these parallel information channels, which will motivate their implementation either in the focal plane or in post-processing vision sensor output.

Even considering the input photoreceptors, there are opportunities: a cone photoreceptor isolated in a dish is an example of a distributed amplifier with interesting dynamical properties (Sarpeshkar 2010; Shapley and Enroth-Cugell 1984). Because the gain is generated by a chain

of biochemical amplifiers, it can be varied by many orders of magnitude with only a small effect on the overall bandwidth or latency. Right now, the way we build logarithmic photoreceptors using a single stage of amplification, the gain-bandwidth product is constant: if the light becomes 10 times dimmer, the photoreceptor becomes 10 times slower. On the one hand this is good because it helps control shot noise. But on the other hand, it means that logarithmic silicon retina photoreceptor dynamics can change over many orders of magnitude depending on the illumination. Perhaps one possible future solution is to build sensors that emulate the rod-cone systems of biology, where some photoreceptors are optimized for low light levels and others for high levels. Then the subsequent processing circuits can be shared between these detectors as has been recently shown to occur in the mouse retina (Farrow et al. 2013).

The retina designs discussed here are related to each other historically (Figure 3.19) and they illustrate the trade-offs that designers face in creating usable designs for the end user and the difficulty of incorporating multiple retina functionalities within the focal plane. The design style in the Parvo-Magno retina – which uses many current mirrors – led to large mismatch. The circuits can be calibrated to reduce the amount of offset in the responses but the calibration circuits themselves occupy a large percentage of the area in the pixel (see, e.g., Costas-Santos et al. 2007; Linares-Barranco et al. 2003).

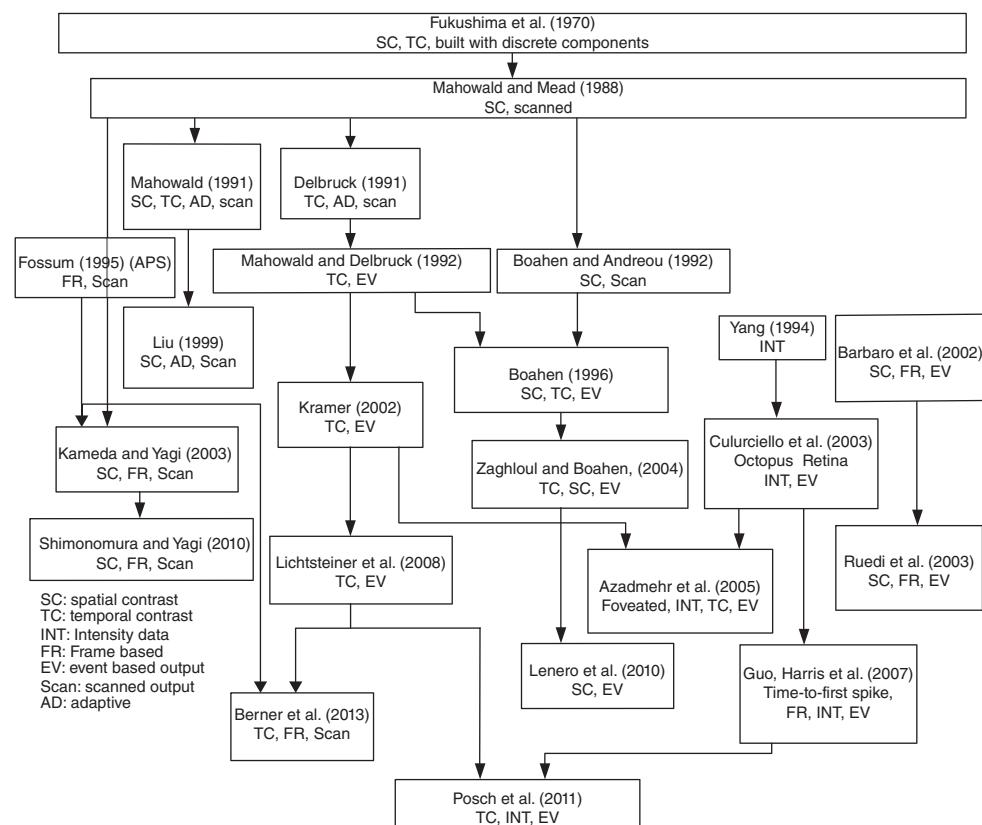


Figure 3.19 Tree diagram of silicon retina designs

In general, the easiest way to improve precision is to ensure that the signal is amplified more than the mismatch. This method is used in the DVS, the ATIS, and the VISe. In the DVS, the signal (change of log intensity) is amplified by the differencing amplifier before it is compared to the thresholds. That way, the comparator mismatch is effectively reduced by a factor of the amplifier gain. In the ATIS and VISe, the signal (intensity or SC) is amplified to a large fraction of the power supply voltage before it is compared to the threshold. Moreover, in the ATIS, the same comparator is used for detecting two different, globally identical threshold levels. This way, the comparator mismatch is canceled.

The retina implementations described in this chapter show the variety of approaches taken toward building high-quality AER vision sensors that can now be used for solving practical machine vision problems. There has been a lot of recent progress using an event-based, data-driven, redundancy-reducing style of computation that underlies the power of biological vision. Chapter 13 discusses examples of multichip AER systems that combine AER retinas with other AER chips. Chapter 15 discusses how AER sensor output are processed algorithmically, by programs running on digital computers, to decrease computational cost and system latency by taking advantage of the sparsity of the events and their precise timing. The use of timing is even more critical in auditory processing, as will be introduced in Chapter 4.

References

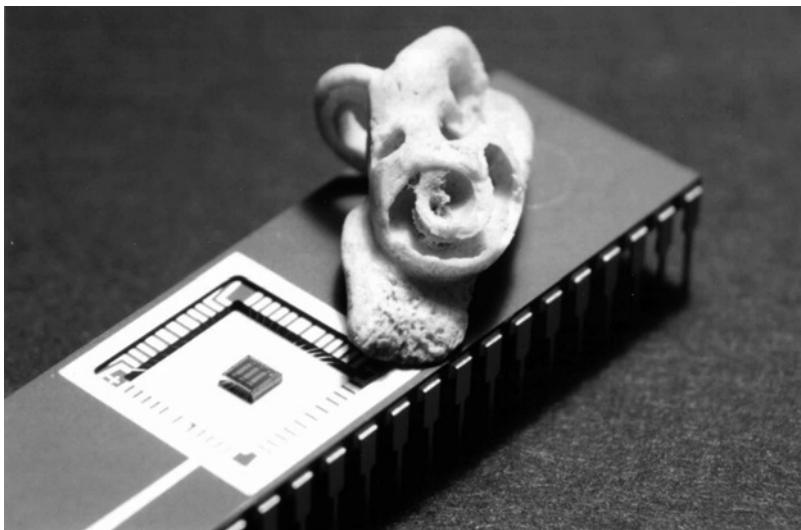
- Azadmehr M, Abrahamsen JP, and Hafliger P. 2005. A foveated AER imager chip. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 2751–2754.
- Barbaro M, Burgi P, Mortara R, Nussbaum P, and Heitger F. 2002. A 100×100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output coding. *IEEE J. Solid-State Circuits* **37**(2), 160–172.
- Barlow HB. 1961. Possible principles underlying the transformation of sensory messages. In: *Sensory Communication* (ed. Rosenblith WA). MIT Press, Cambridge, MA. pp. 217–234.
- Belbachir AN, Litzenberger M, Schraml S, Hofstatter M, Bauer D, Schon P, Humenberger M, Sulzbachner C, Lunden T, and Merne M. 2012. CARE: a dynamic stereo vision sensor system for fall detection. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 731–734.
- Berner R and Delbrück T. 2011. Event-based pixel sensitive to changes of color and brightness. *IEEE Trans. Circuits Syst. I: Regular Papers* **58**(7), 1581–1590.
- Berner R, Lichtsteiner P, and Delbrück T. 2008. Self-timed vertacolor dichromatic vision sensor for low power pattern detection. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1032–1035.
- Berner R, Brandli C, Yang MH, Liu SC, and Delbrück T. 2013. A 240×180 10 mW 12 μ s latency sparse-output vision sensor for mobile applications. *Proc. Symp. VLSI Circuits*, pp. C186–C187.
- Boahen KA. 1996. A retinomorphic vision system. *IEEE Micro* **16**(5), 30–39.
- Boahen KA and Andreou AG. 1992. A contrast sensitive silicon retina with reciprocal synapses. In: *Advances in Neural Information Processing Systems 4 (NIPS)* (eds. Moody, JE, Hanson, SJ, and Lippmann, RP). Morgan-Kaufmann, San Mateo, CA. pp. 764–772.
- Conradt J, Berner R, Cook M, and Delbrück T. 2009a. An embedded AER dynamic vision sensor for low-latency pole balancing. *Proc. 12th IEEE Int. Conf. Computer Vision Workshops (ICCV)*, 780–785.
- Conradt J, Cook M, Berner R, Lichtsteiner P, Douglas RJ, and Delbrück T. 2009b. A pencil balancing robot using a pair of AER dynamic vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 781–784.
- Costas-Santos J, Serrano-Gotarredona T, Serrano-Gotarredona R, and Linares-Barranco B. 2007. A spatial contrast retina with on-chip calibration for neuromorphic spike-based AER vision systems. *IEEE Trans. Circuits Syst. I* **54**(7), 1444–1458.
- Culurciello E, Etienne-Cummings R, and Boahen K. 2003. A biomorphic digital image sensor. *IEEE J. Solid-State Circuits* **38**(2), 281–294.

- Delbrück T and Lichtsteiner P. 2007. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 845–848.
- Delbrück T and Mead CA. 1994. Adaptive photoreceptor circuit with wide dynamic range. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **4**, pp. 339–342.
- Delbrück T, Linares-Barranco B, Culurciello E, and Posch C. 2010. Activity-driven, event-based vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2426–2429.
- Drazen D, Lichtsteiner P, Hafliger P, Delbrück T, and Jensen A. 2011. Toward real-time particle tracking using an event-based dynamic vision sensor. *Exp. Fluids* **51**(55), 1465–1469.
- Farrow K, Teixeira M, Szikra T, Viney TJ, Balint K, Yonehara K, and Roska B. 2013. Ambient illumination toggles a neuronal circuit switch in the retina and visual perception at cone threshold. *Neuron* **78**(2), 325–338.
- Fasnacht DB and Delbrück T. 2007. Dichromatic spectral measurement circuit in vanilla CMOS. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3091–3094.
- Fossum ER. 1995. CMOS image sensors: electronic camera on a chip. *Intl. Electron Devices Meeting*. IEEE, Washington, DC, pp. 17–25.
- Fukushima K, Yamaguchi Y, Yasuda M, and Nagata S. 1970. An electronic model of the retina. *Proc. IEEE* **58**(12), 1950–1951.
- Gilder G. 2005. *The Silicon Eye: How a Silicon Valley Company Aims to Make all Current Computers, Cameras, and Cell Phones Obsolete*. WW Norton & Company, New York.
- jaER. 2007. jaER Open Source Project, <http://jaerproject.org> (accessed July 28, 2014).
- Kameda S and Yagi T. 2003. An analog VLSI chip emulating sustained and transient response channels of the vertebrate retina. *IEEE Trans. Neural Netw.* **15**(5), 1405–1412.
- Kavadias S, Dierickx B, Scheffer D, Alaerts A, Uwaerts D, and Bogaerts J. 2000. A logarithmic response CMOS image sensor with on-chip calibration. *IEEE J. Solid-State Circuits* **35**(8), 1146–1152.
- Kramer J. 2002. An integrated optical transient sensor. *IEEE Trans. Circuits Syst. II* **49**(9), 612–628.
- Kuffler SW. 1953. Discharge patterns and functional organization of mammalian retina. *J. Neurophys.* **16**(1), 37–68.
- Lee J, Delbrück T, Park PKJ, Pfeiffer M, Shin CW, Ryu H, and Kang BC. 2012. Live demonstration: gesture-based remote control using stereo pair of dynamic vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 741–745.
- Leñero Bardallo JA, Bryn DH, and Hafliger P. 2011a. Bio-inspired asynchronous pixel event tri-color vision sensor. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 253–256.
- Leñero Bardallo JA, Serrano-Gotarredona T, and Linares-Barranco B. 2011b. A 3.6 μ s latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE J. Solid-State Circuits* **46**(6), 1443–1455.
- Lichtsteiner P, Delbrück T, and Kramer J. 2004. Improved on/off temporally differentiating address-event imager. *Proc. 11th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 211–214.
- Lichtsteiner P, Posch C, and Delbrück T. 2008. A 128×128 120 dB 15 μ s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* **43**(2), 566–576.
- Linares-Barranco B, Serrano-Gotarredona T, and Serrano-Gotarredona R. 2003. Compact low-power calibration mini-DACs for neural massive arrays with programmable weights. *IEEE Trans. Neural Netw.* **14**(5), 1207–1216.
- Litzenberger M, Kohn B, Belbachir AN, Donath N, Gritsch G, Garn H, Posch C, and Schraml S. 2006. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. *Proc. 2006 IEEE Intell. Transp. Syst. Conf. (ITSC)*, pp. 653–658.
- Liu SC. 1999. Silicon retina with adaptive filtering properties. *Analog Integr. Circuits Signal Process.* **18**(2/3), 243–254.
- Loose M, Meier K, and Schemmel J. 2001. A self-calibrating single-chip CMOS camera with logarithmic response. *IEEE J. Solid-State Circuits* **36**(4), 586–596.
- Luo Q and Harris J. 2004. A time-based CMOS image sensor. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **4**, pp. 840–843.
- Luo Q, Harris J, and Chen ZJ. 2006. A time-to-first spike CMOS image sensor with coarse temporal sampling. *Analog Integr. Circuits Signal Process.* **47**(3), 303–313.
- Mahowald M. 1991. Silicon retina with adaptive photoreceptors. *SPIE/SPSE Symp. Electronic Sci. Technol.: From Neurons to Chips* **1473**, 52–58.
- Mahowald M. 1994. *An Analog VLSI System for Stereoscopic Vision*. Kluwer Academic, Boston.
- Mahowald M and Mead C. 1991. The silicon retina. *Scientific American* **264**(5), 76–82.
- Mallik U, Vogelstein RJ, Culurciello E, Etienne-Cummings R, and Cauwenberghs G. 2005. A real-time spike-domain sensory information processing system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 1919–1923.

- Masland R. 2001. The fundamental plan of the retina. *Nat. Neurosci.* **4**(9), 877–886.
- Matolin D, Posch C, and Wohlgemant R. 2009. True correlated double sampling and comparator design for time-based image sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1269–1272.
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Mortara A, Vittoz EA, and Venier P. 1995. A communication scheme for analog VLSI perceptive systems. *IEEE J. Solid-State Circuits* **30**(6), 660–669.
- Ni Z, Pacoret C, Benosman R, Ieng S, and Regnier S. 2012. Asynchronous event-based high speed vision for microparticle tracking. *J. Microscopy* **245**(3), 236–244.
- Posch C and Matolin D. 2011. Sensitivity and uniformity of a 0.18 μm CMOS temporal contrast pixel array. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1572–1575.
- Posch C, Matolin D, and Wohlgemant R. 2011. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid-State Circuits* **46**(1), 259–275.
- Qi X, Guo X, and Harris J. 2004. A time-to-first-spike CMOS imager. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **4**, pp. 824–827.
- Rodieck RW. 1988. The primate retina. In: *Comparative Primate Biology* (eds Steklis HD and Erwin J). vol. 4. Alan R. Liss, New York. pp. 203–278.
- Rogister P, Benosman R, Leng S, Lichtsteiner P, and Delbrück T. 2012. Asynchronous event-based binocular stereo matching. *IEEE Trans. Neural Netw. Learning Syst.* **23**(2), 347–353.
- Ruedi PF, Heim P, Kaess F, Grenet E, Heitger F, Burgi PY, Gyger S, and Nussbaum P. 2003. A 128×128 , pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction. *IEEE J. Solid-State Circuits* **38**(12), 2325–2333.
- Ruedi PF, Heim P, Gyger S, Kaess F, Arm C, Caseiro R, Nagel JL, and Todeschini S. 2009. An SoC combining a 132 dB QVGA pixel array and a 32 b DSP/MCU processor for vision applications. *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, **1**, pp. 46–47.
- Sarpeshkar R. 2010. *Ultra Low Power Bioelectronics*. Cambridge University Press, Cambridge, UK.
- Schraml S, Belbachir AN, Milosevic N, and Schön P. 2010. Dynamic stereo vision system for real-time tracking. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1409–1412.
- Serrano-Gotarredona R, Oster M, Lichtsteiner P, Linares-Barranco A, Paz-Vicente R, Gomez-Rodriguez F, Camunas-Mesa L, Berner R, Rivas M, Delbrück T, Liu SC, Douglas R, Häfliger P, Jimenez-Moreno G, Civit A, Serrano-Gotarredona T, Acosta-Jimenez A and Linares-Barranco B. 2009. CAVIAR: a 45 K-neuron, 5 M-synapse, 12 G-connects/sec AER hardware sensory-processing-learning-actuating system for high speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* **20**(9), 1417–1438.
- Serrano-Gotarredona T and Linares-Barranco B. 2013. A 128×128 1.5% contrast sensitivity 0.9% FPN 3 μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE J. Solid-State Circuits* **48**(3), 827–838.
- Shapley R and Enroth-Cugell C. 1984. Visual adaptation and retinal gain controls. *Prog. Retin. Res.* **3**, 263–346.
- Shimonomura K and Yagi T. 2005. A 100×100 pixels orientation-selective multi-chip vision system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 1915–1918.
- Shoushun C and Bermak A. 2007. Arbitrated time-to-first spike CMOS image sensor with on-chip histogram equalization. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **15**(3), 346–357.
- Thorpe S, Fize D, and Marlot C. 1996. Speed of processing in the human visual system. *Nature* **381**(6582), 520–522.
- Werblin FS and Dowling JE. 1969. Organization of the retina of the mudpuppy *Necturus maculosus*: II. Intracellular recording. *J. Neurophys.* **32**(3), 339–355.
- Yang W. 1994. A wide dynamic range low power photosensor array *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, pp. 230–231.
- Zaghoul KA and Boahen KA. 2004a. Optic nerve signals in a neuromorphic chip I: Outer and inner retina models. *IEEE Trans. Biomed. Eng.* **51**(4), 657–666.
- Zaghoul KA and Boahen KA. 2004b. Optic nerve signals in a neuromorphic chip II: Testing and results. *IEEE Trans. Biomed. Eng.* **51**(4), 667–675.
- Zaghoul KA and Boahen KA. 2006. A silicon retina that reproduces signals in the optic nerve. *J. Neural Eng.* **3**(4), 257–267.

4

Silicon Cochleas



While the previous chapter was about neuromorphic silicon retinas, this one is on silicon cochleas. The cochlea is biology's sound sensor – it turns vibrations in the air into a neural signal. This chapter briefly explains the operation of the various components of the biological cochlea and introduces circuits that can simulate these components. Silicon cochlea designs typically divide the biological cochlea into several sections that are equally spaced along its length. Each section is then modeled by an electronic circuit. Silicon cochlea designs may be classified as 1D or 2D, depending on the coupling between these sections. Circuits for both variants are presented. They may also be

The figure shows a biological cochlea on top of a chip package. Reproduced with permission of Eric Fragnière.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

classified as active or passive, depending on whether the quality factor of each cochlear section changes as a function of the output signal of the section or not. Details for both implementations are presented. A tree diagram at the end of the chapter illustrates the progression of silicon cochlea modeling.

4.1 Introduction

The biological cochlea is a bony, fluid-filled, spiral structure that forms the majority of the inner ear. It performs the transduction between the pressure signal representing the acoustic input and the neural signals that carry information to the brain. Figure 4.1 shows the location of the cochlea relative to other key features in the human ear.

The cochlea is spiraled from the base (lowest turn) to the apex (highest turn) and contains approximately 2.5 turns (Figure 4.2). Inside, the cochlea is divided into three chambers (scalae): scala vestibuli, scala media, and scala tympani; Reissner's membrane separates the first from the second chamber and the basilar membrane (BM) separates the second and third chambers. The organ of Corti sits atop the BM. It contains both the inner and outer hair cells (IHCs and OHCs, respectively). The tips of these cells are hair-like stereocilia. Deflections in the stereocilia of the IHCs generate neural signals (afferents) that travel to the brain. Neural signals from the brain (efferents) can alter the length and width of the OHC cell bodies.

Figure 4.2 shows the simplified, uncoiled cochlea and its relationship to the rest of the auditory pathway. Here we see both the oval window, which is connected to the stapes (see also Figure 4.1), and the round window, which is a membrane that allows the pressure within the cochlea duct to be equalized. The fluid within the cochlea is assumed to be incompressible. When the oval window moves, both Reissner's membrane and the BM are deflected as a result, and the round window moves in a direction opposite to the initial movement in the oval window.

The BM changes both in width (illustrated in Figure 4.2) and stiffness, from narrow and stiff at the base to wide and flexible at the apex. These changes in the physical characteristics of the BM assist in the way in which the cochlea divides an input signal into its frequency components. At the base of the cochlea the physical characteristics of the BM are such that it responds better (i.e., greater movement is produced) to high-frequency stimuli, whereas the

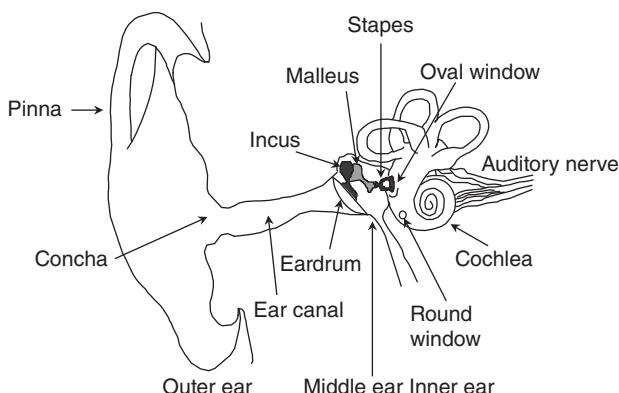


Figure 4.1 The human ear. Adapted from van Schaik (1998)

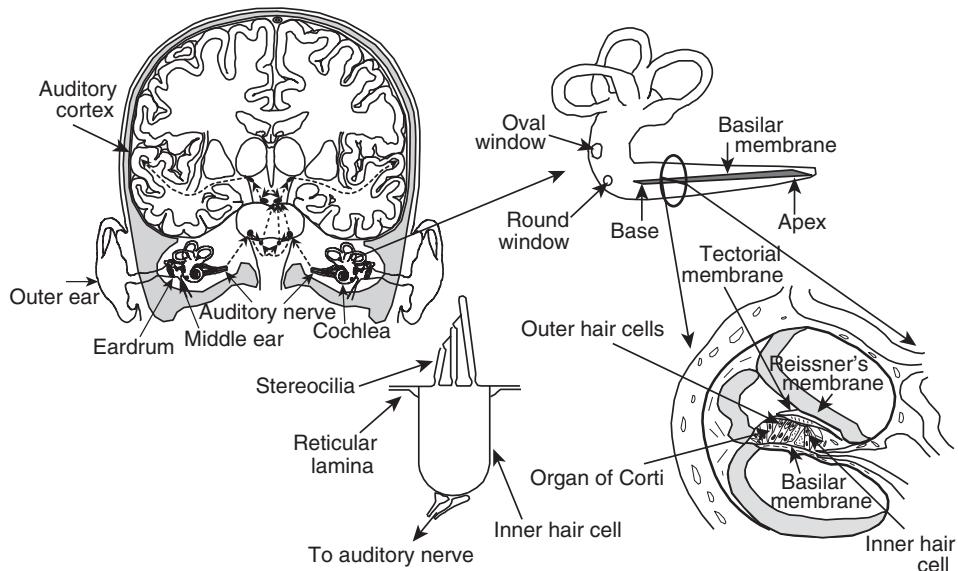


Figure 4.2 The auditory pathway. Adapted from van Schaik (1998)

apex responds better to low-frequency stimuli. The characteristic frequency at a particular place along the BM is defined to be the frequency that produces the greatest deflection at that place.

When the BM moves, the organ of Corti is displaced. The stereocilia of the OHCs, which are attached to the tectorial membrane, are displaced due to the shear between the reticular lamina (the rigid upper surface of the organ of Corti) and the tectorial membrane. The OHCs, which change their length as a function of their membrane potential, are thought to provide active undamping of the mechanical structure. This allows the selectivity of the cochlear filters to adapt as a function of the intensity of the input sound, as shown in Figure 4.4b. The stereocilia of the IHCs do not touch the tectorial membrane, but fit loosely into a raised groove known as Hensen's stripe on the lower surface of the tectorial membrane. When the reticular lamina moves with the BM, forces are exerted on the stereocilia, mainly due to the viscous drag of the cochlear fluid. The displacements of the IHC's stereocilia are thus proportional to the velocity of the BM motion. The membrane voltage of the IHC responds asymmetrically to the displacement of the stereocilia (Figure 4.3a) and the cell body functions as a low pass filter (Figure 4.3b). The IHCs also respond more strongly to the onset of a sound than to the sustained part of the sound (Figure 4.3c).

The cochlea has been the object of neuroscientific research for almost 150 years. The interest is mainly due to its huge dynamic range (approximately 120 dB) and its ability to adapt to a wide variety of listening environments. Researchers have also been building, improving, and studying silicon cochleas for over 20 years. The challenge and attraction of building silicon cochleas lies in the design and implementation of a complex signal processing system that follows the basic principles of biological cochleas. With the introduction of low-cost analog Very Large Scale Integration (VLSI) technology and the promise of being able to implement relatively complex signal processing systems with real-time operation, the first silicon cochleas were produced by Lyon and Mead (1988).

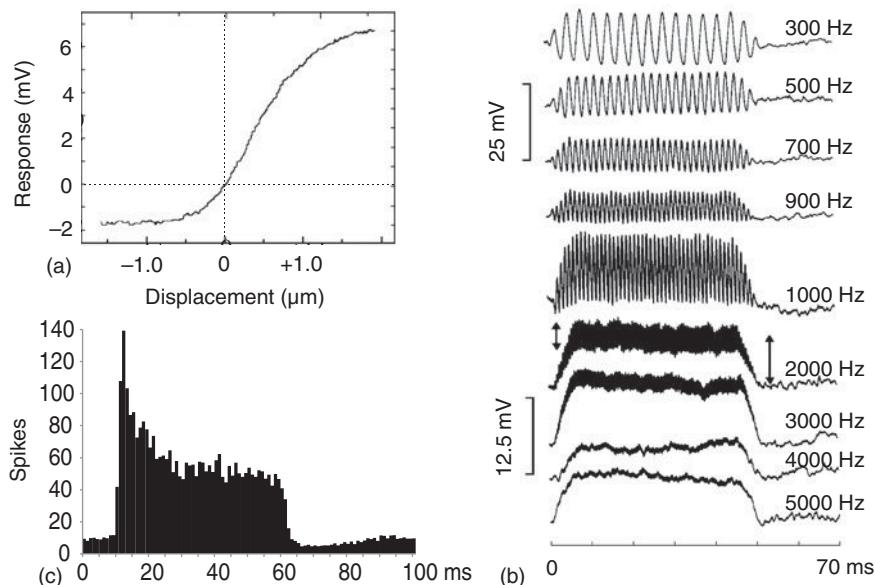


Figure 4.3 Inner hair cell response. (a) The IHC has an asymmetric saturating response to deflections of its stereocilia resulting in a weak form of half-wave rectification. Adapted from Hudspeth and Corey (1977) with permission of AJ Hudspeth. (b) As a result of this half-wave rectification, combined with the IHCs input resistance and membrane capacitance, the IHC's membrane potential will preserve the fine time structure of the BM vibration only at lower frequencies. At higher frequencies, the membrane potential is only a function of the amplitude of the BM vibration. Adapted from Palmer and Russell (1986), Copyright 1986, with permission from Elsevier. (c) Illustration of a typical peristimulus time histogram of auditory nerve spiking. The IHC releases neurotransmitter that causes the auditory nerve neurons to spike. Because the available neurotransmitter is reduced as it is being used, the response on the auditory nerve is stronger to the onset of a sound than to the sustained part of the sound

All silicon cochleas discretize the BM using a number of filters or resonators with an exponentially decreasing fundamental frequency (from base to apex). In general, we classify silicon cochleas based on the details of the coupling between the cochlear filter elements and whether or not the gain and frequency selectivity of the cochlear filters adapt dynamically to changes in input intensity. Silicon cochleas may be classified as either 1D or 2D based on the coupling between the cochlear elements. One-dimensional silicon cochleas model the longitudinal wave propagation of the BM from base to apex, while 2D silicon cochleas model the wave propagation along the BM as well as through the fluid within the cochlear duct, taking both the longitudinal and vertical wave propagation into account. The systematic changes in the properties of the BM with longitudinal position, such as stiffness and width, are generally modeled by systematic changes in the parameters of the cochlear filter elements. It should also be noted that there are many other types of silicon cochleas that cannot be considered 1D or 2D but rather a combination of both.

The quality factor (Q) of a cochlear filter element is a measure of its frequency selectivity; it is defined as the bandwidth of the filter at half its maximum gain (i.e., 6 dB below maximum gain), divided by the frequency at which this maximum gain occurs. Silicon cochleas are

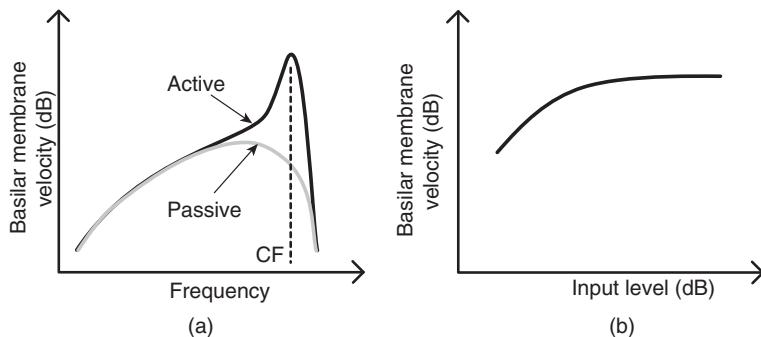


Figure 4.4 Nonlinear active properties of the BM. (a) Response of a point on the BM without the effect of OHCs (Passive) and with OHCs (Active). CF indicates the characteristic frequency of the BM section. (b) Response level as a function of input level at the characteristic frequency of the BM section

classified as active when the gain and/or quality factor of the cochlear filter elements change dynamically based on the intensity of the input, essentially increasing the gain and frequency tuning at low intensities and reducing these at high intensities. This active behavior essentially models the behavior of the OHCs which are believed to be involved with increasing sensitivity of the BM in the mammalian cochlea. Figure 4.4 shows the effect of the OHCs on the gain and tuning along the BM. The OHCs have little effect on frequencies well below the characteristic frequency of the BM section, but provide a significant gain around the characteristic frequency, particularly at low input levels (Figure 4.4a). At higher input levels, the effect of the OHCs vanishes and the response saturates (Figure 4.4b).

Ideally, the more features that a silicon cochlea has, the closer the results should match those from biology. However, as more features usually demand greater complexity and cost in the implementation, silicon cochleas are generally designed with only those features required for a specific application. Applications for silicon cochleas include speech processing, sound localization, and sound scene analysis, to name but a few. Experimenting with silicon cochleas in real time allows researchers to isolate individual components of the model and gain better insight as to how they work. In this regard, it is important in the design of silicon cochleas to use realistic biological models.

Despite 20 years of research there has yet to be developed a silicon cochlea that comes close to matching the power consumption, frequency range, input dynamic range, or noise immunity of the real biological cochlea. This needs to be put in perspective, however, considering that we are attempting to build and understand a complex system that has evolved over millions of years. In this chapter we will give a detailed description for the design and the circuits of the 1D (cascaded) and the 2D silicon cochlear architectures.

4.2 Cochlea Architectures

This section starts with the simplest architecture which consists of a cascade of filters. After discussing the advantages and shortcomings of 1D architectures, it then discusses 2D architectures that better model the interaction of the fluid with the BM. This discussion is continued

over details of the circuit elements of these architectures, concluding with attempts to model active nonlinear behavior.

4.2.1 Cascaded 1D

A rigorous explanation of the 1D cochlea model is given in Lyon and Mead (1988). However, a good understanding of Lyon and Mead's silicon cochlea and subsequent 1D silicon cochleas depends mainly on understanding the reasoning behind the use of a filter cascade to model the BM. As sound enters the cochlea it initiates a pressure wave which travels along the length of the BM from base to apex. The physical properties of the BM change from base to apex in such a way that the various frequency components of the pressure wave result in maximum displacement at varying positions along the BM. The design of the 1D silicon cochlea is based solely on the consideration of changes in the properties of the BM along its length. Specifically, measurements of the biological cochlea in a number of animals, including humans, have shown that the characteristic frequency along the BM is exponentially decreasing, that is, it is linear on a logarithmic scale: high frequencies at the base to low frequencies at its apex. To create the silicon cochlea, the BM is discretized into sections of equal length, Δx , and the array of sections is subsequently modeled by a cascade of filters with a characteristic frequency, or rather its inverse, a time constant, τ , that scales according to the characteristic frequency of each BM segment. Each filter that comprises the cascade is identical, except for its characteristic frequency, and has a low-pass or band-pass characteristic, such that the higher frequency signal components are selectively filtered out as the signal travels through the cascade. This removal of high-frequency components results in a steep roll-off in the response curves at the output of each filter after the characteristic frequency. This steep roll-off in the high-frequency signal components is seen in the biological cochlea as the pressure wave travels along the BM. Thus, despite its gross simplifications, this model provides a reasonable first-order approximation of the signal processing within the biological cochlea. Figure 4.5 shows a top-level schematic of the 1D cascade model along with a representation of the output of a single filter section. Here

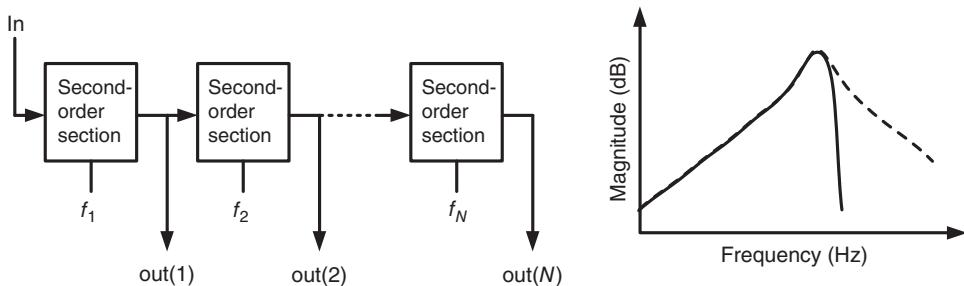


Figure 4.5 The 1D cascade cochlea model (left) with N filter sections and the output of one its filter section (right, solid line) when compared with the output of a single second-order section (right, dashed line). In the cascade model, the input (In) goes only to the first section, and the respective outputs of each filter section, $\text{out}(i)$, drive the input of the next section. The sections have individual best characteristic frequencies as indicated by f_i .

the steep roll-off after the characteristic frequency of the filter (solid line) is compared to the slope of a stand-alone second-order filter section (dashed line).

4.2.2 Basic 1D Silicon Cochlea

The 1D cascaded cochlea model in Figure 4.5 is realized using second-order sections (SOS) described in Chapter 9, and biasing such that each SOS has an exponentially decreasing frequency. The floor plan for an 1D silicon cochlea from Lyon and Mead (1988) is reproduced in Figure 4.6. Here the input signal snakes through the filter sections from In to Out. The τ - and Q -lines set the time constants and the quality factors of the SOS respectively, and outputs are taken after every 10 filters using a simple buffer. As the input signal travels through the cascade of SOS, the high-frequency components of the signal are selectively filtered out, with the higher frequency components only present at the start of the cascade and only the lower frequency components remaining toward the end of the cascade. The frequency-gain curves of a filter section in the cascade display the characteristic steep cut-off slope above the section's characteristic frequency due to the successive removal of the high-frequency components by the filters before it. This is not the case for the first few sections, as the suppression of high-frequency components has not yet accumulated. Measurements of the output gain for two output taps of the Lyon and Mead cochlea are shown for a number of frequencies in Figure 4.7. In this figure, the measurement data are shown as small dots, while the continuous line shows the theoretical gain curve.

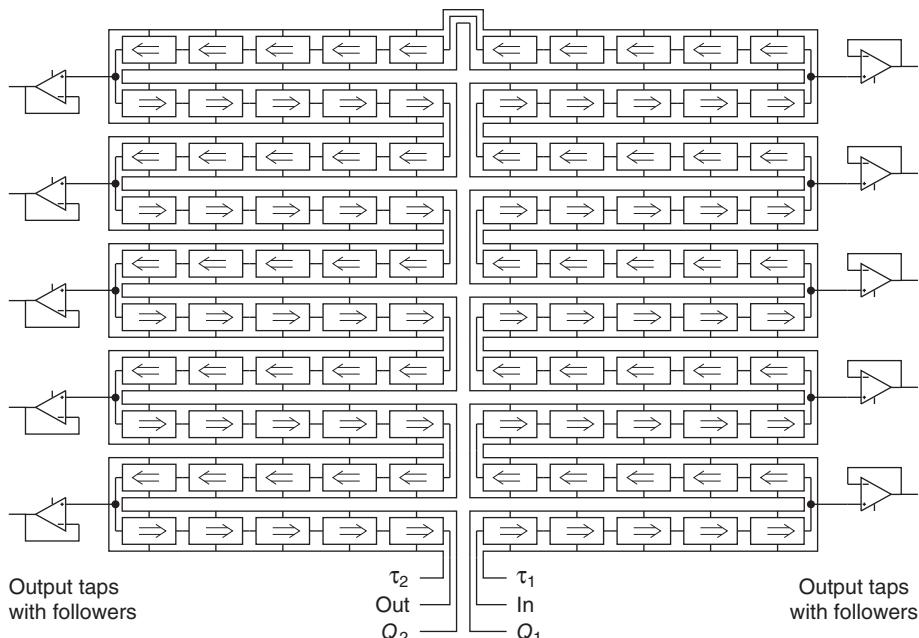


Figure 4.6 The floor plan for the silicon cochlea. © 1988 IEEE. Reprinted, with permission, from Lyon and Mead (1988)

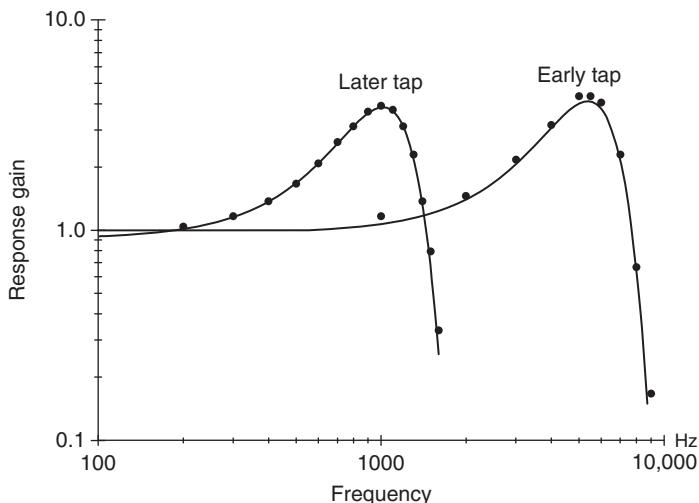


Figure 4.7 Measurement data (dots) and theoretical gain (solid line) from sections of the silicon cochlea. © 1988 IEEE. Reprinted, with permission, from Lyon and Mead (1988)

The design of 1D silicon cochleas has changed very little since its initial design. More recent designs include improvements to the linear range and stability of the SOS, improvements to the biasing scheme, and general improvements in matching. These enhancements are discussed in the following section.

Issues with the 1D Silicon Cochlea

One of the issues with the 1D cascade type of silicon cochlea is that if one of the SOSs in the cascade fails, all of the subsequent second-order stages become unusable although this is not usually observed in practice. Another problem with the 1D cascaded filter structure is that there is a limit to the number of stages that can be employed in practical real-time systems. This limit arises because as the number of filter stages increases, the delay through the cascade also increases, and at a certain point the latency of the output signal from the filter stages becomes too large. This problem can be mitigated by using a filter transfer function where this delay is reduced (Katsiamis et al. 2009). Additional issues with the 1D silicon cochlea include the limited dynamic range and large signal stability of the transconductance amplifiers and the accumulation of noise in the cascade. Silicon cochleas that consist of 1D parallel sections or a 2D structure circumvent some of the major drawbacks of the cascaded filter sections.

4.2.3 2D Architecture

In the 2D silicon cochlea, the number of cochlear filtering elements is not limited by the accumulated delay through the filters since the pressure signal is coupled through a model of

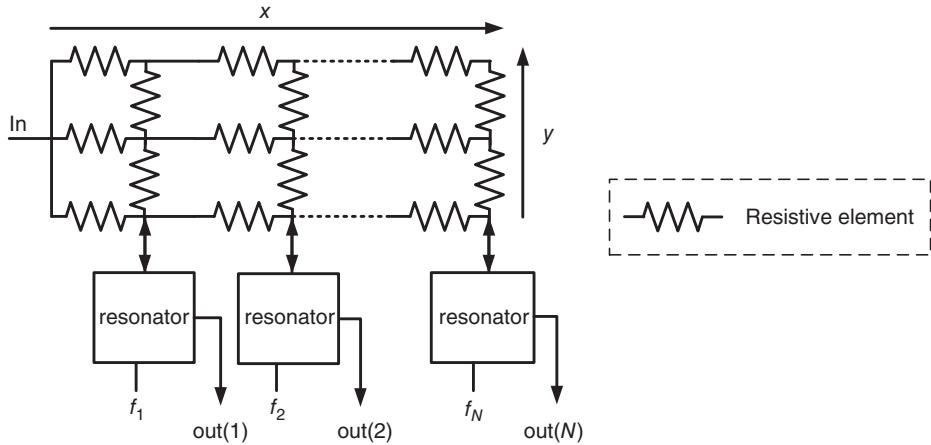


Figure 4.8 The schematic structure of the 2D silicon cochlea

the cochlear fluid rather than through a cascade of filters. This results in a silicon cochlea that is capable of being used in real-time applications for the entire frequency range of the cochlea. Second, errors in a single resonator/filter have little effect on the operation of the remainder of the cochlea. A schematic of the structure of the 2D silicon cochlea is presented in Figure 4.8.

4.2.4 The Resistive (Conductive) Network

In the 2D model, the fluid within the cochlea duct is assumed to be inviscid, incompressible, and irrotational. Under these conditions it was demonstrated in both Watts (1992) and Fragnière (1998) that a resistive (conductive) network could be used to model the fluid in the electrical domain; in effect, these resistive networks implement a finite difference approximation to the Laplace's equation describing fluid motion. In Watts (1992), the velocity potential, ϕ , of the fluid was demonstrated to be the analog of the voltage, V , in the resistive network. While in Fragnière (1998), the pressure of the fluid, ρ , was shown to be equivalent to the voltage, V , in the resistive network. These two derivations, while different, each produce 2D cochlea fluid models in silicon using a resistive network. It is, in fact, the fluid which is 2D in this model while the resonators that model the BM are tapped off from the resistive network which models it (see Figure 4.8).

In the resistive networks proposed in Watts (1992) and Fragnière (1998), only one scala is modeled, that is, the scala media which is above the BM. Their model is achieved assuming symmetry above and below the BM. At the cochlear apex (which is also the helicotrema in these models), the boundary conditions are specified such that it is a point of zero pressure. In the electrical domain, this is modeled by a capacitor to ground. In Wen and Boahen (2006b), both scalas (the scala media and the scala tympani) above and below the BM were modeled by resistive networks.

The acceleration of the fluid is assumed to be zero at the boundaries of the resistive network. The relationship between acceleration and pressure follows from Newton's second law and can be written as

$$\frac{\partial p(x, y)}{\partial x} = \rho(x, y)a_x(x, y), \quad \frac{\partial p(x, y)}{\partial y} = \rho(x, y)a_y(x, y), \quad (4.1)$$

where ρ is the fluid density, and $a_x(x, y)$ and $a_y(x, y)$ are the acceleration of the fluid in the x and y direction, respectively.

4.2.5 The BM Resonators

Following from the 2D model derivation in Fragnière (1998), the equation describing BM motion (or fluid motion) on the boundary of the fluid ($y = 0$) is given by

$$\Delta p_{\text{BM}}(x)w(x)dx = a_{\text{BM}}(x)\frac{S(x)}{s^2}dx + \frac{\beta(x)}{s}dx + M(x)dx, \quad (4.2)$$

where $p_{\text{BM}}(x)$ is the pressure difference across the BM (equivalent to $2p(x, 0)$ since only one scala is modeled), $w(x)$ is the width of the BM as a function of x (equivalent to the z dimension in the 3D model), $a_{\text{BM}}(x)$ is the acceleration of the BM, $S(x)$ is the membrane stiffness, $\beta(x)$ is the viscous loss term, and $M(x)$ is the membrane mass. The width, w , and stiffness, S , are modeled by logarithmically decreasing the characteristic frequency along the length of the BM, while the mass, M , and viscosity (or damping) term, β , are assumed constant in the passive model.

Substitution of voltage for pressure and current for acceleration gives a second-order equation which can be modeled in silicon by a filter/resonator circuit. This can be written in the Laplace domain as

$$\frac{V(x)}{I(x)}dx = \frac{S(x)}{s^2}dx + \frac{\beta(x)}{s}dx + M(x)dx = Z_m(x)dx, \quad (4.3)$$

where $I(x)$ is the current drawn by a resonator at position x and $Z_m(x)$ is the impedance looking into that resonator.

4.2.6 The 2D Silicon Cochlea Model

The implementation of 2D silicon cochlea designs has been carried out using voltage-mode circuits (Watts et al. 1992) and current-mode circuits (Fragnière 1998; Hamilton et al. 2008a; Shirashi 2004; van Schaik and Fragnière 2001; Wen and Boahen 2006b). This section describes the architecture for the most commonly used 2D silicon cochlea model which was first proposed in Fragnière (1998). In this model, pressure and voltage are mathematical analogs as are BM acceleration and current. A simple schematic of this model is shown in Figure 4.9. Detailed and systematic simulations using this model have demonstrated that increasing the height of the network at the basal end (to mirror biology) makes little difference in the simulation

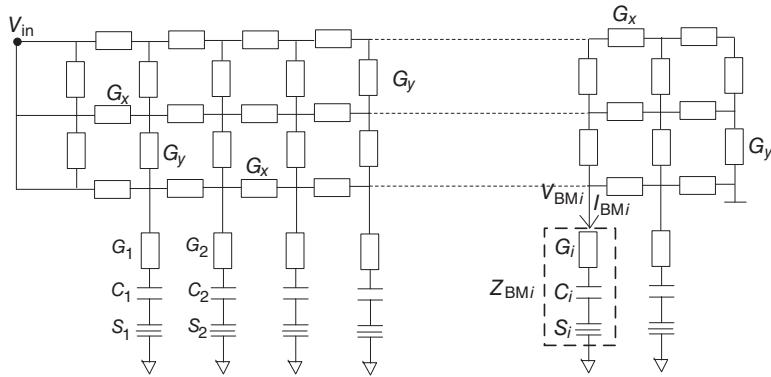


Figure 4.9 A simplified 2D cochlea model. © 2001 IEEE, reprinted, with permission, from van Schaik and Fragnière (2001)

outputs and that a constant height of two elements (resistors/conductors) provides a reasonable approximation.

Using this model, the fluid motion within the cochlea can be modeled using Eq. (4.3), where $V_{\text{BM}i}$ is the voltage analog of pressure, $I_{\text{BM}i}$ is the current analog of BM acceleration, G_x and G_y represent the conductance of the BM fluid in the x and y direction, respectively. The capacitance, C_i (where i is equivalent to a section, dx , of the BM after spatial quantization) is inversely proportional to the viscosity loss term $\beta(x)$; the super capacitance (frequency-dependent negative resistance), S_i , is inversely proportional to the stiffness, $S(x)$; and the conductance, G_i , is inversely proportional to the mass, $M(x)$, of the BM. Equation (4.3) can thus be re-written for a single resonator as

$$Z_{\text{BM}i} = \frac{V_{\text{BM}i}}{I_{\text{BM}i}} = \frac{1}{s^2 S_i} + \frac{1}{s C_i} + \frac{1}{G_i}. \quad (4.4)$$

The sensing cells in the cochlea, the IHCs, transduce the BM velocity into a neural signal. BM velocity is thus taken as the output for each resonator. Since the current, I , represents the acceleration of the BM, it must be integrated to obtain a representation of BM velocity. Integrating Eq. (4.4) with respect to I , we get

$$\frac{I_{\text{BM}i}}{s} = \frac{s S_i}{s^2 \frac{S_i}{G_i} + s \frac{S_i}{C_i} + 1} V_{\text{BM}i}. \quad (4.5)$$

The response of the circuit model described above to a given input signal closely matches the response of a passive biological cochlea in which the OHCs have been inhibited (Fragnière 1998; van Schaik and Fragnière 2001). The output from a software implementation of this model (Shirashi 2004) is shown in Figure 4.10. Here we see the response of the BM in response to the input being swept between 100 Hz and 10 kHz at discrete points along the BM. As with the 1D model we see the distinctive steep slope after the resonant (or characteristic) frequency.

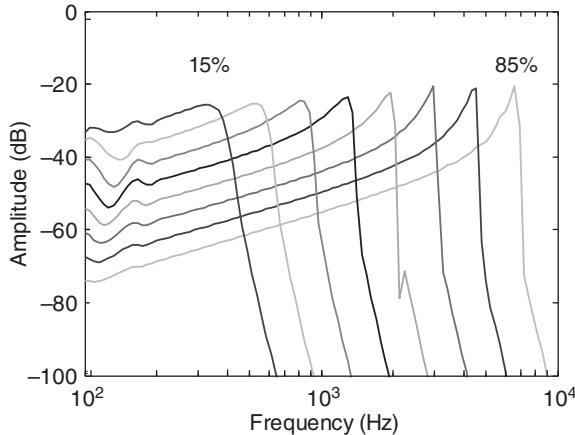


Figure 4.10 A software implementation of the 2D model showing the output, at discrete points along the BM in response to an input frequency sweep. The low-frequency ripple comes from a reflection of the input at the front of the helicotrema

We also see that the quality factor (Q) is greater at the higher frequencies. This trend is also seen in biological measurements.

4.2.7 Adding the Active Nonlinear Behavior of the OHCs

In order to obtain the active nonlinear behavior of the cochlea depicted in Figure 4.3, several different approaches can be taken. One is to use a fixed nonlinearity as in Sarpeshkar et al. (1996). Another is to model the mathematics of the micromechanics of the biological system as in Wen and Boahen (2006b). The most common way is to use automatic gain control (AGC). Typically AGC only includes gain changes with input signal changes; however, in most cochlea models this also includes changes in bandwidth with input signal changes, that is, the concept of automatic quality factor control (AQC). This idea was first proposed for use in a silicon cochlea in Lyon (1990) and has been used since in other silicon cochleas (Hamilton et al. 2008a, 2008b; Sarpeshkar et al. 1998; Summerfield and Lyon 1992).

Results

Figure 4.11 shows the output of a 2D silicon cochlea with AQC. When we compare this output with biological measurements from a Chinchilla cochlea (Fig. 1c of Ruggero 1992), we see that the resulting changes in BM velocity at a particular place along the BM respond with increasing gain and selectivity with decreases in input intensity. In Figure 4.11 we see that the output of the silicon cochlea does not extend to the same input dynamic range as that of the biological cochlea. It does, however, show similar key features such as a shift upwards in characteristic frequency with a decrease in input intensity. It also shows that the tuning of the resonator gets sharper as the input intensity gets weaker. Along with replicating the gain characteristics of the biological cochlea, this silicon cochlea also reproduces some of its defining nonlinear characteristics such as two-tone suppression and combinational tones. The two-tone suppression data from the 2D silicon cochlea with AQC (Figure 4.12) shows that

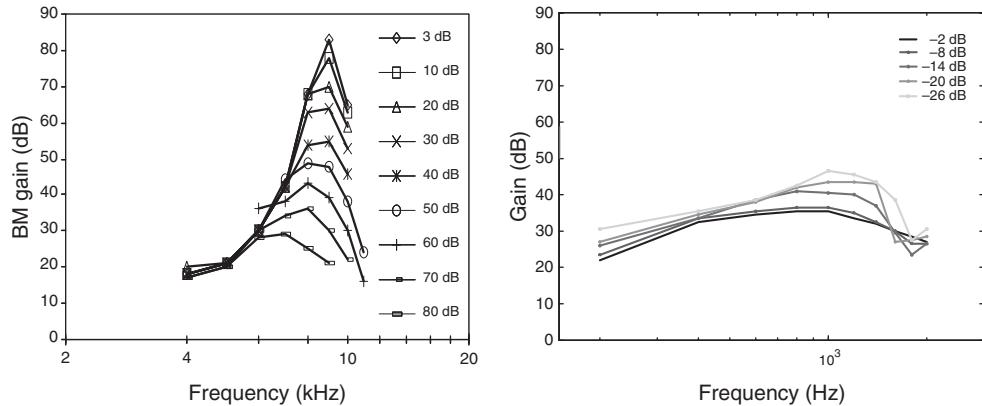


Figure 4.11 Comparison in gain between biology (left, reprinted from Ruggero 1992, copyright 1992, with permission from Elsevier) and the 2D silicon cochlea (right, © 2008 IEEE, reprinted, with permission, from Hamilton et al. 2008b)

the response of the silicon cochlea at the output of a resonator tuned to 1.3 kHz is suppressed in the presence of a nearby 1.8 kHz tone. Similar behavior is seen in measurements from the chinchilla cochlea (Fig. 3 of Ruggero et al. 1992). The nonlinear responses to combinational tones as shown in Figure 4.13 are also characteristic of similar measurements in the biological cochlea (see Fig. 1 of Robles et al. 1997).

4.3 Spike-Based Cochleas

Recent silicon cochlea designs include circuits for the BM, IHCs, and the spiral ganglion cells (Abdalla and Horiuchi 2005; Chan et al. 2007; Fragnière 2005; Liu et al. 2010b, 2014; Sarpeshkar et al. 2005; Wen and Boahen 2006a). The spike outputs are transmitted using the

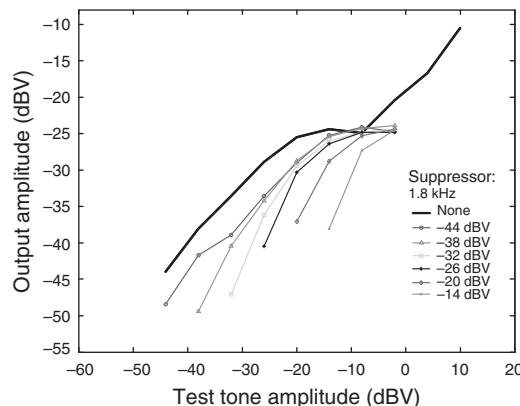


Figure 4.12 Two-tone suppression demonstrated by a resonator tuned to a resonant frequency of 1.3 kHz in the presence of a 1.8 kHz suppressor tone as measured from the 2D silicon cochlea © 2008 IEEE. Reprinted, with permission, from Hamilton et al. (2008b)

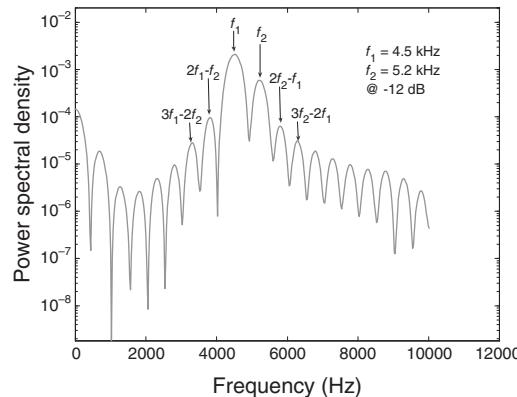


Figure 4.13 Frequency spectra at a place along the BM show odd-order distortion products from the 2D silicon cochlea © 2008 IEEE. Reprinted, with permission, from Hamilton et al. (2008b)

address-event representation (AER) protocol for spike transmission (discussed in Chapter 2). The latest cochlea implementation (AEREAR2) is a binaural cochlea intended for spatial audition and auditory scene analysis (Liu et al. 2010b). It integrates many features of former designs in a more user-friendly form. It uses cascaded SOSs (Figure 4.14) that drive IHCs, which in turn drive multiple ganglion cells with different spike thresholds. The resonance of individual sections can be adjusted by a local digital-to-analog converter (DAC) within each stage. This chip includes a variety of features including a matched binaural pair of cochleas, on-chip digitally controlled biases, on-chip microphone preamplifiers, and open-sourced host software APIs and algorithms (jAER 2007). A bus-powered USB board enables easy interfacing to standard PCs for control and processing (Figure 4.15). Figure 4.16 shows the raw PFM outputs of the 64 channels of the two cochleas in response to a frequency-swept chirp. All channels respond to only a limited frequency range of the chirp.

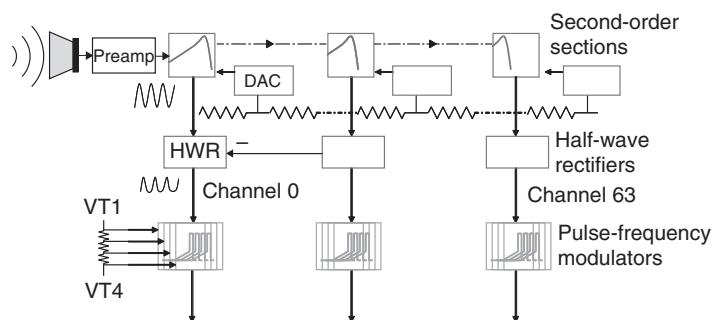


Figure 4.14 AEREAR2 cochlea architecture. © 2010 IEEE. Reprinted, with permission, from Liu et al. (2010b)

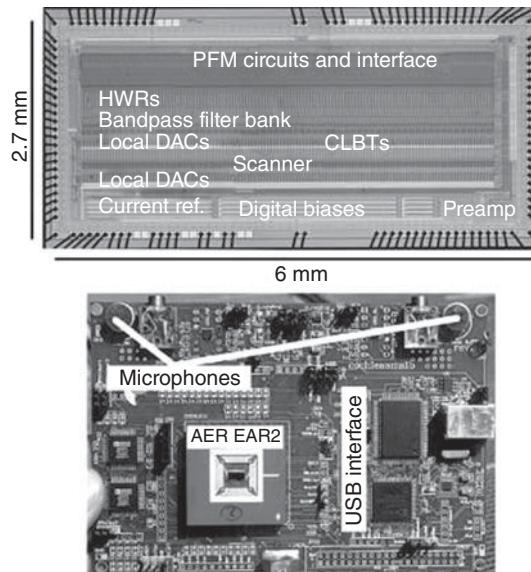


Figure 4.15 Prototype USB board with AEREAR2 cochlea, on-board microphones and microphone preamplifiers, and on-board digitally controlled biases. © 2010 IEEE. Reprinted, with permission, from Liu et al. (2010b)

4.3.1 Q-control of AEREAR2 Filters

The Q_s of the PFM outputs of the AEREAR2 can be sharpened by using the local DAC (Figure 4.17a) and by turning on the lateral inhibition from the nearest neighbor downstream (Figure 4.17b). The response of the filters can be further sharpened by subtracting the PFM responses of neighboring channels, since the filter responses have a steep roll-off.

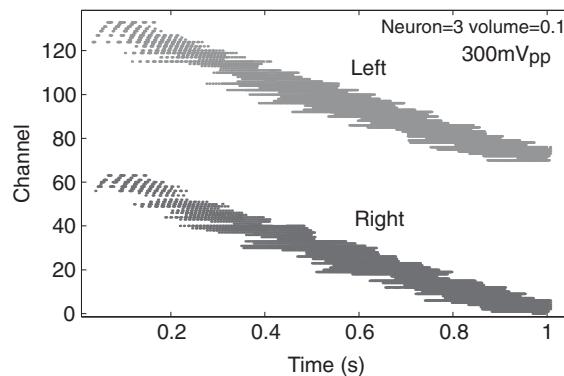


Figure 4.16 Event rasters recorded from the 64 channels of both ears. Frequency is logarithmically swept from 30 to 10 kHz with input amplitude 300 mVpp. © 2010 IEEE. Reprinted, with permission, from Liu et al. (2010b)

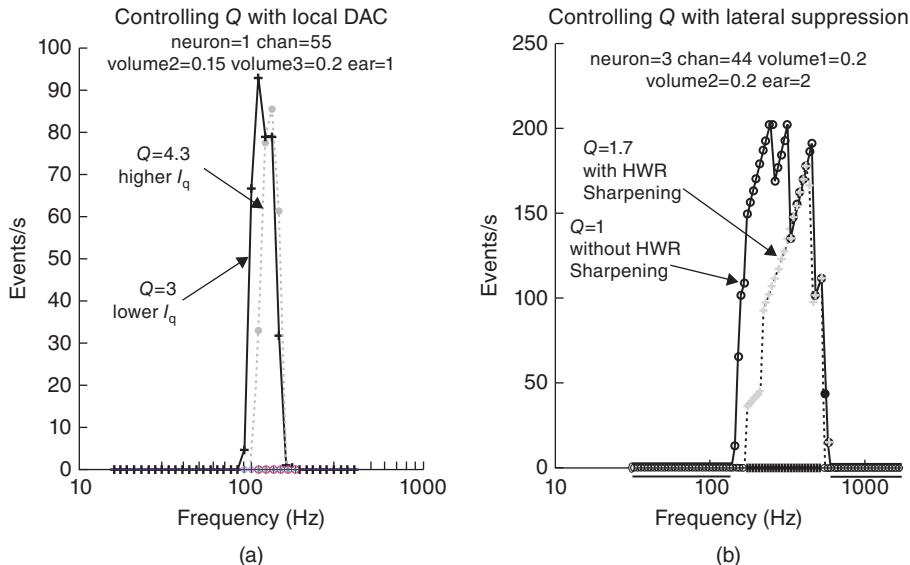


Figure 4.17 Two ways of increasing the Q of a filter. (a) Sharpening of frequency response of one channel as a function of two locally programmed Q settings. Input amplitude adjusted to obtain the same peak response rate. (b) Sharpening of response due to lateral suppression. Mean increase over all channels of Q is factor of 1.18. © 2010 IEEE. Reprinted, with permission, from Liu et al. (2010b)

4.3.2 Applications: Spike-Based Auditory Processing

The sparse asynchronous output from the silicon AER cochleas could reduce the postprocessing load when compared to conventional auditory signal processing which is based on regular sampling of the input signals at the Nyquist frequency. One reason for the reduction in computational load is because processing is not carried out until there are input events from the cochlea. Using an event-based output representation, timing information carried by the outputs of the AER cochleas can be used for inferring the location of an acoustic source, for example, through the interaural time differences (ITDs) between sounds arriving to the two ears (Chan et al. 2012, 2007; Finger and Liu 2011) or by emulating the echolocation mechanism of bats (Abdalla and Horiuchi, 2005).

The ITD cue is one of the three major cues used by animals for localizing a sound source. The other two major cues are interaural level differences and spectral differences. The ITD cue allows one to extract a sound source in the azimuth direction. The data in Figure 4.18 show how well the reconstructed ITD matches the actual source ITD illustrating that the AER spikes preserve timing information for the extraction of ITDs. Finger and Liu (2011) describes an algorithm that extracts the ITD information in real time by the use of a running histogram of ITDs. The resolution of the ITD information is similar to that extracted through conventional localization algorithms based on cross-correlation. See Section 15.4.5 for further discussion of this algorithm.

The timing information in the spike outputs has also been used to extract higher-level auditory features suitable for tasks such as harmonicity detection (Yu et al. 2009) and speaker

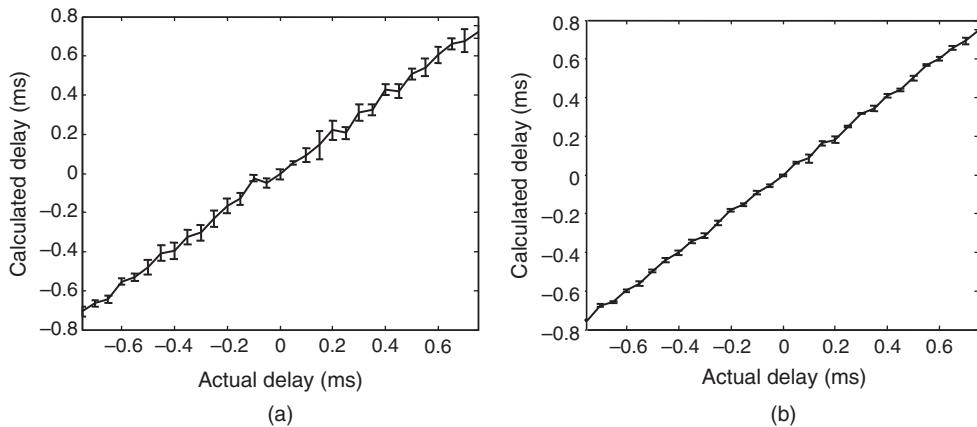


Figure 4.18 Calculated delay versus actual delay based on cross-correlation of spikes from left and right cochleas for (a) 500 Hz tone and (b) white noise. © 2007 IEEE. Reprinted, with permission, from Chan et al. (2007)

identification (Abdollahi and Liu 2011; Li et al. 2012; Liu et al. 2010a). From various feature representations such as the inter-spike interval and activity information, the authors obtained recognition performance of $>90\%$ over 40 speakers and with latencies of 700 ± 200 ms (Li et al. 2012).

4.4 Tree Diagram

Figure 4.19 shows a tree diagram illustrating the progression of silicon cochlea modeling since the days of Lyon and Mead's first silicon cochlea. The tree diagram is organized such that time increases vertically. The 1D cochleas are on the left, the 2D cochleas are on the right, and silicon cochleas that can be considered neither 1D nor 2D are shown in the middle. Silicon cochleas which possess the active nonlinear properties of the OHCs are shaded. All of the models presented here are for VLSI implementations of the cochlea except that from Leong et al. (2001), which use a Field Programmable Gate Array to implement the model, and Stoop et al. (2007), which is an electronic model built with discrete components with resonant structures that possess the dynamics of a Hopf bifurcation. The latter model suggests that the dynamics of the Hopf bifurcation (or something similar) drive the active nonlinear behavior of the cochlea.

4.5 Discussion

In this chapter we have described the various 1D and 2D silicon cochlea models. As our understanding of the biological cochlea has improved, these designs have utilized models that have gradually included more and more of the biological cochlea's impressive signal processing characteristics. Silicon cochleas are used for a variety of applications: from a tool for researchers to validate mathematical models of the biological cochlea in real time to

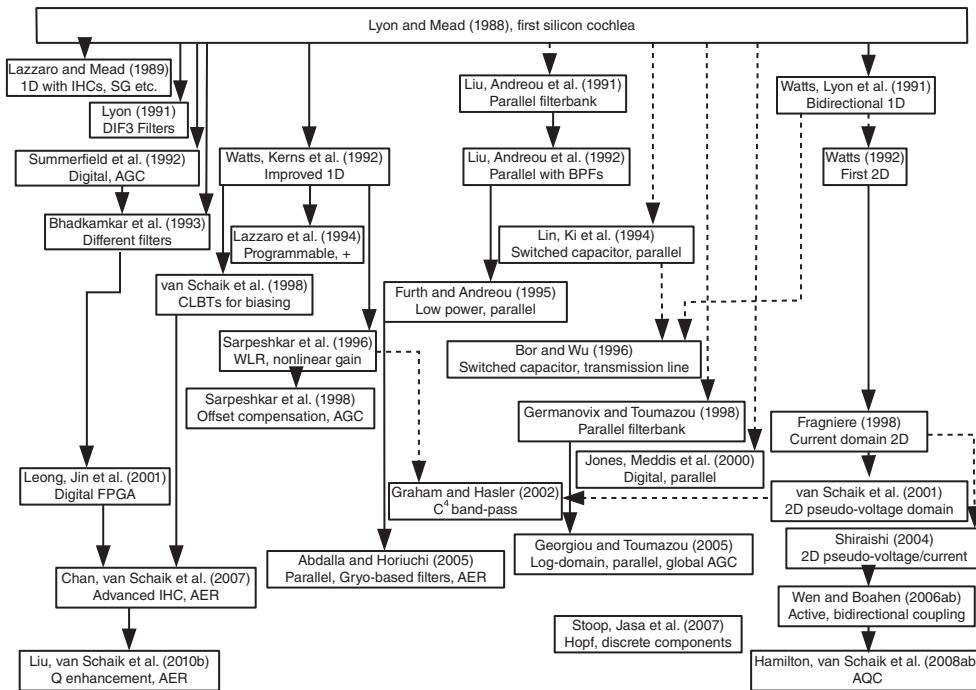


Figure 4.19 Historical tree diagram of silicon cochleas. Solid arrows show models that follow directly from one another; dashed arrows show where elements of previous models were used. Shaded boxes show those models in which the nonlinear properties of the OHCs are modeled

audio front ends in speech recognition systems. The complexity of a model used to design a silicon cochlea will depend on its end application. However, the spirit of silicon cochlea design remains the same no matter what the application, that is, it is to leverage biological functionality in modern technology. In our quest to build better and more realistic silicon cochleas, we are being confronted with many of the challenges that biology has had to overcome. While there has not been a silicon cochlea built to date which rivals the biological cochlea, the situation is improving, with better models and better technology. Over the past 20 years we have seen a move from simple 1D filter cascades to more biologically plausible 2D structures which include OHC functionality. There have also been improvements in circuit design, and as miniaturization of CMOS fabrication technology continues, we can include more circuits (filters, cell structures, etc.) on a single integrated circuit. There are still many open issues in silicon cochlea design today: the use of analog over digital or vice versa, the operating domain (current or voltage), and the choice of biological model. Several of these issues are engineering issues which, with time and experimentation, will be resolved. The choice of biological model is dependent on our understanding of the biological cochlea which continues to change as our observational capabilities improve. In the two decades since Lyon and Mead's silicon cochlea, we are still faced with many of the same design challenges: noise, dynamic range, and mismatch to name but a few. We have shown in this chapter, however, that our silicon cochlea designs have improved significantly in 20 years and by overcoming the

design challenges that still face us, we are improving our understanding of how the biological cochlea evolved into what it is today.

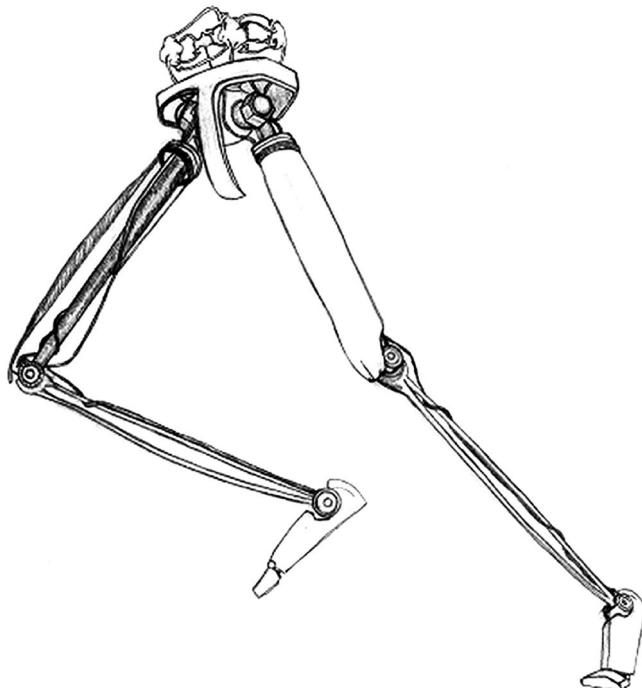
References

- Abdalla H and Horiuchi T. 2005. An ultrasonic filterbank with spiking neurons. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **5**, pp. 4201–4204.
- Abdollahi M and Liu SC. 2011. Speaker-independent isolated digit recognition using an AER silicon cochlea. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 269–272.
- Bhadkamkar N and Fowler B. 1993. A sound localization system based on biological analogy. *Proc. 1993 IEEE Int. Conf. Neural Netw.* **3**, pp. 1902–1907.
- Bor JC and Wu CY. 1996. Analog electronic cochlea design using multiplexing switched-capacitor circuits. *IEEE Trans. Neural Netw.* **7**(1), 155–166.
- Chan V, Liu SC, and van Schaik A. 2007. AER EAR: A matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuits Syst. I: Special Issue on Smart Sensors* **54**(1), 48–59.
- Chan V, Jin CT, and van Schaik A. 2012. Neuromorphic audio-visual sensor fusion on a sound-localizing robot. *Front. Neurosci.* **6**, 1–9.
- Finger H and Liu SC. 2011. Estimating the location of a sound source with a spike-timing localization algorithm. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2461–2464.
- Fragnière E. 1998. *Analogue VLSI Emulation of the Cochlea*. PhD thesis. Ecole Polytechnique Federale de Lausanne Lausanne, Switzerland.
- Fragnière E. 2005. A 100-channel analog CMOS auditory filter bank for speech recognition. *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers* **1**, pp. 140–589.
- Forth PM and Andreou AG. 1995. A design framework for low power analog filter banks. *IEEE Trans. Circuits Syst. I: Fundamental Theory and Applications* **42**(11), 966–971.
- Georgiou J and Tounazou C. 2005. A 126 μ W cochlear chip for a totally implantable system. *IEEE J. Solid-State Circuits* **40**(2), 430–443.
- Germanovix W and Tounazou C. 1998. Towards a fully implantable analogue cochlear prosthesis. *IEE Colloquium on Analog Signal Processing (Ref. No. 1998/472)*, pp. 10/1–1011.
- Graham DW and Hasler P. 2002. Capacitively-coupled current conveyer second-order section for continuous-time bandpass filtering and cochlea modeling. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **5**, pp. 482–485.
- Hamilton TJ, Jin C, van Schaik A, and Tapson J. 2008a. An active 2-D silicon cochlea. *IEEE Trans. Biomed. Circuits Syst.* **2**(1), 30–43.
- Hamilton TJ, Tapson J, Jin C, and van Schaik A. 2008b. Analogue VLSI implementations of two dimensional, nonlinear, active cochlea models. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 153–156.
- Hudspeth AJ and Corey DP. 1977. Sensitivity, polarity, and conductance change in the response of vertebrate hair cells to controlled mechanical stimuli. *Proc. Nat. Acad. Sci. USA* **74**(6), 2407–2411.
- jAER. 2007. jAER Open Source Project, <http://jaerproject.org> (accessed July 28, 2014).
- Jones S, Meddis R, Lim SC, and Temple AR. 2000. Toward a digital neuromorphic pitch extraction system. *IEEE Trans. Neural Netw.* **11**(4), 978–987.
- Katsiamis A, Drakakis E, and Lyon R. 2009. A biomimetic, 4.5 μ W, 120 + dB, log-domain cochlea channel with AGC. *IEEE J. Solid-State Circuits* **44**(3), 1006–1022.
- Lazzaro J and Mead C. 1989. Circuit models of sensory transduction in the cochlea. In: *Analog VLSI Implementations of Neural Networks* (eds. Mead C and Ismail M). Kluwer Academic Publishers. pp. 85–101.
- Lazzaro J, Wawrynek J, and Kramer A. 1994. Systems technologies for silicon auditory models. *IEEE Micro.* **14**(3), 7–15.
- Leong MP, Jin CT, and Leong PHW. 2001. Parameterized module generator for an FPGA-based electronic cochlea. *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '01*. IEEE. pp. 21–30.
- Li CH, Delbrück T, and Liu SC. 2012. Real-time speaker identification using the AEREAR2 event-based silicon cochlea. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1159–1162.
- Lin J, Ki WH, Edwards T, and Shamma S. 1994. Analog VLSI implementations of auditory wavelet transforms using switched-capacitor circuits. *IEEE Trans. Circuits Syst. I: Fundamental Theory and Applications* **41**(9), 572–583.

- Liu SC, Mesgarani N, Harris J, and Hermansky H. 2010a. The use of spike-based representations for hardware audition systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 505–508.
- Liu SC, van Schaik A, Minch B, and Delbrück T. 2010b. Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2027–2030.
- Liu SC, van Schaik A, Minch B, and Delbrück T. 2014. Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output. *IEEE Trans. Biomed. Circuits Syst.* **8**(4), 453–464.
- Liu W, Andreou AG, and Goldstein, Jr. MH. 1991. An analog integrated speech front-end based on the auditory periphery. *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)* **II**, pp. 861–864.
- Liu W, Andreou AG, and Goldstein MH. 1992. Voiced-speech representation by an analog silicon model of the auditory periphery. *IEEE Trans. Neural Netw.* **3**(3), 477–487.
- Lyon RF. 1990. Automatic gain control in cochlear mechanics. In: *The Mechanics and Biophysics of Hearing* (eds Dallos P, Geisler CD, Matthews JW, Ruggero MA, and Steele CR). Lecture Notes in Biomathematics, vol. 87. Springer, New York. pp. 395–420.
- Lyon RF. 1991. Analog implementations of auditory models. *Human Language Technology Conference Proceedings of the Workshop on Speech and Natural Language*. Association of Computational Linguistics, Stroudsburg, PA. pp. 212–216.
- Lyon RF and Mead CA. 1988. An analog electronic cochlea. *IEEE Trans. Acoust. Speech Signal Process.* **36**(7), 1119–1134.
- Palmer AR and Russell IJ. 1986. Phase-locking in the cochlear nerve of the guinea-pig and its relation to the receptor potential of inner hair cell. *Hear. Res.* **24**(1), 1–15.
- Plack CJ. 2005. *The Sense of Hearing*. Lawrence Erlbaum Associates, New Jersey.
- Robles L, Ruggero MA, and Rich NC. 1997. Two-tone distortion on the basilar membrane of the chinchilla cochlea. *J. Neurophys.* **77**(5), 2385–2399.
- Ruggero MA. 1992. Responses to sound of the basilar membrane of the mammalian cochlea. *Curr. Opin. Neurobiol.* **2**(4), 449–456.
- Ruggero MA, Robles L, and Rich NC. 1992. Two-tone suppression in the basilar membrane of the cochlea: mechanical basis of auditory-nerve rate suppression. *J. Neurophys.* **68**(4), 1087–1099.
- Sarpeshkar R, Lyon RF, and Mead CA. 1996. An analog VLSI cochlea with new transconductance amplifiers and nonlinear gain control. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 292–296.
- Sarpeshkar R, Lyon RF, and Mead CA. 1998. A low-power wide-dynamic-range analog VLSI cochlea. *Analog Integr. Circuits Signal Process.* **16**, 245–274.
- Sarpeshkar R, Salthouse C, Sit JJ, Baker MW, Zhak SM, Lu TKT, Turicchia L, and Balster S. 2005. An ultra-low-power programmable analog bionic ear processor. *IEEE Trans. Biomed. Eng.* **52**(4), 711–727.
- Shirashi H. 2004. *Design of an Analog VLSI Cochlea*. PhD thesis. School of Electrical and Information Engineering, University of Sydney, Australia.
- Stoop R, Jasa T, Uwate Y, and Martignoli S. 2007. From hearing to listening: design and properties of an actively tunable electronic hearing sensor. *Sensors* **7**(12), 3287–3298.
- Summerfield CD and Lyon RF. 1992. ASIC implementation of the Lyon cochlea model. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)* **5**, pp. 673–676.
- van Schaik A. 1998. *Analogue VLSI Building Blocks for an Electronic Auditory Pathway*. PhD thesis. Ecole Polytechnique Federale de Lausanne Lausanne, Switzerland.
- van Schaik A and Fragnière E. 2001. Pseudo-voltage domain implementation of a 2-dimensional silicon cochlea. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 185–188.
- Watts L. 1992. *Cochlear Mechanics: Analysis and Analog VLSI*. PhD thesis. Computation in Neural Systems Dept., California Institute of Technology, Pasadena, CA.
- Watts L, Lyon R, and Mead C. 1991. A bidirectional analog VLSI cochlear model. In: *Advanced Research in VLSI: Proceedings of the 1991 University of California/Santa Cruz Conference* (ed. Squin CH). MIT Press, Cambridge, MA. pp. 153–163.
- Watts L, Kerns D, Lyon R, and Mead C. 1992. Improved implementation of the silicon cochlea. *IEEE J. Solid-State Circuits* **27**(5), 692–700.
- Wen B and Boahen K. 2006a. A 360-channel speech preprocessor that emulates the cochlear amplifier. *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 556–557.
- Wen B and Boahen K. 2006b. Active bidirectional coupling in a cochlear chip. In: *Advances in Neural Information Processing Systems 18 (NIPS)* (eds. Weiss Y, Schölkopf B, and Platt J). MIT Press, Cambridge, MA. pp. 1497–1504.
- Yu T, Schwartz A, Harris J, Slaney M, and Liu SC. 2009. Periodicity detection and localization using spike timing from the AER EAR. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 109–112.

5

Locomotion Motor Control



This chapter turns from the sensors discussed in Chapters 3 and 4 to the motor output side of neuromorphic systems. Understanding how to engineer biological motor systems has far reaching implications, particularly in the fields of medicine and robotics. However, there are many complexities when it comes to understanding biological design. This chapter describes ways in which scientists and engineers have uncovered truths about the control circuitry behind animal locomotor skills (the motor circuits responsible for an

organism's locomotion) and discusses practical design solutions used for implementing these circuits. Neuromorphic design of locomotor systems in particular aims to replicate the efficiency and efficacy of biological design for its remarkable performance under real-world conditions. The end result of such efforts helps spawn innovative solutions for individuals with impaired motor function, and in the field of robotics, neuromorphic design will continue to redefine the level of robotic interactions within nature, as well as situations and environments adapted to human ergonomics. (Sections of this chapter have been adapted from the thesis work of Vogelstein (2007).)

5.1 Introduction

Evolution has produced a variety of animals that have developed interesting ways of moving in their environment. As scientists and engineers, we sometimes look toward nature for inspiration, tapping into millennia of evolutionary history to help solve problems relevant in the world of today and tomorrow. Nature, not being an engineer, but rather a master of ‘getting the job done,’ often bases its designs on longevity. That is to say, what qualities did a particular organism acquire to enable it to stand the test of time and fulfill the Darwin ideology, ‘survival of the fittest’? Although we may borrow ideas for the engineering of systems and devices from biology, we must also keep in mind that biology is motivated by much more than just a narrow functional purpose. A structure such as the leg of an insect may have obvious form and function responsibilities in helping the animal move, however, as in the case of the flower mantis, legs and other appendages are also designed to look like their environment for camouflage and are not related to form or function with regard to locomotion. This idea introduces the concept of functional biomimesis, a term used by Klavins et al. (2002) to denote the use of only the functional elements of biological design that help engineers solve their problems.

5.1.1 Determining Functional Biological Elements

Continuing with the theme of borrowing from biology, we look at vertebrate motor systems primarily responsible for locomotion in an attempt to learn what structural entities and parameters are important in giving animals such robust control while navigating through their environment. There are two classical regimes involved with this subject, a kinematic approach and a neural approach. In a kinematic study of the way biological systems move, one focuses heavily on the mechanical aspects by analyzing how animal posture, shape, position, and body angle relative to the direction of travel enable locomotion (Quinn et al. 2001). In contrast, study of neural elements engaged in locomotion reveals important concepts that are relevant regardless of the species under study. For example, the notion of a central pattern generator (CPG) being responsible for controlling locomotor functions of an animal can be isolated and decomposed into individual neural elements (Grillner and Wallén 1985; Marder and Calabrese 1996); in general, CPGs are responsible for almost all rhythmic motor functions. Our emphasis in this chapter is on understanding the neural elements like CPGs that guide locomotion and other rhythmic motor activity. Through case studies, we also observe how engineers identify functional neural control elements to guide engineering design, and we shall continue with the general discussion of neuromorphic design as it applies to motor control circuits. As our applications delve deeper into the cellular level for deriving biological inspiration,

constraints in engineering processes like electronic device fabrication technologies limit the way we model and implement nature's design. So, as in functional biomimesis, neuromorphic design practices do not seek to replicate every anatomical detail of biology, but rather use the principles that make the system work.

5.1.2 Rhythmic Motor Patterns

At a conceptual level, the vertebrate propulsive locomotor system can be divided into four components, one each for selection, initiation, maintenance, and execution of the motor program (Figure 5.1) (Grillner 2003; Grillner et al. 2000). Selecting a motor program involves multimodal sensory integration and is therefore performed in the forebrain. Once a program has been selected, this information is communicated to the brainstem. The mesencephalic locomotor region (MLR) in the brainstem is responsible for initiating and maintaining locomotion, but it does not directly produce the locomotor rhythm (Shik and Orlovsky 1976). Instead, it relays information to specialized neural circuits within the spinal cord (collectively called CPG for locomotion) that are responsible for generating rhythmic motor output (Grillner 2003; Grillner et al. 2000). These circuits are distributed over multiple spinal segments and they control movements of the legs, arms, and torso, all of which are coordinated during normal locomotion. In this chapter we are exclusively concerned with the elements of the CPG that control the legs and are located in the lumbar spinal cord. Therefore, when

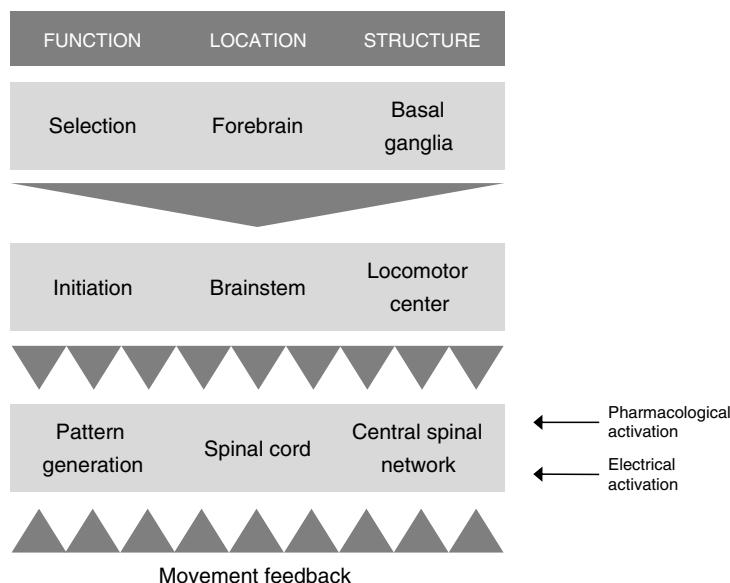


Figure 5.1 System diagram of the vertebrate locomotor system (Grillner 2009; Grillner et al. 2000). The brainstem locomotor center includes regions such as the diencephalic and mesopontine locomotor areas; communication from the brainstem to the spinal cord is largely routed through reticulospinal neurons

we use the terms ‘CPG’ and ‘locomotor circuits,’ we are actually referring to one specific component of this extensive network.

The effect of descending supraspinal inputs on the CPG is to establish a set of network parameters, or operating conditions, for these neural circuits (Deliagina et al. 2002; Matsuyama et al. 2004; Zelenin 2005; Zelenin et al. 2001). Although the complete list of supraspinal targets within the CPG varies between species (and is not yet fully characterized), descending inputs are known to both inhibit and excite specific sub-populations of spinal neurons that affect the motor rhythm (Li et al. 2003; Matsuyama et al. 2004; Wannier et al. 1995). Efferent signals from the brain can also modulate the effects of afferent inputs to the CPG (Hultborn 2001). The net result of all of these efferent and afferent inputs on the CPG is that it produces a particular sequence of motor outputs that implements the desired motor program (Cohen et al. 1988).

CPGs have long been established as the origin of rhythmic motor activity in primitive and quadrupedal vertebrates (Cohen and Wallén 1980; Graham-Brown 1911; Grillner and Zanger 1979), but convincing evidence of their existence in primates has been presented only recently (Fedirchuk et al. 1998). To convincingly show that the CPG is capable of generating locomotion by itself, the spinal cord must be deprived of phasic descending and sensory inputs. This is required because, under normal conditions, the CPG’s output is affected by both efferent and afferent signals, and it is constantly integrating information from the brain with information from the limbs and torso (Cohen et al. 1988; Van de Crommert et al. 1998). However, if these inputs were removed and rhythmic motor output were still produced, it would be clear that the CPG was the source of this activity. This conceptual experiment can be physically demonstrated by decerebrating or spinalizing an animal and using a neuromuscular blockade or deafferentiation to remove rhythmic sensory feedback. Tonic stimulation of the MLR or injection of a suitable pharmaceutical agent can then induce ‘fictive locomotion’, that is, rhythmic output from the motor neurons in the absence of physical movement (reviewed in Orlovsky et al. 1999).

A basic structure for the CPG network was first proposed by T. G. Brown, more than a century ago, in the form of a half-center oscillator (HCO) (Graham-Brown 1911). The HCO (Figure 5.2) represents two neural ‘centers’ that alternate in their activities – the centers can represent flexors and extensors in one limb, or the left and right limbs, or any other opposing motor pair. This central concept remains at the core of modern CPG models, which have filled in many of the details of the oscillator substructure for specific vertebrate systems (Grillner et al. 1998; Ijspeert 2001; Rybak et al. 2006a, 2006b; Sigvardt and Miller 1998). In some animals, specific populations of cells that compose the CPG have been identified, and in a few cases, particular ion channels have been identified as the source of oscillations (Dale 1995; Gosgnach et al. 2006; Grillner 2003; Grillner et al. 2000; Huss et al. 2007; Kiehn 2006; Nistri et al. 2006). However, just as development of the cochlear implant preceded a complete

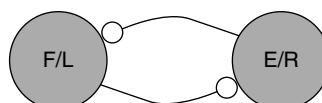


Figure 5.2 Half-center oscillator model of the CPG (Graham-Brown 1911). The two neural ‘centers’ can represent either flexors (F) and extensors (E) in one limb, or the left (L) and right (R) limbs, or any other opposing motor pair

understanding of the auditory system, extensive knowledge about the cellular mechanisms of the CPG is not a prerequisite to controlling the CPG.

5.2 Modeling Neural Circuits in Locomotor Control

Our goal is to understand how a CPG, or a network of CPGs, can enable an animal to walk, swim, or crawl, and how this knowledge can be used in the design of neuromorphic controllers for robots. Therefore, we will ignore some of the stabilizing mechanics that are controlled by various neurons that modulate CPG functionality. With this simplification we can focus on the three important components in CPG design: (1) the neural signals that CPGs receive as inputs, (2) the outputs of CPGs to control essential muscle activations that drive the animal's locomotion, and (3) the connectivity of neural and motor elements that enable CPGs to function. Biological organisms with less complex CPG circuits such as lamprey, leech, and tadpoles have offered scientists the opportunity to predict the behavior of CPGs and subsequent patterning of muscle activation by diagramming the neural circuit involved (Grillner et al. 1991; Sigvardt and Williams 1996), but, as evident in Grillner et al. (1991), the level of detail is very coarse. Yet, even in lamprey and simply structured organisms like invertebrates where some diagramming has been completed, CPG output behavior can easily be altered by a number of normal biological processes such as changes in membrane potential, synaptic conductance, neurotransmitter concentration, and sensory afferents (Harris-Warrick 1990; Marder and Calabrese 1996; Selverston 2010; Sponberg and Full 2008). Because of this, CPGs that are realized from careful connectionist diagrams of the organism should be coupled with knowledge of how the organism moves *in vivo* or during fictive motion.

It has been well established that a dedicated CPG or network of CPGs enables locomotor function in many animal species. Yet, there still remain many unanswered questions surrounding the interconnections of neurons involved in locomotion and how neural modulation, timing, sensory afferents, and firing patterns modify the behavior of CPGs. This can alter CPG output for specific actions in ways that are hard to predict with current CPG models which often can only emulate one or two types of motor behavior and lack the ‘richness’ observed in the biological counterpart (Ijspeert 2008). To this end, scientists and engineers have developed methods for investigating how CPGs are most likely constructed to enable the animal under study to move and perform actions; analytical techniques for exploring hypotheses, model generation, and stability; and implementations that allow investigators to test theories and designs under real-world conditions (see Figure 5.3).

Mathematical analysis and simulations have been used to study CPG dynamics in lamprey (Cohen et al. 1982) and connectionist models help identify how neural circuitry (at the cellular or oscillator level) can reproduce observed motor behavior (Ijspeert 2001). 2D and 3D models can provide context for CPG models by simulating body mechanics, force, and friction interactions (Ijspeert 2001). Statistical analyses of measured effects of CPG control like gait cycle frequency, phase lag, and angles between joints or segments, or ground reaction forces, give insight into what control strategies CPGs use to maintain stability under various circumstances (Lay et al. 2006, 2007).

CPG models have been implemented in software, which encompasses the majority of the published implementations (see Ijspeert (2008) and Selverston (2010) for reviews), in neuromorphic silicon design (DeWeerth et al. 1997; Kier et al. 2006; Lewis et al. 2000;

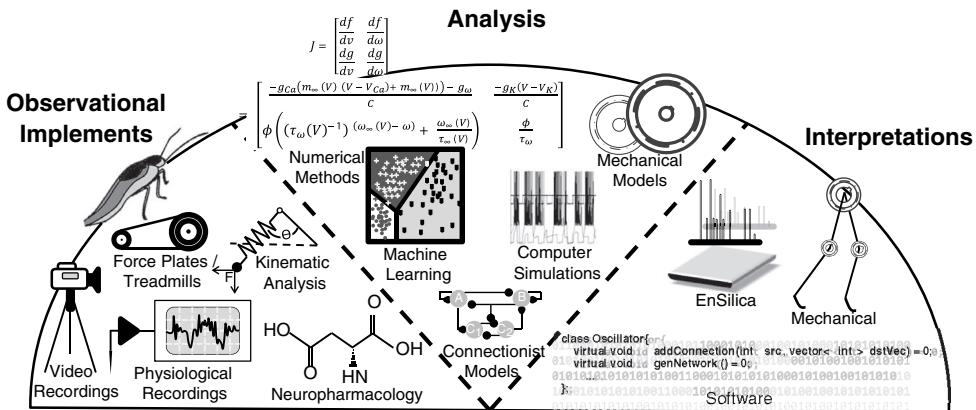


Figure 5.3 Illustration of the locomotor neuromorphic end-to-end design process. From left to right, common observational tools, analysis techniques, and results are shown in graphical form. Several combinations from each major step in the design process are commonly used

Nakada et al. 2005; Still et al. 2006; Tenore et al. 2004b; Vogelstein et al. 2006, 2008), and in mechanical systems whose properties match dynamical systems controlled by biological CPGs (Collins et al. 2005).

5.2.1 Describing Locomotor Behavior

If multiple locomotor behaviors are to be defined, then it may also be necessary to describe the way in which the animal naturally transitions between different types of movement.

A locomotor function can be decomposed into a set of unique phases. Each phase describes the relevant beginning and end positions of the body and or limbs at a particular instance in time relative to some initial state. Through ordered repetition of each phase, a cycle of normal locomotor behavior can be defined. Properly defining each phase is the key to a deeper understanding of the sequence of motor activity that defines a locomotor function. If multiple locomotor behaviors are to be defined, then it may also be necessary to describe the way in which the animal naturally transitions between types of movement.

Our first example is that of human bipedal locomotion. Figure 5.4 depicts the positional changes that occur within each leg. Imagine each phase to be stills that were captured from a video recording of a person in motion. Here, seven relevant and unique positions comprise the gait cycle. Within this gait cycle there are two main phases to take note of: (1) the stance phase and (2) the swing phase. Each phase is further subdivided into periods that uniquely describe the transition between successive events. The events themselves are indicative of the most salient attribute of the gait.

Accurate and detailed descriptions of the events can then lead to hypotheses about what muscle groups must be active to maintain stability at each event and to sustain movement through each period. However, other techniques are often used for more thorough analysis. Some of these techniques include static and dynamic mathematical models of the physical system, the use of force plates, and electromyography (EMG) data. Many of these techniques

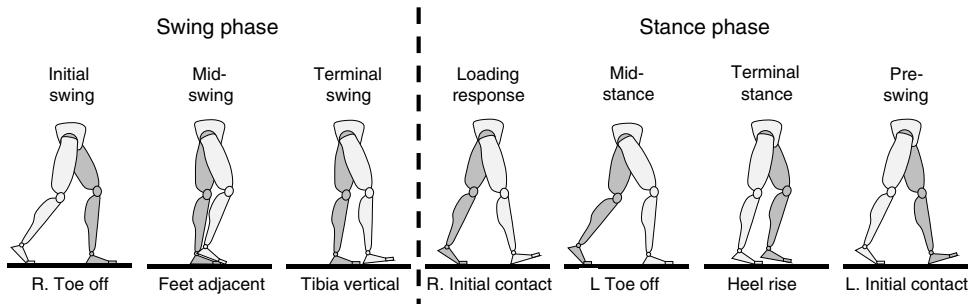


Figure 5.4 Gait analysis diagram

can be used in conjunction in order to paint the best picture of muscle activity during movement. Some of the important parameters that are found through gait analysis and that lead to the eventual creation of functional CPG circuits are listed here.

- *Muscle activity patterns:* Enables the placement of neurons within a CPG to excite or inhibit activity of specific motor neurons or actuators within a robotic system.
- *Joint angle/joint and body contact forces/joint power:* Can be used as sensory feedback to CPG neurons to excite or inhibit activation of muscles at a particular instant within an event or onset of a period between events.
- *Angular phase and timing of gait cycle events and periods for each gait phase:* This is often a good measure of normal performance. Matching gait phase and timing parameters will help one tune and adjust properties of neurons and synaptic weighting for excitation or inhibition within a CPG circuit.

5.2.2 Fictive Analysis

Some investigative quandaries involving the underlying circuitry of CPGs are not easily answered through external observations such as those described in the previous section. Study of lamprey once presented such a problem to investigators first trying to uncover what motor programming could maintain a constant phase angle between the body segments of the undulating animal, invariant to the speed of undulation (Marder and Calabrese 1996). Lamprey move through the water by creating a single traveling wave as depicted in Figure 5.5. Maintaining a single wave as the animal swims is important for energy efficiency as well as locomotor stability. Failure to maintain a single wave at any frequency and regardless of direction (forward or backward swimming) would be considered abnormal for the animal's locomotor behavior (Marder and Calabrese 1996).

Marder and Calabrese (1996) detail how analysis of isolated lamprey nerve cords helped investigators answer important questions regarding lamprey CPGs. The following questions are summarized from their report. Is lamprey swimming the result of a coordinated effort of individual segmental oscillators or is the pattern generated by a CPG network distributed over the segments? If there are coordinated segmental oscillators, how do they communicate? How does sensory information modify locomotor behavior? Last, how is the pattern initiated and maintained?

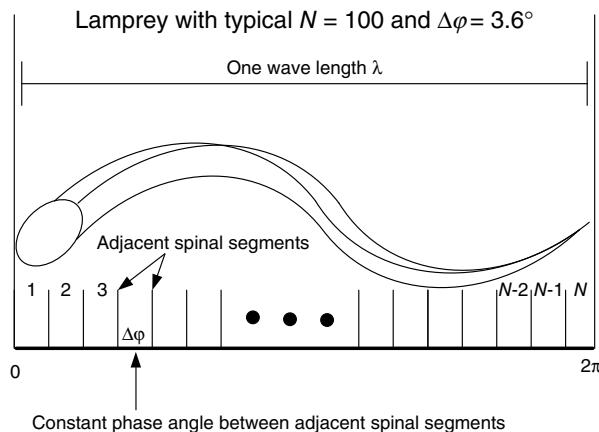


Figure 5.5 Depiction of a lamprey swimming showing one complete body wave. There are N segments to this lamprey where N is typically 100. The phase lag in degrees between each adjacent body segment in a lamprey can be readily computed from N using the formula $\Delta\phi = \frac{2\pi}{N} \cdot \frac{180}{\pi}$

The use of fictive motion generated by the various nerve cord preparations was crucial in uncovering the answers to the above questions. Experiments with fictive swimming can help one understand the contributions of different aspects of CPG circuits through the study of behavior in isolation. For example, analysis of small spinal segments showed that normal motor function could still be achieved, indicating that each segment was capable of producing its own stable and sustainable oscillatory patterns. Further investigation with fictive swimming also revealed that two types of stretch receptors are responsible for communicating important state information to various segments of the animal, helping to maintain body waves over a wide range of frequencies (Marder and Calabrese 1996). Stretch receptors are also thought to be responsible for the difference in constant phase lag observed in intact animal swimming (5° per segment) and that of EMG muscle activation and fictive swimming (3.6° per segment) (Hill et al. 2003; Marder and Calabrese 1996). Studies involving fictive motion have shown that decerebrated preparations, as well as deafferentiated preparations, are able to display a wide range of motor skills (Grillner and Wallén 1985). However, they fail to produce smooth natural rhythms and are more susceptible to the loss of stability due to perturbations. Although these preparations have traditionally been used to provide evidence that CPGs control motor functions locally, they also suffice to show that coordinated efforts from cerebellar and sensory pathways are necessary for robust motor activity.

Identifying the pathways of intersegmental coordination has proved challenging. For example, questions still remain about the degree of symmetry in the coupling of segment oscillators and if segments operating at different oscillation frequencies are responsible for maintaining the phase lag among the segments. As well studied as lamprey models are, its locomotor system still has a number of complexities that go beyond the basic rhythmic motor patterns we aim to understand. Fortunately, a full understanding of all these issues is not needed in order to realize a neuromorphic design that is capable of replicating the basic motions of the creature, as evident in DeWeerth et al. (1997).

5.2.3 Connection Models

Connectionist models of neural networks serve many purposes in the scientific community. With respect to CPGs, there are investigations that seek to understand the anatomical connectivity of the networks and some that seek to understand the function of that connectivity (Getting 1989; Grillner and Wallén 1985). Understanding the functional connectivity of a neural circuit is paramount when building neuromorphic controllers for locomotion. Capturing the details about the anatomical structure is important but that alone may not lead to the desired control output. As previously mentioned, modulation of neural elements in a circuit can drastically change its behavior and the synaptic weighting and cellular properties are not reflected in the study of anatomical connections alone. Rather, the system must be probed at key points to capture the subset of neural elements responsible for a particular action, the types of connection (whether it is inhibitory or excitatory), cellular dynamics such as regular spiking or bursting, and some synaptic properties like strength. Table 1 of Getting (1989) provides a more complete list of neural network properties that are important to its function. All of these properties may not be applicable to the design of neuromorphic systems (see Figure 5.6). However, such details can provide further insight into the mechanisms by which locomotor control is achieved in the organism. Once the more complex mechanisms are better understood, it may improve the way engineers approach neuromorphic solutions in the future.

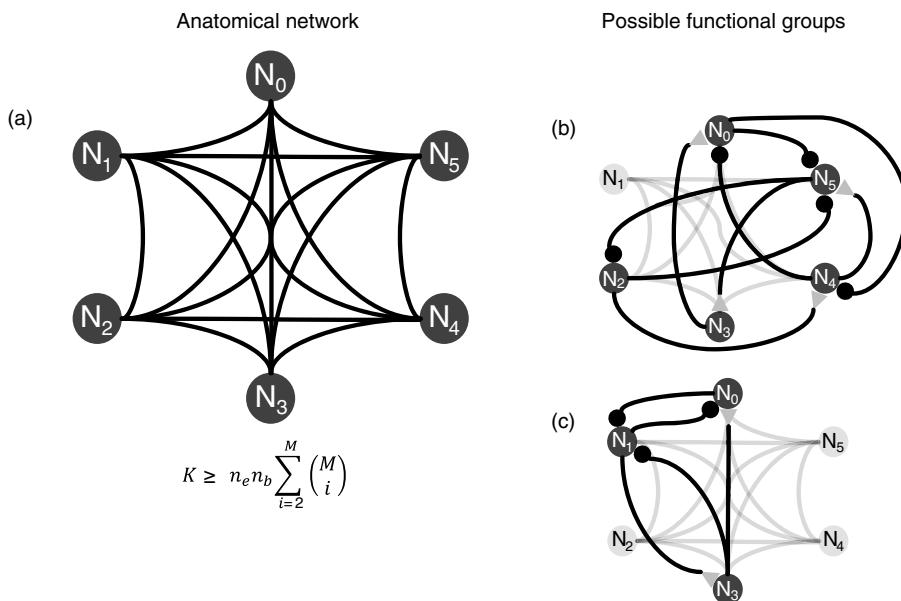


Figure 5.6 Simplifications into relevant functional blocks can help engineers design models that are easier to describe analytically and implement, especially when it comes to hardware design. The anatomical network (a) can have two possible functional blocks (b) and (c). The number of network configurations, K , is determined by n_e , the number of ways a connection between neurons can be expressed, n_b , the types of synaptic connection, and the number of neurons, M . In this example, $n_e = 3$ because of the 3 types of connections (out, in, or reciprocal), and $n_b = 2$ because of the inhibitory or excitatory synapses

The work of Ijspeert (2001) is a good demonstration of the use of connectionist models. Although a complete description of the salamander locomotor circuitry is not available, a controller model was developed which incorporated relevant cellular dynamics (via leaky-integrator neurons), connection types, and synaptic weights. In this study, parameters for the network were found using a genetic algorithm rather than data from actual salamanders. Nevertheless, the CPG controller developed was able to simulate swimming and trotting gait patterns of salamanders. In many cases, connectionist models are derived from hypotheses but they have the ability to produce biologically feasible models of CPGs and reproduce gait kinematics in simulators and mechanical models (Ijspeert 2001).

5.2.4 Basic CPG Construction

In the following sections we will discuss some of the integral components of biological CPGs that make them stable, flexible, and efficient controllers of motor activity. We will also discuss how these components are translated into neuromorphic design.

The Unit Oscillator

Aside from the neuron itself, the HCO is the most basic component of many biological CPG networks (Getting 1989). To demonstrate how powerful a simple oscillating structure can be in practice, we examine the work of Still et al. (2006) who use a fixed CPG architecture that is capable of producing the seven types of four-legged animal gait patterns. Each oscillating structure is then entrained via bias voltages that modify the output characteristics of the oscillator duty cycle, frequency of oscillation, and phase lag between other oscillators that are part of the CPG network.

Oscillator duty cycle, frequency, and phase lag have the ability to directly affect locomotor behavior, especially in the simplified examples that we have constructed. The four-legged walking robot of Still et al. (2006) has only one actuating joint on each leg, the hip, which connects it to the trunk of the robot. In order to mimic stable gait patterns observed in nature, each oscillator controlling a hip joint has to be active for a certain duration (the duty cycle), operate at a given frequency to support the velocity of translation, and be active at the appropriate intervals with respect to other actuating hip joints (the phase lag). Bias voltages are used to set the oscillator duty cycle and frequency, while the coupling circuit bias voltages help instantiate the phase lag between the coupled oscillators. To achieve the desired output behavior, the oscillating units must be properly coupled with the appropriate phase lag between units (Figure 5.7). This can be solved analytically or through machine learning. Their design was verified using two (commonly used) architectures: the chain model and the ring model (Figure 5.7).

Phase-Dependent Response

Phase-dependent response (PDR) characteristics can enable the stability and repeatability of events required to sustain locomotion (Vogelstein 2007). Much like our analysis of bipedal gait, PDR characteristics are observed and measured events that help describe phases or periods within a movement. Obtaining PDR characteristics can be tedious because we will inherently be dealing with nonlinear systems whose output we are measuring as a function of phase. The input parameter is typically an injection of current or some other electrical signal into the CPG

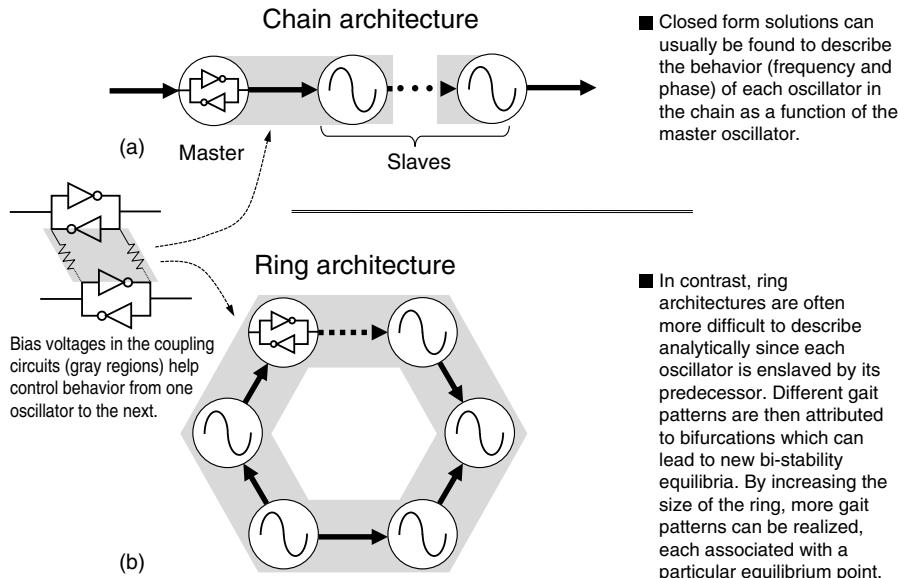


Figure 5.7 Oscillator blocks used by Still et al. (2006) to realize their walking robot. The building blocks on the chip can be used to construct (a) a chain architecture or (b) a ring architecture

circuit under investigation. The stimulus injection process is repeated at appropriately spaced phase intervals in the CPGs cycle in order to gather results. Effective use of PDR characteristics requires that one understands how a certain stimulus will affect motor output when injected at a given phase within the locomotor cycle and that the injected stimulus maintains the stability of the locomotor cycle over time (Vogelstein 2007). The aforementioned requirements both assume that detailed knowledge of baseline behavior has been well documented and can thus serve as a comparison with PDR characteristic data.

PDR characteristics add to the diversity of neural circuits in biological organisms, allowing for simple CPG circuits to exhibit varied outputs in response to descending motor commands and important sensory afferents that control reflex motion. These signals modulate output at the motor end. Under normal circumstances, the programming of a CPG circuit will contain PDR characteristics. For example, a signal to initiate hip flexion to raise the knee in a bipedal gait occurs at a very specific phase for a given leg (90% through a given cycle). Likewise, in the discussion of results by Vogelstein et al. (2006), descending motor commands enable turning in lamprey by injection of stimulation at specific phases that temporarily break the stable swimming pattern followed by subsequent phase-specific stimulations to recover the normal swimming pattern. On the other hand, a sensory signal could induce a PDR via reflex, in which case, the signal to activate the oscillators controlling hip flexion could be under the control of a stretch receptor. If a motor output is tied to a sensory signal in this fashion, the controlling oscillating unit is said to be phase locked to the sensory presentation (Lewis et al. 2000). In either case, the desired motor output can be achieved if the hip flexion continues to occur at the appropriate time. For a discussion on PDR the reader is referred to Vogelstein et al. (2006).

The concept of PDR is closely related to that of delay and timing of activation for individual oscillator units, neurons, or synapses within a CPG. Essentially every signal generated within

a CPG at a given time, t_{event} , can be prescribed a phase with respect to the time it takes to complete one locomotor cycle, T_{cycle} . The phase is then

$$\phi = \frac{t_{\text{event}} - t_0}{T_{\text{cycle}}},$$

where t_0 is the starting time from which the event at time t_{event} is measured through one complete cycle of time T_{cycle} . How that signal is used to affect motor behavior during locomotion is based on the system and the goals desired by the designer.

Sensory Feedback

Sensory feedback is necessary for the proper locomotor function in many animal species. It is important in the maintenance of posture and stability, normal activation of muscle groups during locomotor periods and events, as well as in providing the most rewarding response in the face of unexpected perturbations. Lewis et al. (2000) presented one of the first demonstrations of how sensory feedback could be used within neuromorphic design to not only trigger normal CPG events, but also allow for adaptation to novel environmental perturbations in addition to conditions specific to the robotic system.

5.2.5 Neuromorphic Architectures

Modeling the CPG in Silicon

Since many details surrounding the structure of CPGs in higher vertebrates remain unknown, implementers have a wide latitude of freedom when it comes to modeling these circuits. Correspondingly, a number of silicon CPGs can be found in the literature (Arena et al. 2005; Kier et al. 2006; Lewis et al. 2001; Nakada et al. 2005; Simoni et al. 2004; Still et al. 2006; Tenore et al. 2004b), most of which are designed to control robotic locomotion, with varying degrees of biological plausibility.

In order to make the problem more tractable, simple neuron representations are used in some silicon CPG designs. This simplification allows for the behavior of the CPG to be described analytically. For example, Still et al. (2006) created a CPG chip based on coupled oscillator circuits that generate square waves with three parameters (frequency, duty-cycle, and relative phase) controlled by off-chip bias voltages. When a gait is defined as a function of these parameters, the system of equations can be solved to determine the appropriate voltage levels for establishing the desired output pattern. Moreover, because the circuits themselves are relatively simple, many of these oscillators were implemented on the same die. A potential disadvantage of this design is that it may be difficult to translate findings in biological systems to the representation of square waves. Additionally, it has been shown that more complex neural models perform better at CPG-related tasks (Reeve and Hallam 2005).

The use of simple integrate-and-fire (I&F) neural models in place of simple oscillators brings the design closer to biology, but also increases the analytical complexity. Due to their physical simplicity, I&F cells can have compact representations in silicon, allowing for implementation of many neurons on the same silicon die (Tenore et al. 2004b). However, the basic I&F model

forges neural dynamics such as plateau potentials and spike-frequency adaptation (SFA) that have been shown to generate oscillatory outputs in biological CPGs (Marder and Bucher 2001). On the other hand, more sophisticated neural models based on the Hodgkin–Huxley (HH) formalism can generate realistic spike outputs and emulate the dynamics of real neurons (Simoni et al. 2004). Unfortunately, silicon implementations of HH neurons typically occupy a large area and are not well suited to large-scale integrated networks. The transistor cost of these different neuron circuits is described in Chapter 7.

The system described below is a compromise between a simple I&F model and a complicated HH model. The implemented design augments the basic I&F circuit with features such as a refractory period and SFA, although without the corresponding ion channel dynamics. As a result, it was possible to place an array of 24 neurons with full interconnectivity on a 3 mm × 3 mm die. A previous version of the CPG chip (Tenore et al. 2004b) used an even more primitive neural model, but allowed for only 10 cells in the same area. The limiting factor in that design was the digitally controlled synapses that set the connection strengths between cells. To address this issue, a circuit for compact synapses was developed using nonvolatile analog storage on floating-gate (FG) transistors (Tenore et al. 2005). The FG technology is described in Chapter 10. The chip as shown in Figure 5.8 includes 1032 programmable FG synapses.

Unlike many of the CPG chips presented to date (Arena et al. 2005; Kier et al. 2006; Lewis et al. 2001; Nakada et al. 2005; Simoni et al. 2004; Still et al. 2006), this silicon CPG is not designed to implement any specific CPG network. Instead, it has sufficient flexibility to implement a wide variety of CPG architectures for bipedal locomotion. At minimum, this should involve 12 neurons, one each to control the flexors and extensors of the hips, knees, and ankles (although a first-order system can suffice with four neurons if the hip, knee, and ankle muscles are all coactive). More complicated networks, such as one proposed to model the CPG in cats (Gosgnach et al. 2006; Nistri et al. 2006), require additional elements.

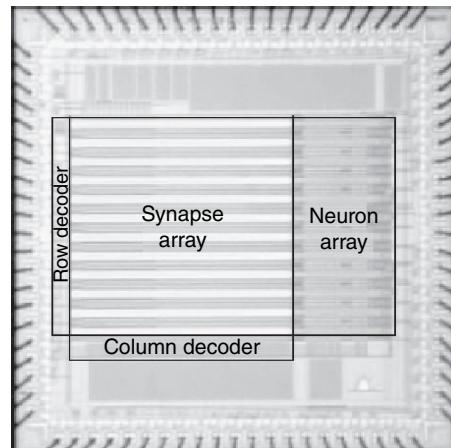


Figure 5.8 Chip microphotograph of the 3 mm × 3 mm FG-CPG chip fabricated in a three-metal two-poly 0.5 μm process

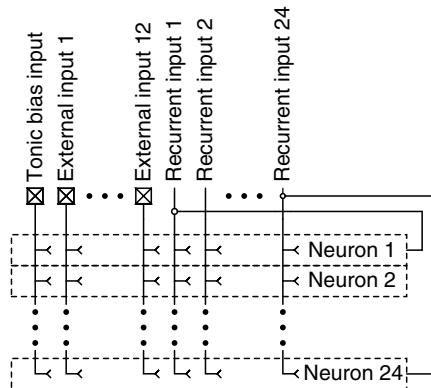


Figure 5.9 The FG-CPG chip floor plan. Twenty-four silicon neurons are arranged in rows, and 37 inputs are arranged in columns. A FG synapse is located at the intersection between each input and neuron (drawn as \rightarrow). The first input is a tonic bias input; the next 12 inputs are ‘external’ in the sense that they are gated by off-chip signals; and the remaining 24 inputs are gated by the outputs of the 24 on-chip neurons and are therefore called ‘recurrent.’ A fully interconnected network can be created by activating all of the recurrent synapses

Chip Architecture

The FG central pattern generator (FG-CPG) chip architecture is similar to that described in Tenore et al. (2004b, 2005) and is illustrated in Figure 5.8. Twenty-four identical silicon neurons are arranged in rows, with 12 external inputs, 1 tonic bias input, and 24 recurrent inputs running in columns across the chip. Each of the 37 inputs makes synaptic connections to all 24 neurons via the FG synapse circuits described below and shown in Figure 5.9.

Neuron Circuit

Also shown in Figure 5.8 is a block diagram overlay of the neuron circuit. Each of the neurons has three different compartments – dendritic, somatic, and axonal – that are both functionally and spatially distinct. From the figure, it can be seen that the dendritic compartments (labeled ‘synapse array’) occupy most of the silicon area, while the somatic and axonal compartments (labeled ‘neuron array’) are relatively smaller. This is partially due to the large number of synapses implemented, but mostly because the membrane capacitance is distributed throughout the dendritic compartment to achieve a compact layout.

The dendritic compartment contains 1 tonic bias, 12 external, and 24 recurrent synapses. All of these inputs deliver current to the somatic compartment, which contains a large capacitor (modeling the membrane capacitance of a biological neuron), a hysteretic comparator (modeling the axon hillock), and some specialized ‘synapses’ that discharge the cell and implement a refractory period. The axonal compartment contains additional specialized synapses that produce variable-duration spike outputs and allow for SFA. The spike outputs are buffered and sent off-chip to control external motor systems and are also used to gate the 24 recurrent synapses between each cell, all of its neighbors, and itself.

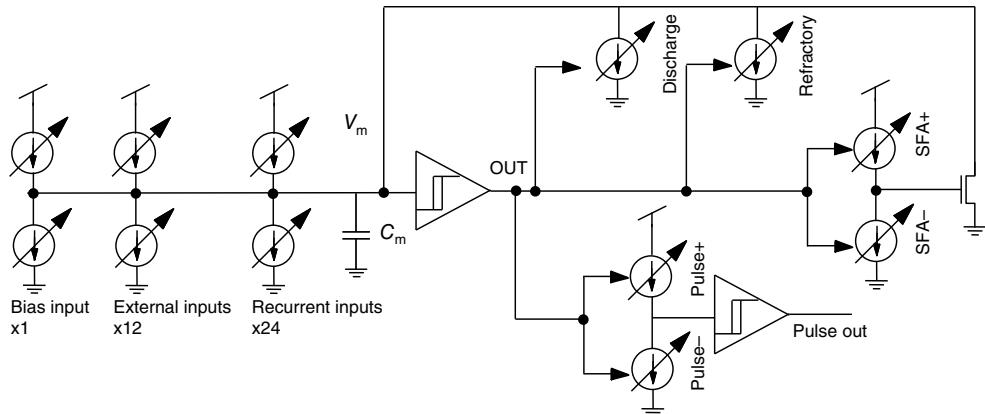


Figure 5.10 Block diagram of the FG-CPG silicon neuron. Including the specialized ‘synapses’ for discharging the cell and implementing a refractory period, pulse width modulation, and SFA, each neuron contains 43 FG synapses (drawn as an individual or a pair of adjustable current sources)

Although all three compartments of the neurons contain the same synapse circuit, only the 37 dendritic inputs are synapses in the traditional sense. Programmable current sources used to reset the neuron and implement a refractory period, spike-width modulation, and SFA are implemented with a FG synapse circuit only for convenience. All four of these special ‘synapse’ circuits are activated after the output of the hysteretic comparator in the somatic compartment goes high (indicating that the neuron’s membrane potential has crossed the spike threshold). Each circuit serves a different function in resetting the cell or adjusting its dynamics. The discharge circuit drains charge from C_m (Figure 5.10) until the membrane potential falls below the hysteretic comparator’s lower threshold, causing the output to go low again. The refractory circuit also discharges C_m and sets the membrane potential to a fixed (tunable) voltage for a fixed (tunable) amount of time. The pulse-width circuit generates a fixed-duration (tunable) pulse whose rising edge coincides with the rising edge of the hysteretic comparator. Finally, the SFA circuit generates a prolonged discharge current proportional to the spike rate, essentially imposing a maximum spike frequency. SFA has been shown to be critical in implementing CPG networks (Lewis et al. 2005).

Silicon neurons with SFA capabilities have been realized in the past (Boahen 1997; Indiveri 2003; Schultz and Jabri 1995; Shin and Koch 1999). Our SFA circuit design does not attempt to capture many details of the biology but instead focuses on the core functionality. The input to the SFA circuit (Figure 5.11a) is composed of two programmable current sources, I_+ and I_- , both gated by the inverse of the output of a neuron’s hysteretic comparator (OUT). The output from the SFA circuit is attached to the neuron’s V_m node. The current I_+ charges up the capacitor C_{sfa} when OUT is high and I_- discharges it when OUT is low. When the voltage on C_{sfa} approaches the threshold voltage of M3, it draws significant current from V_m , making it increasingly difficult for the neuron’s membrane potential to reach threshold. This makes the spikes more sparse until an equilibrium state is reached, in which the spike frequency is stable and V_{sfa} oscillates around a nonzero fixed point (Figure 5.11c).

In normal operation, the voltage on C_{sfa} will not reach the threshold voltage of M3, so M3 will operate in subthreshold and the output frequency will depend exponentially on V_{sfa}

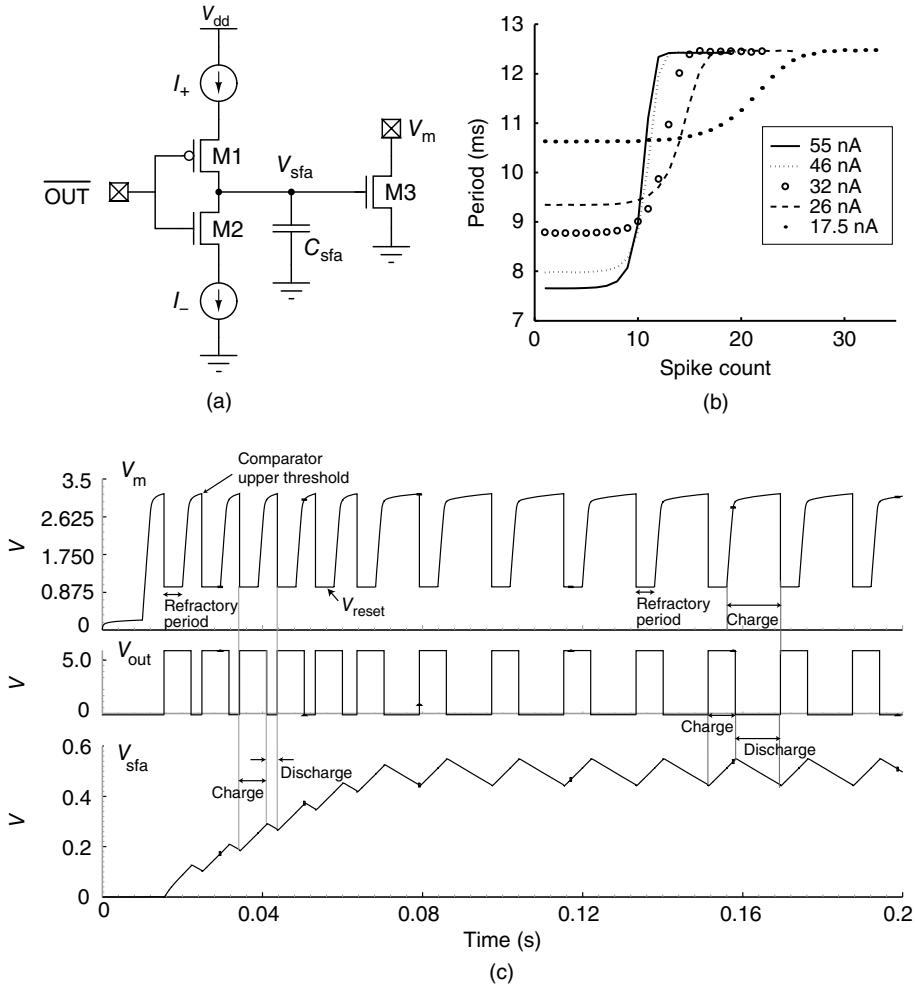


Figure 5.11 (a) Conceptual drawing of SFA circuit. (b) Plot showing an increase in the period of oscillations as the voltage on the SFA capacitor increases (as a function of the number of incoming spikes). (c) Transistor-level simulation of SFA circuit. Top: membrane voltage, center: neuron output, bottom: SFA capacitor voltage

(Figure 5.11b). This relationship can be mathematically described via direct application of Kirchhoff's Current Law at the V_m node of neuron i :

$$C_{m_i} \frac{dV_{m_i}}{dt} = \sum_j W_{ij}^+ I_j - \sum_k W_{ik}^- I_k - S_i I_d - S_i I_{rf} - I_0 e^{\frac{V_{sfa}}{V_t}}$$

$$S_i(t + dt) = \begin{cases} 1, & \text{if } (S_i(t) = 1 \wedge V_i^m > V_T^-) \vee (V_i^m > V_T^+) \\ 0, & \text{if } (S_i(t) = 0 \wedge V_i^m > V_T^+) \vee (V_i^m > V_T^-) \end{cases}$$

where C_{m_i} is the membrane capacitance of the i th neuron; V_T^+ and V_T^- are the high and low thresholds of the hysteretic comparator, respectively; V_{m_i} is the voltage across the membrane capacitor; and $S_i(t)$ is the state of the hysteretic comparator at time t . The first summation is over all the excitatory synapses of the i th neuron and the second summation is over all the inhibitory synapses. The discharge and refractory currents, I_d and I_{rf} , are responsible for the rate of discharge and the duration of the refractory period, respectively. The final element represents the current that is drained from the capacitor due to the SFA circuitry, and specifically to the gate voltage on transistor M3 (Figure 5.11a). This current is the main source of discharge of the subthreshold contributions; all other dependencies are assumed to be negligible and, with some additional considerations, can be incorporated into I_0 .

Synapse Circuit

The 1032 on-chip FG synapses (including external, recurrent, and ‘special’ synapses) are all implemented with a simple nine-transistor operational transconductance amplifier (OTA). The relative voltages on any individual OTA’s differential input determines the strength of that synapse and those voltages are controlled by a pair of FGs. Therefore, the FGs on each synapse need to be programmed before the synapse can be used.

To program a synapse (see Figure 5.12), it must first be selected by setting the row select line (V_{rw}) low and the column-wise V_{prog+} or V_{prog-} line high. A current is then programmed onto the FG, for example, C_{FG1} , by attaching a current source to I_{prog} , causing hot electron injection (HEI) (Diorio 2000) on transistor FG2. This decreases the voltage on the gate of FG1 and allows more current to flow along the transconductance amplifier’s positive branch. The difference of the voltages on the two gates FG1 and FG3 determines the properties of the synapse: the polarity of the synaptic current (excitatory or inhibitory) is determined by the sign of the difference FG1–FG3 while the synaptic strength is set by the magnitude of the difference. Because HEI can only reduce the voltage on the FGs, they may eventually need to be reset to a high potential. This is achieved by Fowler–Nordheim tunneling (Lenzlinger and Snow 1969) through a tunneling junction attached to V_{syn+} and activated by the global V_{erase} pin (not shown). However, because tunneling requires high voltages, it is avoided as often as possible – the differential pair design allows for multiple increases and decreases in synaptic strength exclusively through HEI.

Recurrent synapses are activated when the output of the appropriate neuron in the array pulses the *trigger* and *trigger* lines, which gate the two currents generated by the differential pair (and mirrored by M3, M4, M5, M13 and M7–M8) onto the cell’s membrane capacitor C_m . The diode-connected transistors M9–M12 implement a voltage-dependent conductance that decreases the synaptic current as the potential stored on the membrane capacitor approaches the voltage rails (Tenore et al. 2004b). This allows a wider range of coupling strengths between neurons (Tenore et al. 2004a).

External Input Circuit

The impact of any given external input upon each of the 24 neurons is a function of both the strength of the synapses (controlled by the FGs, as described above) and the magnitude of

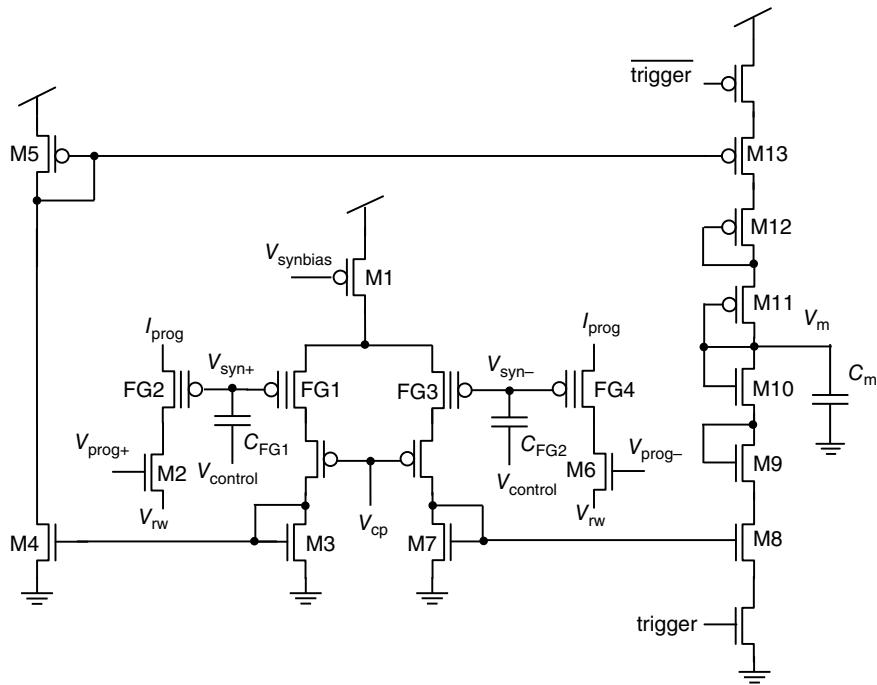


Figure 5.12 FG-CPG synapse circuit schematic. Each synapse contains nonvolatile analog memory elements for storing a differential voltage on the gates of FG1–FG4, which determines the strength and polarity of the synapse. The particular synapse shown is a recurrent synapse. The *trigger* and *trigger* signals are generated from the neurons’ spike outputs (e.g., the spike output from Neuron 1 is routed to the trigger input of all the synapses in the column labeled ‘Recurrent Input 1’ in Figure 5.9). External and tonic bias synapses are not gated by trigger signals

the external signal. This latter value is conveyed to the neurons by a circuit that scales the amplitude of V_{synbias} by the amplitude of the external signal voltage and applies the result to all 24 OTAs responsive to that input (refer to Figures 5.12 and 5.9). As seen in Figure 5.13, the circuit simply combines a current conveyor, a translinear circuit, and a current mirror to generate an output current proportional to the external voltage:

$$I_{\text{out}} = \frac{I_{\text{sense}} \cdot I_{\text{synbias}}}{I_{\text{bp}}},$$

where I_{bp} is a scaling constant, I_{sense} is a current proportional to the amplitude of the external input voltage, and the voltage V_{out} generated by I_{out} is applied to the gate of M_1 in the OTAs.

5.3 Neuromorphic CPGs at Work

We start this discussion following the neuromorphic design and construction of the silicon CPG (SiCPG) and describe its application as a neuroprosthesis.

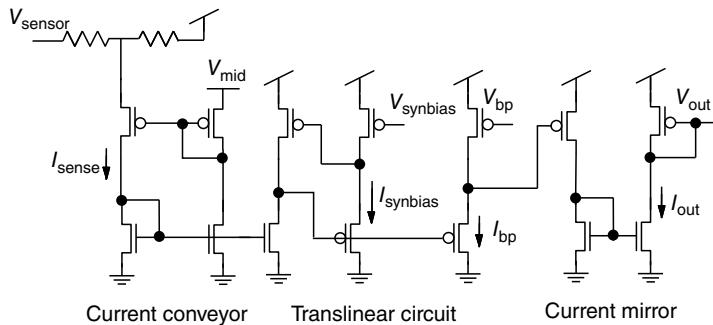


Figure 5.13 Schematic for external input scaling circuit. The external signal is applied to V_{sensor} , and the output V_{out} is applied to the gate of M1 in the individual synapses (Figure 5.12)

5.3.1 A Neuroprosthesis: Control of Locomotion *in Vivo*

The SiCPG described in Section 5.2.5 was used to realize the *in vivo* control of cat hind limbs for locomotion (Vogelstein et al. 2008). The authors recognized that many of the key aspects of recreating the animal's natural gait lie in the correct implementation of a CPG controller. If the artificial CPG is able to reproduce the basic control signals expected by key motor neurons, then a stable locomotor pattern should emerge upon application of the SiCPG in a cat that had been paralyzed.

Although the walking gait of the cat is well studied, it was understood that all the nuances of the cat hind limb locomotion would not be addressed with the limited number of CPG neurons and crude stimulus delivery methods. Instead, Vogelstein et al. (2008) describe the re-characterization of the cat hind limb stance and swing phases into four coarse muscle groups for right and left extensor and flexors activations. Specifically, these motor groups control the flexion or extension of the hip, knee, and ankle resulting in 12 individual intramuscular electrode sites. Dual channel constant-current were used to convert the output of the SiCPG to pulses of electrical activity that could stimulate the motor neurons at the specified sites.

Figure 5.14 illustrates the CPG implemented on the programmable SiCPG. The four neurons control the flexion or extension of the right and left hip, knee, and ankle of the hind limb. Using HCO structures, each neuron is reciprocally coupled with the appropriate antagonistic muscle group or contralateral muscle group, as in the case with the left and right flexor neurons. Swing and stance phases are regulated by load and hip angle sensors. For example, load applied to the left leg of the cat ensures that the right leg maintains its extension to provide stability during stance. Once the right leg has reached its maximum extension behind the body, we expect the right flexion signal to be activated by the hip angle sensors and initiate the right leg swing phase. Sensory feedback from the loading of the ankle extensors (Duyens and Pearson 1980) and the degree of hip extension (Grillner and Rossignol 1978) is critical in the maintenance of the phases in the cat's gait cycle. Supplying the sensory feedback are two-axis accelerometers and ground reaction force plates. These signals can be seen in Figure 5.14 as load and flexion/extension inputs to the CPG.

Results from this SiCPG application prove the technology to be successful as a basic neuroprosthetic hind limb CPG controller (see Figure 5.15). However, as with nearly all

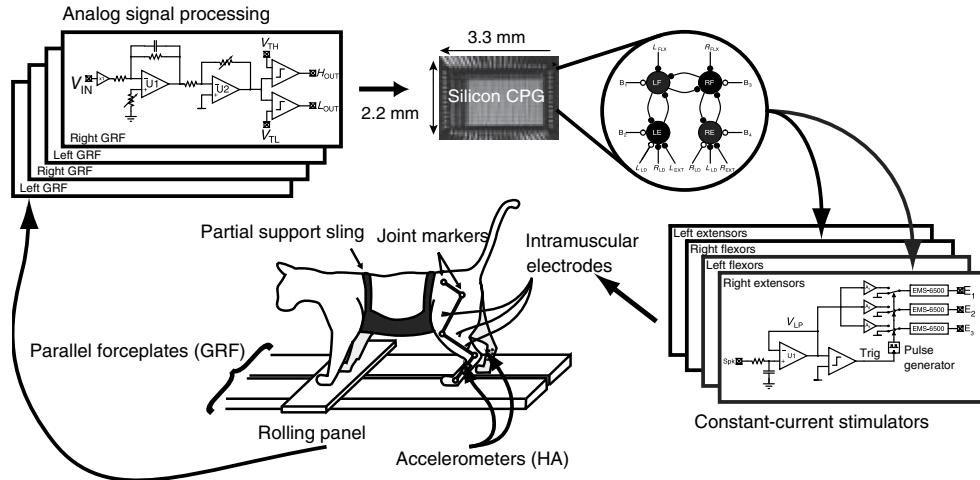


Figure 5.14 The locomotor control system in Vogelstein et al. (2008). The labels LE/RE/LF/RF: left/right extensor/flexor represent silicon neurons. Labeled connectors indicate sensors: $L_{FLX}/R_{FLX}/L_{EXT}$ / R_{EXT} : left/right flexion/extension sensors, L_{LD}/R_{LD} : left/right load sensor, and B_1-B_4 are tonic bias inputs for each neuron. Filled circles at the end of the connections denote inhibitory connections and unfilled circles excitatory ones. Each neuron receives sensory feedback about the current state of the hind limb. This information is used to appropriately engage or disengage muscle groups during the walking cycle via intramuscular electrodes. © 2008 IEEE. Reprinted, with permission, from Vogelstein et al. (2008)

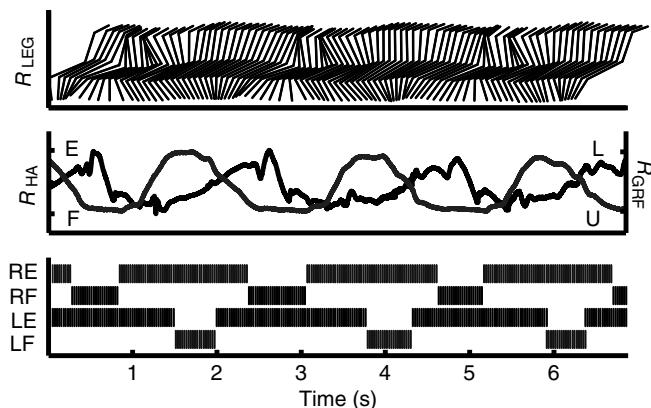


Figure 5.15 Data from Vogelstein et al. (2008): Joint position (top), ground reaction force (R_{GRF}) and hip angle (R_{HA}) (middle), and SiCPG output for each neuron (bottom). © 2008 IEEE. Reprinted, with permission, from Vogelstein et al. (2008)

prosthetics today (especially those that replace outward extremities), the restoration of *natural* function is not expected and there are typically limitations to how adaptable the devices are to varying situations. Such is the case with the SiCPG. Although the authors were able to demonstrate its ability to control the locomotor behavior of the cat hind limb, there were several departures from the natural gait of intact specimen. Most of these departures can be attributed to the coarse level of detail in which: the SiCPG discretizes the normal gait cycle; motor neurons are stimulated; sensory information is integrated as feedback; and limitations in the system design. Despite the modest results, the technology presents itself as a major step in the realization of CPG prosthetics because it was designed with the intent of being implantable and communicating directly. The neuromorphic architecture built into the SiCPG gives advantages in both the power constraints that implantable devices must follow as well as being able to provide action potential-like signals as outputs (Vogelstein et al. 2008).

5.3.2 Walking Robots

A great example of a holistic neuromorphic system comes from Lewis et al. (2001). Their design contains all of the basic elements that one would find in a biological entity such as oscillating units, motor output, and sensory feedback to regulate the behavior of the overall system. Their work is the first such implementation to account for all of these facets in a manner that allows for online adaptation of environmental perturbations. Evidence of how sensor feedback enables the robust performance of the bipedal robot is presented below.

Figure 5.16 details the mechanisms that allow for their design to be adaptive via sensory feedback. From the schematic we see that an oscillating unit is used to set the duration of the motoneurons' spiking and consequently how long the motor is active. The speed at which the motor will operate is determined by the spike frequency of each motor neuron (flexion and extension of the hip). The stretch receptors monitor the activity of the hip joint's position. The feedback from the stretch receptors is then used to control the direction of actuation, whether

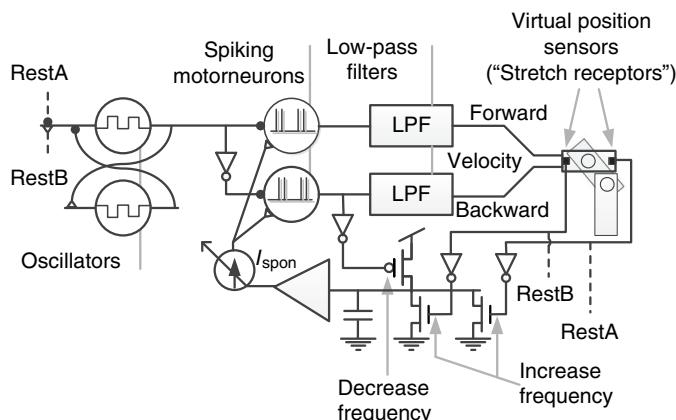


Figure 5.16 Schematic of an adaptive sensory feedback circuit whose frequency is modulated by the position of the stretch receptors. This mechanism allows the controller to maintain a stable gate pattern even as conditions change. © 2000 IEEE. Reprinted, with permission, from Lewis et al. (2000)

it should be in flexion or extension, as well as the speed of the motion. Because the authors make use of simple I&F neuron models, the relationship between input/output behavior is more amenable to online adaptation via their sensory feedback implementation.

Mechanically, the robot consists of legs that are posturally controlled by a supporting arm. Actuation is controlled from the hip joint and maintained through an active swing phase. The knee is a passive joint. The design is based on the entrainment of a pacemaker neuron that regulates the swing phase of the robotic hip joint. Neuromorphically, the locomotor system is constructed from basic I&F neurons which allows for easy analysis of locomotor gate parameters using electrical values. The first neural element is the pacemaker neuron which emulates the behavior of the HCO described in the introduction section of this chapter. The neuron is nonspiking but maintains the envelope which controls the activation of hip ‘motoneurons’. Here a square pulse is generated such that when firing, motoneurons are inhibited. For one hip motoneuron (for clarity let us refer to this motoneuron as motoneuron A), when the pacemaker neuron releases it from inhibition, it is actively in the swing phase, while the other (motoneuron B) is disengaged, not active. Conversely, when the pacemaker neuron is firing, motoneuron B becomes active. This represents the inhibition of motoneuron A’s controlling oscillator element. An inverter is used to allow motoneuron B to be released from inhibition during this period. Unlike the pacemaker neurons, the motoneurons in the system are spiking neurons whose spiking frequency controls the velocity of hip swing.

Feedback is directed to both the pacemaker neurons and the motoneurons. Phase resetting of the pacemaker neuron is modulated by a stretch receptor monitoring the position of the hips. Once a leg has reached its maximum stretch, the pacemaker neuron resets, driving the opposing leg forward. The appropriate feedback connection must be specified to allow the pacemaker neuron to be reset to the correct phase. Specifically, to reset the pacemaker neuron after the leg controlled by motoneuron A has reached its maximal stretch, we require an excitatory feedback connection to shunt motoneuron A spiking and thus enable motoneuron B to be engaged. When the leg controlled by motoneuron B has reached its maximal position, an inhibitory current is supplied to the pacemaker neuron which causes it to release its inhibitory hold on motoneuron A and shunt motoneuron B. This behavior phase-locks the pacemaker neurons output to always be aligned with the appropriate gait phase. Thus, like the PDR in real biological systems, the gait can be adaptively controlled should the hip positions be altered via perturbations.

The last neuromorphic element considered in this design is the learning circuitry that ensures legs are driven to their maximal point before resetting. This feature is important because it dynamically adjusts the excitability of the motoneurons allowing them to spike more or less depending on feedback from the stretch receptors. If a leg fails to reach its maximal stretch, the motor neuron for that leg will increase its swing amplitude until the stretch receptor responds. Activation of the stretch receptor helps maintain the hip swing amplitude by decreasing it if it is overdriven. This allows for a stable gait pattern to be achieved even if internal parameters of the system change or new parts are added to the mechanical design.

5.3.3 Modeling Intersegmental Coordination

Leeches and lampreys use phase-dependent activation of several CPGs located in the various segments of the lamprey’s spinal cord and leech’s ganglia to generate the rhythmic motor

pattern for swimming (Brodfuehrer et al. 1995). From an engineering standpoint, study of lamprey and leech swimming demonstrates the ability of biological systems to realize complex motor control and adaptation dynamics using interconnected blocks of nearly identical subunits. Thus, in practice, they represent an ideal model for VLSI implementation of biological circuits. Nevertheless, there are caveats to such endeavors. DeWeerth et al. (1997) model intersegmental coordination and examine the benefits and challenges of realizing such a design and we draw upon their methods for insight. The AER protocol described in Chapter 2 is the key to enabling unit segmental oscillators (USO) to communicate locally among neurons that make up its unit, and more importantly, nonlocally with other units to achieve the complex characteristic of natural motion. (Unit here refers to the most elementary segmental oscillator within the biological system of a leech, lamprey, or artificial neural system. The USO can be readily implemented using a pair of reciprocally inhibitory neurons, the HCO.) Patel et al. (2006) made use of a modified version of the protocol which they developed to eliminate the dependence of static addressing for neurons and configured delay parameters, thereby making the overall system a better model of the biological systems they sought to emulate and more conducive to experimental assays.

5.4 Discussion

Early works in describing neurons mathematically and implementing them in silicon have allowed scientists to build models and explore some basic principles in locomotor behavior (see Figure 5.17), yet, there are still many unanswered questions. Such issues involve implementing postural control in walking robots using a neuromorphic framework and developing invasive

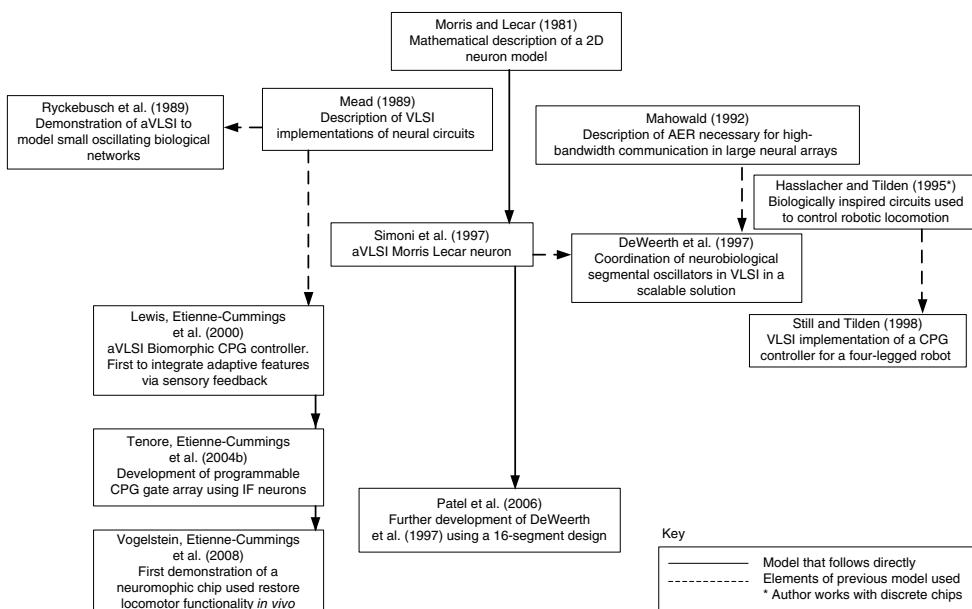


Figure 5.17 Historical tree of CPG chips

and noninvasive solutions for rehabilitative technologies with neuromorphic hardware for persons with locomotor dysfunction.

Terrestrial vertebrates have little trouble maintaining their balance while stationary or in motion. Postural control is largely innate and involuntary but it is complicated because of the numerous body parts involved and the high degree of freedom (DOF) attributed to the head, body, and extremities individually and collectively. Coordination of the head, body, and extremities involves the monitoring of various sensory afferents including vestibular and visual cues to maintain head position and numerous inertial, tactile, vestibular, and proprioceptive signaling of relative and global limb positions and help dictate appropriate muscle activations. In a somewhat restrictive setting, robotic systems can maintain stability through carefully planned movements that are calculated to keep the center of gravity of the robot within a stable region using zero moment point (ZMP) calculations (ASIMO, WABIAN-2R). ZMP designs favor a flat-footed placement of steps, and some compensatory stance, such as bent knees, to ensure the flat-footed placement. The flat-footed placement includes the initial contact of the foot with the ground during loading after the swing phase as well as during the initiation of the swing phase of the opposite leg. Thus, in many designs, heel striking and toe-off are not part of the robot mechanics (with the exception of WABIAN-2R) and rotation of the foot about the heel, toe, or ankle is minimized since such activity creates unstable ZMP criteria (Vukobratovic and Borovac 2004). Humans naturally transition the center of pressure (CoP) on their feet from heel to toe in order to preserve the transfer of energy in the forward direction after contact with the ground, however, such dynamics complicate ZMP design (Westervelt et al. 2007). ZMP postural control systems although popular and successful do not approach the flexibility afforded to biological entities because they rely on limited DOF systems, inverse kinematic descriptions of the mechanical system, and linearized walking models to satisfy ZMP equations. These facets of the underlying design can limit the ability of the robot to perform more rigorous tasks, handle novel environmental conditions, and adapt to changes in its kinematic description. In contrast to designs like ASIMO, others have tackled postural control in walking robots by using very little control hardware but rather relying on passive dynamics to balance coordination and efficient mimicry of bipedal gait patterns (Collins et al. 2005). Both of these control strategies have the potential to produce stable locomotion but tend to be restrictive in application and overall capabilities. Applying neuromorphic design principles to posture control may help create a more robust system that operates through the reflex activation of actuators based on monitoring force, torque, and position sensors and does not rely on strict kinematic definitions. Furthermore, such an approach may yield faster reaction time, online adaptation to maintain gait stability, and better energy efficiency. Although investigators have strategies in place that explain the various processing that motor cortex, cerebellum, brainstem, spinal cord and interneuron circuits perform to enable posture and reflex, explicit detail about connections and neural communication is still lacking, thus gating the progression of more complete robotic designs involving neuromorphic design.

In addition to the use of neuromorphic hardware in robotics, some research endeavors are exploring the use of such technology as a prosthesis to enable locomotor function in individuals with lost limbs, spinal cord injury, or neurological disorders affecting mobility. In such cases neuromorphic hardware would adopt the role of biological CPGs that are no longer able to receive descending commands from the brain to engage muscles for walking, or communicate with intact biological CPGs in a meaningful way so as to reinstitute functional motor behavior. However, integration of such a device within the nervous system has some caveats when we compare ease of implementation with practicality for long-term use as a prosthetic device. It

also raises the issue of the clinical acceptance of neuromorphic hardware as a prosthetic since the long-term effects of implantable devices and chronic electrode stimulation is still being investigated.

We have described neuromorphic hardware being used to control the hind limb of a paralyzed animal preparation using intramuscular stimulation (IMS) (Vogelstein et al. 2008). The framework is simple and does not require any understanding of how the animal's own CPGs are structured and organized. All that is required is that the muscles controlling movement are activated in a way that approximates the animal's natural gait and this is efficiently achieved through neuromorphic design principles. Still, this solution does not represent the optimal entry point for communication with neuromorphic hardware and the biological system. Ideally, if spinal circuits for locomotion are preserved, we would like our neuromorphic hardware to communicate with motor circuits in the spinal cord directly through intraspinal micro stimulation (ISMS) rather than the motor neurons. There are several advantages in adopting an ISMS framework, the foremost being the low-stimulation currents used to communicate with intraspinal circuits in comparison with the large motor neurons and the mitigation of muscle fatigue which has been observed in other prosthetic systems relying on stimulation of motor neurons. Nevertheless, ISMS presents more challenges for neuromorphic design because it demands that one understands how to stimulate intraspinal circuits to produce functional limb movement and the ability to provide appropriate feedback to intraspinal neurons for overall robustness. A relevant question in this domain is whether or not ISMS should be implemented within locomotor circuits alone or if interfacing with higher level cerebral signals is also necessary. What complications may arise if descending locomotor commands do not match local CPG outputs produced from ISMS? Many of these challenges are still open research interests for biologists and engineers.

In lieu of an implanted device, one could also imagine the use of a mechanical prosthesis that fits to a patient like an exoskeleton, and that can be actuated without the need for invasive monitoring of biological signals. Historically, the technology has been mostly geared toward individuals who have limited control over limb activity, whereby the exoskeleton provides a mechanical assist to the individual for daily use or in rehabilitation. One of the more advanced exoskeletons of today (the 'Robo Suit Hal', Cyberdyne) derives motor intent based on monitoring weak myoelectric signals of various motor groups. Much of the success of the suit is owed to the synergistic use of human motor intent and the robotic systems ability to accurately assist and carry out that intent. For individuals lacking the ability to provide such motor intent to an assistive exoskeleton, the use of a more autonomous robotic controller would be necessary. To date however, the benefit of using a neuromorphic controller with an exoskeleton has not been explored. Many of the challenges and benefits presented for such a system mirror those of designing neuromorphic controllers for fully autonomous robots discussed above. However, the requirements for continuous adaptation and learning of future neuromorphic controllers will involve the subjects of the next chapter, on learning in neuromorphic systems.

References

- Arena P, Fortuna L, Frasca M, Patane L, and Vagliasindi G. 2005. CPG-MTA implementation for locomotion control. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, **4**, pp. 4102–4105.
- Boahen KA. 1997. *Retinomorphic Vision Systems: Reverse Engineering the Vertebrate Retina*. PhD thesis. California Institute of Technology, Pasadena, CA.

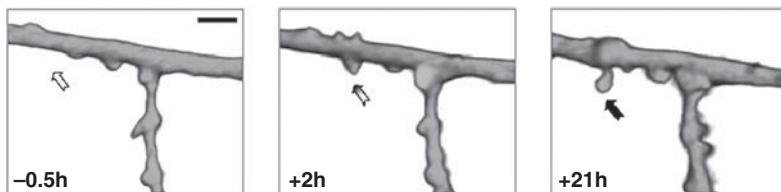
- Brodfuehrer PD, Debski EA, O'Gara BA, and Friesen WO. 1995. Neuronal control of leech swimming. *J. Neurobiol.* **27**(3), 403–418.
- Cohen AH and Wallén P. 1980. The neuronal correlate of locomotion in fish. *Exp. Brain Res.* **41**(1), 11–18.
- Cohen AH, Holmes PJ, and Rand RH. 1982. The nature of the coupling between segmental oscillators of the lamprey spinal generator for locomotion: a mathematical model. *J. Math. Biol.* **13**, 345–369.
- Cohen AH, Rossignol S, and Grillner S. 1988. *Neural Control of Rhythmic Movements in Vertebrates*. John Wiley & Sons, Inc., New York.
- Collins S, Ruina A, Tedrake R, and Wisse M. 2005. Efficient bipedal robots based on passive-dynamic walkers. *Science* **307**(5712), 1082–1085.
- Dale N. 1995. Experimentally derived model for the locomotor pattern generator in the *Xenopus* embryo. *J. Physiol.* **489**, 489–510.
- Deliagina TG, Zelenin PV, and Orlovsky GN. 2002. Encoding and decoding of reticulospinal commands. *Brain Res. Rev.* **40**(1–3), 166–177.
- DeWeerth SP, Patel GN, Simoni MF, Schimmel DE, and Calabrese RL. 1997. A VLSI architecture for modeling intersegmental coordination. *Proc. 17th Conf. Adv. Res. VLSI*, pp. 182–200.
- Diorio C. 2000. A p-channel MOS synapse transistor with self-convergent memory writes. *IEEE Trans. Elect. Devices* **47**(2), 464–472.
- DuySENS J and Pearson KG. 1980. Inhibition of flexor burst generation by loading ankle extensor muscles in walking cats. *Brain Res.* **187**(2), 321–332.
- Fedorichuk B, Nielsen J, Petersen N, and Hultborn H. 1998. Pharmacologically evoked fictive motor patterns in the acutely spinalized marmoset monkey (*Callithrix jacchus*). *Exp. Brain Res.* **122**(3), 351–361.
- Getting PA. 1989. Emerging principles governing the operation of neural networks. *Annu. Rev. Neurosci.* **12**, 185–204.
- Gosgnach S, Lanuza GM, Butt SJB, Saueressig H, Zhang Y, Velasquez T, Riethmacher D, Callaway EM, Kiehn O, and Goulding M. 2006. V1 spinal neurons regulate the speed of vertebrate locomotor outputs. *Nature* **440**(7081), 215–219.
- Graham-Brown T. 1911. The intrinsic factors in the act of progression in the mammal. *Proc. R. Soc. Lond. B* **84**(572), 308–319.
- Grillner S. 2003. The motor infrastructure: from ion channels to neuronal networks. *Nature Rev. Neurosci.* **4**, 573–586.
- Grillner S and Rossignol S. 1978. On the initiation of the swing phase of locomotion in chronic spinal cats. *Brain Res.* **146**(2), 269–277.
- Grillner S and Wallén P. 1985. Central pattern generators for locomotion, with special reference to vertebrates. *Annu. Rev. Neurosci.* **8**(1), 233–261.
- Grillner S and Zanger P. 1979. On the central generation of locomotion in the low spinal cat. *Exp. Brain Res.* **34**(2), 241–261.
- Grillner S, Wallén P, Brodin L, and Lansner A. 1991. Neural network generating locomotor behavior in lamprey: circuitry, transmitters, membrane properties, and simulation. *Annu. Rev. Neurosci.* **14**, 169–199.
- Grillner S, Ekeberg O, El Manira A, Lansner A, Parker D, Tegnér J, and Wallén P. 1998. Intrinsic function of a neuronal network – a vertebrate central pattern generator. *Brain Res. Rev.* **26**(2–3), 184–197.
- Grillner S, Cangiano L, Hu GY, Thompson R, Hill R, and Wallén P. 2000. The intrinsic function of a motor system – from ion channels to networks and behavior. *Brain Res.* **886**(1–2), 224–236.
- Harris-Warrick RM. 1990. Mechanisms for neuromodulation of biological neural networks. In: *Advances in Neural Information Processing Systems 2 (NIPS)* (ed. Touretzky D). Morgan Kaufman, San Mateo, CA. pp. 18–27.
- Hasslacher B and Tilden MW. 1995. Living machines. *Robot. Auton. Syst.* **15**(1–2), 143–169.
- Hill AAV, Masino MA, and Calabrese RL. 2003. Intersegmental coordination of rhythmic motor patterns. *J. Neurophys.* **90**(2), 531–538.
- Hultborn H. 2001. State-dependent modulation of sensory feedback. *J. Physiol.* **533**(1), 5–13.
- Huss M, Lansner A, Wallén P, El Manira A, Grillner S, and Kotaleski JH. 2007. Roles of ionic currents in lamprey CPG neurons: a modeling study. *J. Neurophys.* **97**(4), 2696–2711.
- Ijspeert AJ. 2001. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biol. Cybern.* **84**(5), 331–348.
- Ijspeert AJ. 2008. Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw.* **21**, 642–653.
- Indiveri G. 2003. Neuromorphic bistable VLSI synapses with spike-timing dependent plasticity. In: *Advances in Neural Information Processing Systems 15 (NIPS)* (eds Becker S, Thrun S, and Obermayer K). MIT Press, Cambridge, MA. pp. 1115–1122.

- Kiehn O. 2006. Locomotor circuits in the mammalian spinal cord. *Annu. Rev. Neurosci.* **29**, 279–306.
- Kier RJ, Ames JC, Beer RD, and Harrison RR. 2006. Design and implementation of multipattern generators in analog VLSI. *IEEE Trans. Neural Netw.* **17**(4), 1025–1038.
- Klavins E, Komsuoglu H, Full RJ, and Koditschek DE. 2002. The role of reflexes versus central pattern generators in dynamical legged locomotion. In: *Neurotechnology for Biomimetic Robots* (eds Ayers J, Davis JL, and Rudolph A). MIT Press. pp. 351–382.
- Lay AN, Hass CJ, and Gregor RJ. 2006. The effects of sloped surfaces on locomotion: a kinematic and kinetic analysis. *J. Biomech.* **39**(9), 1621–1628.
- Lay AN, Hass CJ, Nichols TR, and Gregor RJ. 2007. The effects of sloped surfaces on locomotion: an electromyographic analysis. *J. Biomech.* **40**(6), 1276–1285.
- Lenzlinger M and Snow EH. 1969. Fowler-Nordheim tunneling into thermally grown SiO₂. *J. Appl. Phys.* **40**(1), 278–283.
- Lewis MA, Etienne-Cummings R, Cohen AH, and Hartmann M. 2000. Toward biomorphic control using custom aVLSI CPG chips. *Proc. IEEE Int. Conf. Robot. Automat.* **1**, pp. 494–500.
- Lewis MA, Hartmann MJ, Etienne-Cummings R, and Cohen AH. 2001. Control of a robot leg with an adaptive aVLSI CPG chip. *Neurocomputing* **38-40**, 1409–1421.
- Lewis MA, Tenore F, and Etienne-Cummings R. 2005. CPG design using inhibitory networks *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3682–3687.
- Li WC, Perrins R, Walford A, and Roberts A. 2003. The neuronal targets for GABAergic reticulospinal inhibition that stops swimming in hatchling frog tadpoles. *J. Comp. Physiol. [A]: Neuroethology, Sensory, Neural, and Behavioral Physiology* **189**(1), 29–37.
- Mahowald M. 1992. VLSI analogs of neural visual processing: A synthesis of form and function. PhD thesis. California Institute of Technology.
- Marder E and Bucher D. 2001. Central pattern generators and the control of rhythmic movements. *Curr. Biol.* **11**(23), R986–R996.
- Marder E and Calabrese RL. 1996. Principles of rhythmic motor pattern generation. *Physiol. Rev.* **76**(3), 687–717.
- Matsuyama K, Mori F, Nakajima K, Drew T, Aoki M, and Mori S. 2004. Locomotor role of the corticoreticular-reticulospinal-spinal interneuronal system. In: *Brain Mechanisms for the Integration of Posture and Movement* (eds Mori S, Stuart DG, and Wiesendanger M). vol. **143** of *Prog. Brain Res.* Elsevier. pp. 239–249.
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Morris C and Lecar H. 1981. Voltage oscillations in the barnacle giant muscle fiber. *Biophys. J.* **35**(1), 193–213.
- Nakada K, Asai T, Hirose T, and Amemiya Y. 2005. Analog CMOS implementation of a neuromorphic oscillator with current-mode low-pass filters. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 1923–1926.
- Nistri A, Ostroumov K, Sharifullina E, and Taccolla G. 2006. Tuning and playing a motor rhythm: how metabotropic glutamate receptors orchestrate generation of motor patterns in the mammalian central nervous system. *J. Physiol.* **572**(2), 323–334.
- Orlovsky GN, Deliagina TG, and Grillner S. 1999. *Neuronal Control of Locomotion: From Mollusc to Man*. Oxford University Press, New York.
- Patel GN, Reid MS, Schimmel DE, and DeWeerth SP. 2006. An asynchronous architecture for modeling intersegmental neural communication. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **14**(2), 97–110.
- Quinn R, Nelson G, Bachmann R, and Ritzmann R. 2001. Toward mission capable legged robots through biological inspiration. *Auton. Robot.* **11**(3), 215–220.
- Reeve R and Hallam J. 2005. An analysis of neural models for walking control. *IEEE Trans. Neural Netw.* **16**(3), 733–742.
- Rybak IA, Shevtsova NA, Lafreniere-Roula M, and McCrea DA. 2006a. Modelling spinal circuitry involved in locomotor pattern generation: insights from deletions during fictive locomotion. *J. Physiol.* **577**, 617–639.
- Rybak IA, Shevtsova NA, Lafreniere-Roula M, and McCrea DA. 2006b. Modelling spinal circuitry involved in locomotor pattern generation: insights from the affects of afferent stimulation. *J. Physiol.* **577**, 641–658.
- Ryckebusch S, Bower JM, and Mead C. 1989. Modeling small oscillating biological networks in analog VLSI. In: *Advances in Neural Information Processing Systems 1 (NIPS)* (ed Touretzky D). Morgan-Kaufmann, San Mateo, CA. pp. 384–393.
- Schultz SR and Jabri MA. 1995. Analogue VLSI integrate-and-fire neuron with frequency adaptation. *Electron. Lett.* **31**(16), 1357–1358.
- Silverston AI. 2010. Invertebrate central pattern generator circuits. *Phil. Trans. R. Soc. B* **365**, 2329–2345.
- Shik ML and Orlovsky GN. 1976. Neurophysiology of locomotor automatism. *Physiol. Rev.* **56**(3), 465–501.

- Shin J and Koch C. 1999. Dynamic range and sensitivity adaptation in a silicon spiking neuron. *IEEE Trans. Neural Netw.* **10**(5), 1232–1238.
- Sigvardt KA and Miller WL. 1998. Analysis and modeling of the locomotor central pattern generator as a network of coupled oscillators. *Ann. N.Y. Acad. Sci.* **860**(1), 250–265.
- Sigvardt KA and Williams TL. 1996. Effects of local oscillator frequency on intersegmental coordination in the lamprey locomotor CPG: theory and experiment. *J. Neurophys.* **76**(6), 4094–4103.
- Simoni MF, Cymbalyuk GS, Sorensen ME, Calabrese RL, and DeWeerth SP. 2004. A multiconductance silicon neuron with biologically matched dynamics. *IEEE Trans. Biomed. Eng.* **51**(2), 342–354.
- Simoni MF, Patel GN, DeWeerth SP, and Calabrese RL. 1998. Analog VLSI model of the leech heartbeat elemental oscillator. In: *Computational Neuroscience: Trends in Research, 1998. Proceedings of the 6th Annual Computational Neuroscience Conference* (ed Bower JM). Kluwer Academic/Plenum Publishers.
- Sponberg S and Full RJ. 2008. Neuromechanical response of musculo-skeletal structures in cockroaches during rapid running on rough terrain. *J. Exp. Biol.* **211**(3), 433–446.
- Still S and Tilden MW. 1998. Controller for a four legged walking machine. In: *Neuromorphic Systems: Engineering Silicon from Neurobiology* (ed Smith LS and Hamilton A). World Scientific, Singapore. pp. 138–148.
- Still S, Hepp K, and Douglas RJ. 2006. Neuromorphic walking gait control. *IEEE Trans. Neural Netw.* **17**(2), 496–508.
- Tenore F, Etienne-Cummings R, and Lewis MA. 2004a. Entrainment of silicon central pattern generators for legged locomotory control. In: *Advances in Neural Information Processing Systems 16 (NIPS)* (eds Thrun S, Saul L, and Schölkopf B). MIT Press, Cambridge, MA.
- Tenore F, Etienne-Cummings R, and Lewis MA. 2004b. A programmable array of silicon neurons for the control of legged locomotion. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, **5**, 349–352.
- Tenore F, Vogelstein RJ, Etienne-Cummings R, Cauwenberghs G, Lewis MA, and Hasler P. 2005. A spiking silicon central pattern generator with floating gate synapses. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, **IV**, 4106–4109.
- Van de Crommert HWAA, Mulder T, and Duysens J. 1998. Neural control of locomotion: sensory control of the central pattern generator and its relation to treadmill training. *Gait Posture* **7**(3), 251–263.
- Vogelstein RJ. 2007. *Towards a Spinal Neuroprosthesis: Restoring Locomotion After Spinal Cord Injury*. PhD thesis. The Johns Hopkins University.
- Vogelstein RJ, Tenore F, Etienne-Cummings R, Lewis M, and Cohen A. 2006. Dynamic control of the central pattern generator for locomotion. *Biol. Cybern.* **95**(6), 555–566.
- Vogelstein RJ, Tenore F, Guevremont L, Etienne-Cummings R, and Mushahwar VK. 2008. A silicon central pattern generator controls locomotion in vivo. *IEEE Trans. Biomed. Circuits Syst.* **2**(3), 212–222.
- Vukobratovic M and Borovac B. 2004. Zero-moment point – thirty five years of its life. *Int. J. Hum. Robot.* **1**(1), 157–173.
- Wannier T, Orlovsky G, and Grillner S. 1995. Reticulospinal neurones provide monosynaptic glycinergic inhibition of spinal neurones in lamprey. *NeuroReport* **6**(12), 1597–1600.
- Westervelt E, Grizzle JW, Chevallereau C, Choi JH, and Morris B. 2007. *Feedback Control of Dynamic Bipedal Robot Locomotion*. CRC Press.
- Whittle M. 1996. *Gait Analysis: An Introduction*, 2nd edn. Butterworth-Heinemann, Oxford.
- Zelenin PV. 2005. Activity of individual reticulospinal neurons during different forms of locomotion in the lamprey. *Eur. J. Neurosci.* **22**(9), 2271–2282.
- Zelenin PV, Grillner S, Orlovsky GN, and Deliagina TG. 2001. Heterogeneity of the population of command neurons in the lamprey. *J. Neurosci.* **21**(19), 7793–7803.

6

Learning in Neuromorphic Systems



In this chapter, we address some general theoretical issues concerning synaptic plasticity as the mechanism underlying learning in neural networks, in the context of neuromorphic VLSI systems, and provide a few implementation examples to illustrate the principles. It ties to Chapters 3 and 4 on neuromorphic sensors by proposing theoretical means for utilizing events for learning. It is an interesting historical fact that when considering the problem of synaptic learning dynamics within the constraints imposed by implementation, a whole new domain of previously overlooked theoretical problems became apparent, which led to rethinking much of the conceptual underpinning of learning models. This mutual cross-fertilization of theory and neuromorphic engineering is a nice example of the heuristic value of implementations for the progress of theory.

We first discuss some general limitations arising from plausible constraints to be met by any implementable synaptic device for the working of a neural network as a memory; we then illustrate strategies to cope with such limitations, and how some of those have been translated into design principles for VLSI plastic synapses. The resulting type of Hebbian, bistable, spike-driven, stochastic synapse is then put in the specific, but wide-scope, context of associative memories based on attractor dynamics in recurrent neural networks. After briefly reviewing key theoretical concepts we discuss some frequently overlooked problems arising from the finite (actually small) neural and

Two-photon microscopy images of a calcein-filled CA1 pyramidal neuron showing newly growing spines. Adapted from Figure 1 of Nägerl et al. (2007). Reproduced with permission of the Society for Neuroscience.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

synaptic resources in a VLSI chip. We then illustrate two implementation examples: the first demonstrates the dynamics of memory retrieval in an attractor VLSI network and explains how a good procedure for choosing interesting regions in the network's parameter space can be derived from the mean field theory developed for attractor network; the second is a learning experiment, in which a stream of visual stimuli (acquired in real time through a silicon retina) constitutes the input to the spiking neurons in the VLSI network, thereby driving continuously changes in the synaptic strengths and building up representations of stimuli as attractors of the network's dynamics.

6.1 Introduction: Synaptic Connections, Memory, and Learning

Artificial neural networks can imitate several functions of the brain, ranging from sensory processing to high cognitive functions. For most of the neural network models, the function of the neural circuits is the result of complex synaptic interactions between neurons. Given that the matrix of all synaptic weights is so important, it is natural to ask: how is it generated? How can we build a neural network that implements a certain function or that imitates the activity recorded in the brain? And finally, how does the brain construct its own neural circuits? Part of the structure observed in the pattern of synaptic connections is likely to be genetically encoded. However, we know that synaptic connections are plastic and they can be modified by the neural activity. Every time we go through some new experience, we observe some specific pattern of neural activity, that in turn modifies the synaptic connections. This process is the basis of long-term memory, as the mnemonic trace of that experience is most likely retained by the modified synapses, even when the experience is over. Memories can then be reactivated at a later time by a pattern of neural activity that resembles the one that created the memory, or more in general, that is strongly affected by the synapses that have been modified when the memory was created. This form of memory is also fundamental for any learning process in which we modify our behavior on the basis of relevant past experiences.

There are several models of artificial neural networks (see, e.g., Hertz et al. 1991) that are able to learn. They can be divided into three main groups: (1) networks that are able to create representations of the statistics of the world in an autonomous way (unsupervised learning) (Hinton and Sejnowski 1999); (2) networks that can learn to perform a particular task when instructed by a teacher (supervised learning) (Anthony and Bartlett 1999); and (3) networks that can learn by a trial and error procedure (reinforcement learning Sutton and Barto 1998). These categories can have a different meaning and different nomenclature depending on the scientific community (machine learning or theoretical neuroscience). All these models rely on some form of efficient long-term memory. Building such networks in analog electronic circuits is difficult, as analog memories are usually volatile, or when they are stable the synaptic modifications can be remembered with a limited precision. In the next section we review the memory problem for neuromorphic hardware and some of the solutions proposed in the last two decades. We first discuss how memories can be retained over long timescales (memory maintenance); then we address the issue of how new memories are stored (learning) and we discuss the process of memory retrieval in binary and spiking attractor neural networks. Finally, we offer an example of memory retrieval in a neuromorphic, VLSI recurrent spiking neural network, and a simple instance of Hebbian dynamic learning in the same chips, based on the stochastic synaptic dynamics described in the previous sections.

6.2 Retaining Memories in Neuromorphic Hardware

6.2.1 The Problem of Memory Maintenance: Intuition

Highly simplified neural networks like perceptrons (Block 1962; Minsky and Papert 1969; Rosenblatt 1958) and Hopfield networks (Amit 1989; Hopfield 1982) can store and retrieve an enormous number of memories, to the point that, at the beginning of the 1990s, theoretical neuroscientists thought that the memory problem was solved. It was actually in an attempt to implement these models in neuromorphic hardware that Amit and Fusi realized in 1992 that there is a serious fundamental problem related to memory storage (Amit and Fusi 1992 1994). Models of biological synapses are inevitably highly simplified and they try to capture the features of the synaptic dynamics that are important to implement a cognitive function. Amit and Fusi realized that one of the simplifications in the previous modeling work, allowing synaptic strengths to vary in an unlimited range, has dramatic implications for the memory capacity. Networks with realistic, bounded synapses have a strongly reduced memory capacity, because old memories are overwritten by new ones at a very fast rate.

We explain this point with a simple intuitive argument (Fusi and Senn 2006). Consider a neural network that is exposed to a continuous stream of experiences that we would like to store in the memory of the synaptic weights (see Figure 6.1). We now consider one generic synapse and we focus on a particular experience whose memory we intend to track (we name it A). We first consider the case of an unbounded synapse (Figure 6.1a). The synapse starts from some initial state (we set it arbitrarily to zero), and it is modified when the network goes through each experience. For example, its efficacy is strengthened by experience A, as shown in the figure.

We now make the assumption that there is an efficient mechanism of retaining that value of the synaptic strength, at least until the occurrence of a new event that modifies it (B). The maintenance of a precise value might require complex mechanisms, as it certainly does in biological systems, but for now we ignore this issue and we assume that synaptic values can be preserved indefinitely in the absence of new events.

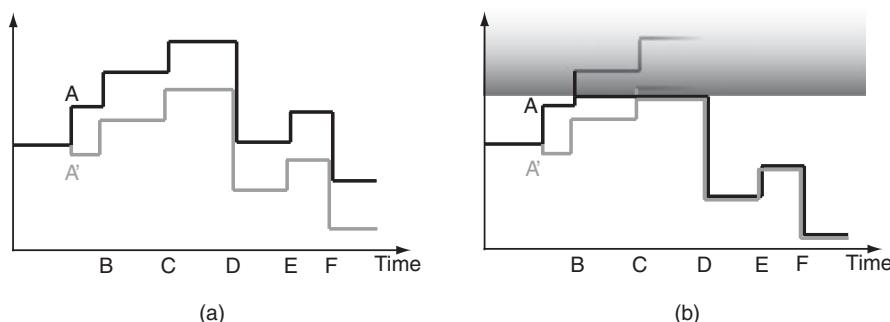


Figure 6.1 Limiting the synaptic strengths between two bounds causes forgetting. (a) An unbounded synapse remembers about experience A. The synaptic value is plotted against time. Every experience (A, ..., F) modifies the synapse either upwards or downwards. (b) The same synapse is now bounded from above (the shaded wall represents a rigid bound for the maximal value of the synapse). The trajectories which can be seen through the wall are those that the synapse would have followed, were not bounded there. Now the modifications induced by A cannot be propagated up to the present time

The final value of the synapse will reflect the series of synaptic modifications induced by all the new events (C,D, ...,F). Is the synapse still retaining any memory of A? One way to answer is to perform the following virtual experiment: we go back to the past, we modify A ($A \rightarrow A'$), and we see whether such a modification can propagate up to the present time. For example, we change experience A in such a way that the synapse is weakened instead of being potentiated. If synaptic modifications simply add up linearly, then the trajectory representing the synaptic weight as a function of time (gray) is parallel to the first one. Obviously the final value will be different, and it will be correlated to the synaptic modification induced by A. So in principle we still have memory of what happened in A, although we might not be able to recollect all the details about the experience A. However, the important point is that the mnemonic trace is still there as the final value of the synaptic strength is correlated with the modification induced by A.

We now consider a bounded synapse (Figure 6.1b). We do exactly the same experiment as we did for the unbounded synapses. Now experience B already brings the synapse to its saturation value (indicated by a shaded wall). The synapse cannot go any further up, so it stays at the maximum, also when experience C tries to potentiate the synapse again. When we now go back to the past and modify A into A', we see that initially the black and the gray trajectories are different, but when the synapse goes through experience C, it hits the upper bound and the two trajectories become identical. In particular, the final value will also be the same, whether the synapse went through A or A'. In this case the modification induced by A cannot affect the final synaptic value. So at the present time A has been forgotten. This simple example shows that every time the variables describing the synaptic dynamics are restricted to vary in a limited range, old memories are naturally forgotten (see also Fusi 2002; Parisi 1986).

6.2.2 The Problem of Memory Maintenance: Quantitative Analysis

Memories are naturally forgotten when the synaptic weights are restricted to vary in a limited range. How fast is such a forgetting? We can easily estimate the typical memory lifetime in a population of bistable synapses for which the lower and the upper bounds coincide with the only two stable states. This may sound as an extreme case of a bounded synapse, but it is actually representative of a wide class of realistic synapses (see Section 6.2.3). When there are only two synaptic states, every time a synapse is modified, the past is entirely forgotten. For example, if one experience wants to strengthen a synapse, then the synapse ends up in the potentiated state, whether it started from the depressed or an already potentiated state. The initial condition preceding the modification contained all the information about previously stored memories, and it is forgotten. As a consequence, every time we go through some new experience, there will be a fraction of synapses that is modified in which we store information about the new experience and we forget the other memories. The remaining synapses are left unchanged, and hence they preserve the old memories, but they do not store any information about the new experience. After experience A, which again leads to the memory that we intend to track, a number $n = qN$ of synapses store information about A (N is the total number of synapses and q is the average fraction of modified synapses). When we go through the experiences B,C,D, ..., we assume that their neural representations are random and uncorrelated. This assumption is motivated by the belief that the brain has an efficient mechanism to compress the information before it stores it. The incompressible part of the information can reasonably be modeled as

some sparse patterns of random and uncorrelated activities. Indeed, were it correlated to one of the previously stored memories, it would still be compressible. We now ask ourselves: how many distinct memories can we store? We assume also that each memory is stored in a single exposure. A second exposure would be equivalent to a highly correlated memory, and we assumed that there is an efficient mechanism to deal with correlations. We will see that there is a problem of memory capacity even under all these simplifying assumptions. When the memories are random and uncorrelated, then after p experiences we have

$$n = Nq(1 - q) \dots (1 - q) = Nq(1 - q)^p \sim Nqe^{-qp}. \quad (6.1)$$

Indeed, the tracked experience A is not special in any way, it is just the one we decided to track. Hence we should use the same rule for updating the synapses for all memories, and that is why q is always the same for every synaptic update. From the formula it is clear that n decays to zero exponentially with p . To retrieve information about the tracked memory, at least one synapse should be able to remember it ($n \geq 1$). This imposes a tight constraint on the number of uncorrelated patterns p which can be stored:

$$p < -\frac{\log N}{q}.$$

The logarithmic dependence on N makes this limitation severe, and the neural network extremely inefficient as a memory. Most of the synaptic resources are devoted to the last stimuli seen, even when the information collected about the stimulus and stored in the synapses is more than what is needed to retrieve correctly the memorized pattern. Notice that the condition that at least one synapse remembers is a necessary condition. In general, it will not be sufficient to retrieve the memory, and sometimes not even to recognize it as familiar. The number of experiences p can also be regarded as the memory lifetime expressed in terms of the number of patterns whose memory trace is still in the present synaptic structure. This memory trace can be so feeble that it might be impossible to retrieve any information about the pattern of activities in the memory sliding window.

This argument, made for simplicity in the case of bistable synapses, actually applies to any realistic memory system and it can be made rigorous for a wide class of synaptic models (see Fusi 2002).

In the case of traditional neural network models, most of the synaptic models had weights that could vary in an unlimited range, or similarly, the weights could be modified with arbitrary precision. For example, in the case of the Hopfield network (Hopfield 1982) or the perceptron learning rule (Rosenblatt 1962), the range in which the synapses vary typically increases linearly with the number of stored memories. In all these cases there is no equilibrium distribution, as the distribution of synaptic weights keeps changing. At any time, the distribution is always correlated to all stored memories, and the reasons why memories cannot be retrieved are only related to the interference between different memories. In other words, the memory signal (i.e., the correlation between a synaptic matrix and a particular memory) remains constant with time, whereas the noise (i.e., its fluctuations) increases with the number of memories. In the case of the Hopfield network, the memory capacity, expressed in terms of random uncorrelated patterns that can be retrieved, grows linearly with the number of neurons, which is significantly better than the logarithmic dependence of bounded synapses, for

which the memory signal decays exponentially fast, whereas the noise remains approximately constant.

It is also important to notice that the limitation due to the boundedness of the synaptic weights is not so severe in the case of off-line learning. For example, in the case of the Hopfield model, if one first computes the unbounded synaptic weights according to the Hopfield prescription, and then binarizes them, the number of retrievable patterns still scales linearly with the number of neurons (Sompolinsky 1986). However, this procedure is not possible in a physical system in which there are not temporary repositories which allow to store synaptic weights with unlimited precision.

6.2.3 Solving the Problem of Memory Maintenance

The expression of the upper bound of the number of uncorrelated patterns p suggests a possible way out to elude the memory constraint: if synaptic changes are small or rare enough ($q \ll 1$), then the sliding window in which modifications are remembered can be extended by a factor $1/q$:

$$p < -\frac{\log N}{\log(1-q)} \sim \frac{\log N}{q}.$$

The price to be paid by reducing the size and the number of synaptic modifications is the reduction of the initial memory trace, the one experienced immediately after the memory is stored. In other words, the amount of storable information per memory decreases whenever the synaptic model is modified to reduce q . This is a general property of any realistic synapse: as the memory lifetime is extended, the initial memory trace and hence the amount of information per memory is sacrificed to some extent. In what follows we review a few effective ways of extending memory lifetimes that are related to both the neural representation of memories and the details of the synaptic dynamics. They all affect in different ways the initial memory trace. In most of the cases, different ‘ingredients’ can be combined together to further improve the memory performance.

Extending Memory Lifetimes with Slow Stochastic Learning

The memory lifetime can also be extended by modifying a randomly selected fraction of all synapses that are eligible for a long-term change (as already discussed in the simple example of bistable synapses). Such a mechanism can be easily implemented if each synapse is modified with a given probability. This stochastic selection turns out to work nicely for random uncorrelated patterns even when the number of synaptic states is reduced to the extreme case of two (Amit and Fusi 1992, 1994; Fusi 2002; Tsodyks 1990). In the case of random uncorrelated binary patterns, if the synapses are modified with a probability q , then we have already seen that the memory trace S , defined as the fraction of synapses that are correlated with a particular tracked memory, decays as

$$S(p) = q(1-q)^p = qe^{p \log(1-q)} \sim qe^{-pq},$$

where p is the number of random uncorrelated memories that are stored after the tracked memory. $S(p)$ is proportional to q , as q is the fraction of synapses that are modified for each memory, and it is multiplied by the probability that the synapse is not modified by the other p memories, which is $(1 - q)$ to the power of p . For a small q (slow learning), the initial memory trace $S(0)$ is also small; however, $S(p)$ decays exponentially with a long ‘time constant’ that goes like $1/q$. If q is large (fast learning), new memories are acquired rapidly because a large fraction of synapses are modified, but memories are also quickly overwritten. If $q = 1$ (deterministic synapses), then only one memory can be remembered (this result can be derived by using the exact expression of $S(p)$). When we read out N independent synapses, the fluctuations are of the order of \sqrt{N} . These fluctuations are due both to the stochastic nature of the memories and to the inherently stochastic processes underlying synaptic modifications. The signal-to-noise ratio $\text{SNR}(p)$, defined as the expected number of synapses that remember a particular memory divided by its standard deviation, goes like

$$\text{SNR}(p) = \sqrt{N}qe^{-pq}.$$

If we require that $\text{SNR}(p)$ is larger than some arbitrary value, say 1, then we have approximately

$$p < \frac{1}{q} \log(q\sqrt{N}).$$

The memory capacity is again catastrophically low, as p scales logarithmically with N . If q is small, and in particular, if it is properly chosen for each given size of the network (specifically, it goes to zero as N increases), then we can extend the memory lifetime. The smallest q we can afford is $q \sim 1/\sqrt{N}$ (otherwise the argument of the log becomes smaller than 1 and the logarithm becomes negative), which means $p < 1/q = \sqrt{N}$. For unbounded synapses we would have had $p < \alpha N$, where α is a constant, so there is still a significant reduction in performance, but the improvement over the logarithmic dependence is also considerable. Stochastic learning is the most widely used solution to the memory problem in learning neuromorphic systems. See also Section 6.3 for a dynamical implementation.

Notice that in general, slow stochastic learning is a good solution to the memory problem for any form of learning that is inherently slow. For example, reinforcement learning (Sutton and Barto 1998) is based on a trial-and-error approach that usually requires a large number of iterations before learning converges to the policy that maximizes the cumulative future reward. In this case the learning process is inherently slow, as the number of synapses that should be modified at each iteration is anyway small. In these situations the introduction of synaptic boundaries does not necessarily disrupt the memory performance of the learning system.

Extending Memory Lifetimes with Sparse Memories

If the neural representations of memories are random and sparse, i.e., with a small fraction f of active neurons, then there are situations in which the number of storable memories increases by a factor f^{-2} (Amit and Fusi 1994; Fusi 2002). This is a consequence of the fact that the learning rule can be designed so that the probability that a single synapse is modified scales like

f^2 . In other words, sparsifying the neural representations is equivalent to reducing the number of synapses that are modified on average every time a new memory is stored (see also Section 6.2.3 on slow learning). Of course the amount of information that can be stored for each memory also decreases as the neural representations become sparser (it scales approximately linearly with f). If f goes to zero as the total number of synapses increases, then the memory lifetime can be extended up to $O(N^2)$ (Amit and Fusi 1994; Fusi 2002). However, this scaling is obtained for neural networks in which the number of synapses increases with the number of neurons. In more realistic situations the number of synapses per neuron is large but fixed, and in particular it does not scale with the total number of neurons. In large-scale neural systems with billions of synapses and a limited number of synapses per neuron, sparseness can increase the memory performance but only to some degree. Indeed, if one assumes that f goes to zero with the total number of synapses, then the neural representations would be required to be so sparse that it would be impossible to retrieve them (Ben Dayan Rubin and Fusi 2007). This conclusion is valid only if one assumes that the neural representations of the memories are random and uncorrelated, and that every neuron has to be able to retrieve some information about the stored memory. It is clear that these assumptions are not met in the real brain, and it is likely that there are specific neural architectures that can take advantage of sparseness in a more efficient way.

Extending Memory Lifetimes with Supervision

Consolidated synaptic modifications should be rare events in the learning process. In the unsupervised learning scenario the selection of the synapses that should be changed can be partially operated by the neural activity and partially by the inherent stochastic mechanism acting at the level of each synapse. Any kind of supervision providing some additional feedback about the correctness of the synaptic modification might help to refine the selection of the synapses to be modified. For example, the principle of the perceptron learning algorithm (Rosenblatt 1962), stating that synapses are modified only if the response of the neuron does not match the response desired by the teacher, can further reduce the average number of modified synapses without sacrificing the stored information. See Senn and Fusi (2005a, 2005b) and Fusi and Senn (2006) for a quantitative analysis of these neural systems in the case of bistable synapses. The number of storable random uncorrelated patterns goes from \sqrt{N} to $2\sqrt{N}$. However, the improvement in memory performance is significantly higher if one considers the problem of classification of correlated patterns.

Extending Memory Lifetimes with Synaptic Complexity

So far we have considered only synapses with two stable states. How does the memory performance improve when we increase the number of synaptic states? This is a particularly important question for neuromorphic systems, as it is fundamental to know how much one should invest in the complexity of the circuits implementing each synapse.

There are many ways of increasing the complexity of a synapse. Not all of them lead to significant improvements in memory performance and it is surprising that the simplest bistable synapses are in many situations as good as more complex synapses. We will first consider

what happens in the case in which each synapse has more than two states between the minimal and the maximal efficacies and then we will summarize the theoretical studies for synaptic complexity in the form of meta-plasticity.

Multistate Synapses

If we increase the number of states that each synapse has to traverse to go from the minimum to the maximum weight, we achieve a relatively small improvement. For simplicity, we consider the case in which the synapses are deterministic (the transitions between states are not stochastic). Synapses with stochastic transitions would have the same dependence on the number of states, but both the initial SNR and the memory lifetime p would be scaled by the same factor ($\text{SNR}(0) \rightarrow \text{SNR}(0)q, p \rightarrow p/q$ if q is the transition probability from one state to the neighboring one). If m is the total number of synaptic states, then the memory capacity scales as shown by Amit and Fusi (1994) and Fusi and Abbott (2007):

$$p < m^2 \log(\sqrt{N}/m).$$

The dependence on N is still logarithmic; however, one also gets an extra factor that is proportional to m^2 . The initial SNR is reduced by a factor $1/m$. For small neural systems, this may be a reasonably good solution to the memory problem, but for large systems the capacity is still far from what we can achieve with stochastic learning. Indeed, in order to match the performance of bistable synapses with stochastic transitions, one would need $m \sim N^{1/4}$, which is significantly more difficult than reducing the transition probabilities to $q \sim 1/\sqrt{N}$ in a bistable synapse. Moreover, there is a more serious limitation to be considered. The m^2 factor that extends the memory lifetime is obtained only when synaptic potentiation is perfectly balanced with synaptic depression. Otherwise there is actually no improvement at all with the number of synaptic states, and the performance is again catastrophically poor, comparable to the case of a deterministic bistable synapse (Fusi and Abbott 2007). Not surprisingly, there is accumulating evidence that biological single synaptic contacts are indeed bistable on long timescales (O'Connor et al. 2005; Petersen et al. 1998).

Metaplastic Synapses

Metaplastic changes are defined as long-term modifications that can affect not only the synaptic efficacy, but also the parameters that characterize the synaptic dynamics, and hence the way synapses are modified by future experiences. There is a class of metaplastic synapses that outperform other memory models by having a strong initial SNR that scales with \sqrt{N} and, at the same time, their memory lifetime scales with \sqrt{N} . These models are constructed on the basis of simple considerations: fast bistable synapses that operate on short timescales ($q \sim 1$) are very plastic and hence good for storing new memories, but bad at retaining old ones ($\text{SNR}(0) \sim \sqrt{N}$ and memory lifetime $p \sim 1$). On the other hand, slow synapses, operating on long timescales ($q \ll 1$), are relatively rigid, and they are good for memory preservation, but bad for the storage of new memories ($\text{SNR}(0) \sim 1$ and memory lifetime $p \sim \sqrt{N}$). Fusi et al. (2005) realized that it is possible to design a synapse that operates on multiple timescales and it is characterized by both long memory lifetimes and strong initial memory traces. The proposed synapse has a cascade of states with different levels of plasticity, connected by

metaplastic states. The level of plasticity depends on the past history of synaptic modifications. In particular there are two chains of m synaptic states, corresponding to two strengths of the synaptic weights. When a synapse is modified, the synapse goes into one of the states at the top of the cascade. These states are very plastic ($q = 1$). If the synapse is repeatedly potentiated, then it moves down in the chain of potentiated states, and each state becomes progressively more resistant to depression ($q \sim x^k$, where $x < 1$ and $k = 1, \dots, m$). In other words, the dynamic properties of the synapse change depending on how many consecutive potentiations occur (metaplasticity). The same is valid for depressions. At equilibrium, all the metaplastic states are equally occupied. This means that there are always synapses at the top of the cascade, that are very plastic and ready to store new memories, as well as synapses that are at the bottom of the cascade and preserve consolidated remote memories. For such a synaptic model the memory trace decays as a power law on a wide range, and then it decays exponentially fast:

$$S(p) \sim \frac{1}{m} \frac{1}{1+p} e^{-pq_s},$$

where q_s is the smallest transition probability and in the optimal case it scales as 2^{-m} , and m is the number of states in the chain (Ben Dayan Rubin and Fusi 2007; Fusi et al. 2005). As anticipated, for such a model both the memory lifetime and the initial memory trace scale like \sqrt{N} . The cascade model is supported by the results of several experiments on synaptic plasticity. It has not yet been implemented in neuromorphic hardware.

6.3 Storing Memories in Neuromorphic Hardware

6.3.1 Synaptic Models for Learning

The way the synapses are modified and the memories are stored depends on the task we need to perform, the type of learning we intend to implement (supervised, unsupervised, or reinforcement learning), how the memories are represented (neural code), and how the memories will be retrieved (memory retrieval). In this section, we briefly describe representative synaptic models that have been implemented or that are implementable. Their dynamics are driven by the spikes emitted by the pre-synaptic neurons and by some of the dynamical variables on the post-synaptic side (timing of the emitted spike, depolarization across the membrane or other variables related to the post-synaptic activity). There is no standard model for synaptic plasticity. This is in part due to the fact that the biology of synapses is extremely rich and very heterogeneous. It is also due to the fact that theoretical synaptic models are designed for different purposes. For example, there are models that implement a specific learning algorithm for solving a computational problem (e.g., classification). These are often supported by a theory (e.g., proving the convergence of the algorithm). Other models are only descriptive as they are designed to capture some of the aspects of the observed dynamics of biological synapses (e.g., STDP – spike timing-dependent plasticity). The descriptive models sometimes find an application to a computational problem after they have been designed. Some of the models described below have been designed to both solve a specific problem and to be compatible with biological observations.

Synaptic Models Encoding Mean Firing Rates

It is widely believed and strongly supported by experimental evidence that the mean firing rates of recorded neurons encode much of the information that is relevant for the task executed by the animal. Mean firing rates are usually estimated by counting the spikes emitted in a short time window (50–100 ms), when single neurons are analyzed. However, most neural and synaptic models are now based on what is known as instantaneous firing rate, which is basically the probability of emitting a spike in a very short time interval (1–2 ms). This quantity can be easily read out by integrating the spikes emitted by a population of neurons.

Most learning algorithms are based on the mean firing rates of the pre- and post-synaptic neurons (see, e.g., Dayan and Abbott 2001; Hertz et al. 1991). Many of them are based on Hebb's postulate (Hebb 1949), which states:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

In other words, every time a neuron contributes to the activation of another neuron, its contribution is increased even further. The situation in which this happens can be detected by the synapse by monitoring the pre- and the post-synaptic activities. The synapse is potentiated when the pre- and post-synaptic neurons are repeatedly simultaneously activated (i.e., they fire at high rates).

This postulate was formalized and implemented only one decade later by Rosenblatt (1962), who introduced one of the most powerful and fundamental algorithms for learning in neural network theory: the perceptron. The algorithm is supervised and it can be used to train a neuron (which we will call the ‘output neuron’) to classify input patterns into two distinct categories. After learning, the synaptic weights of the output neurons converge to a configuration for which the output neuron fires at high rate in response to one class of inputs and does not fire in response to the other class. The algorithm can be described as follows: for each input pattern ξ^μ (ξ^μ is a vector, and its components ξ_i^μ are the mean firing rates of specific neurons), the desired output is o^μ ($o^\mu = -1$ for input patterns that should inactivate the neuron and $o^\mu = 1$ for input patterns that should activate the neuron); each synapse w_i is updated as follows:

$$w_i \rightarrow w_i + \xi_i^\mu o^\mu,$$

but only if, with the current synapses, the post-synaptic neuron does not already respond as desired. In other words, the synapse is updated only if the total synaptic current $I^\mu = \sum_{i=1}^N w_i \xi_i^\mu$ is below the activation threshold θ when the desired output $o^\mu = 1$ (and analogously when $I^\mu > \theta$ and $o^\mu = -1$). The synapses are updated for all input patterns repeatedly, until all the conditions on the output are satisfied.

The perceptron algorithm is guaranteed to converge if the patterns are linearly separable (i.e., if there exists a w_{ij} such that $I^\mu > \theta$ for $o = 1$ and $I^\mu < \theta$ for $o = -1$). In other words, if a solution to the classification problem exists, the algorithm is guaranteed to find it in a finite number of iterations. Notice that the synaptic modification is compatible with the Hebb rule if during learning a supervisor (teacher) imposes the desired activity to the post-synaptic

neuron: when $o^\mu = 1$ because of the input of the supervisor, and $\xi_i^\mu = 1$, then the synapse is potentiated, as prescribed by the Hebb postulate.

Other algorithms are based on similar principles as they are based on the covariance between the pre- and post-synaptic activities (Hopfield 1982; Sejnowski 1977). Some of these algorithms have been implemented in neuromorphic hardware with spike-driven synaptic dynamics. They are described below in detail.

Synaptic Models Encoding the Timing of Spikes

The precise timing of the spikes may contain additional information which is not in the mean firing rates. There are synaptic models that have been explicitly designed to classify specific spatiotemporal patterns of spikes (e.g., a spike at time t from neuron 1, followed by a spike at time $t + 5$ ms from neuron 2, and finally, a spike at time $t + 17$ ms from neuron 3). The tempotron algorithm, introduced by Gütig and Sompolinsky (2006), is an example of synaptic dynamics which leads to an efficient classification of these patterns by one integrate-and-fire (IF) neuron. The algorithm is inspired by the perceptron (from which the name ‘tempotron’ is derived) and is designed to separate two classes of spatiotemporal patterns of spikes. In particular, the synapses are modified so that the output neuron fires at least one time in response to one class of input patterns and it never fires within a certain time window in response to another class of input patterns. In the original algorithm, the synapses need to detect the time at which the maximal depolarization V_{\max} is reached and then they are modified depending on whether this depolarization is above or below the threshold θ for emitting a spike. In particular, when $V_{\max} < \theta$ and the output neuron has to spike, the synapses connecting neurons which have been active within a certain time window are potentiated. If $V_{\max} > \theta$ and the output neuron has to remain silent, then the active synapses are depressed. The authors also propose a synaptic dynamics which does not need to detect the maximal depolarization, but it only relies on a certain voltage convolution. This algorithm has not yet been implemented in hardware.

Biologically Inspired Synaptic Models and STDP

In the last two decades, some experiments on biological synapses (Bi and Poo 1998; Levy and Steward 1983; Markram et al. 1997; Sjöström et al. 2001) showed how the precise timing of pre- and post-synaptic spikes can affect the sign and the magnitude of long-term synaptic modifications. Abbott and colleagues coined the acronym STDP to refer to a specific class of synaptic models that are meant to capture several observations of the experiments (Song et al. 2000). In particular all STDP models incorporate the dependence of the synaptic change on the relative timing between pre- and post-synaptic spikes. Specifically, when the pre-synaptic spike precedes a post-synaptic spike within a time window of 10–20 ms, the synapse is potentiated and for the reverse order of occurrence of pre- and post-synaptic spikes, the synapse is depressed. Although these models were initially only descriptive, the community of theoretical neuroscientists invested a great deal of resources in studying the implications of STDP at the network level (Babadi and Abbott 2010; Song and Abbott 2001; Song et al. 2000) and the possible computational role of STDP (Gerstner et al. 1996; Gütig and Sompolinsky 2006; Legenstein et al. 2005). In particular in Legenstein et al. (2005) the

authors investigate what kind of classification problems STDP can solve and what it cannot solve. Interestingly, Gerstner et al. (1996) is probably the only theoretical work which came out before the experimental papers on STDP and predicted the dependence on the relative phase of pre- and post-synaptic spikes.

More recently, new experimental results revealed that STDP is only one of the aspects of the mechanisms for the induction of long-term changes, and that biological plasticity is significantly more complex than what investigators initially believed. These results are summarized in Shouval et al. (2010) and they are beautifully incorporated in a simple model based on calcium dynamics which reproduced most of the known experimental results, including STDP (Graupner and Brunel 2012). The synaptic efficacy $w(t)$ is a dynamical variable which is inherently bistable (the synapse has only two stable, attractive states). The modifications are driven by another variable $c(t)$, which plays the role of post-synaptic calcium concentration (see Figure 6.2). The synaptic efficacy w increases when c is above a threshold θ_+ (light gray dashed line in the figure) and it decreases when w is between θ_- (dark dashed line) and θ_+ . The calcium variable jumps to a higher value every time a pre-synaptic or a post-synaptic spike arrives. It does so instantaneously for post-synaptic spikes, and with a delay $D \sim 10\text{ms}$ for pre-synaptic spikes. It then decays exponentially with a certain time constant, of the order of a few milliseconds. The calcium-induced changes in the synaptic efficacy w depend on the relative times spent by the calcium trace above the potentiation and the depression thresholds. For example, STDP can be easily obtained by choosing the proper set of parameters (see Figure 6.2).

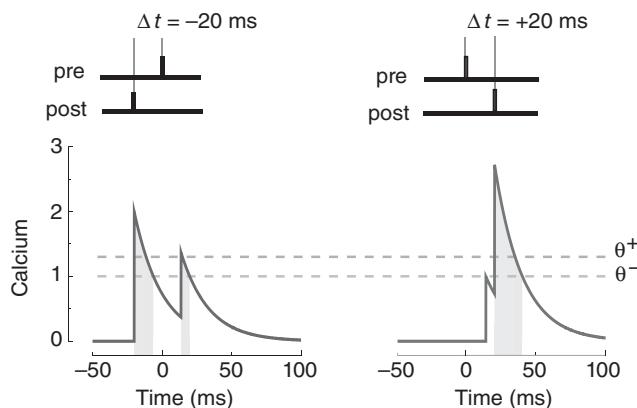


Figure 6.2 The synaptic model proposed by Graupner and Brunel (2012) reproduces STDP. The protocol for inducing long-term modifications is illustrated at the top (left for LTD, right for LTP). When the post-synaptic action potential precedes the pre-synaptic spike, the calcium variable $c(t)$ (indicated with ‘calcium’ on the figure) spends more time between the two dashed lines than above the upper line (see the shaded areas of different gray levels below the curve), eventually inducing long-term depression (left). When the pre-synaptic spike precedes the post-synaptic action potential (right), the calcium variable is above the upper line for a significant fraction of time and it induces a long-term potentiation. Notice that the pre-synaptic spike induces a jump on the post-synaptic calcium concentration with a short delay, whereas the post-synaptic action potential has an instantaneous effect. Adapted from Graupner and Brunel (2012). Reproduced with permission of PNAS

6.3.2 Implementing a Synaptic Model in Neuromorphic Hardware

We now describe a typical procedure for designing a neuromorphic plastic synapse. We start from the theoretical model of the synaptic dynamics and we show how to transform it into a model in which the synapse is multistable (discrete) and hence implementable in hardware. This step requires the introduction of slow stochastic learning (see Section 6.2.3), and hence of a mechanism that transforms an inherently deterministic synapse which varies continuously into a stochastic synapse that is discrete on long timescales. In particular we show that spike-driven synapses can naturally implement stochastic transitions between the stable states, provided that the activity is sufficiently irregular.

We illustrate all these points with a paradigmatic example: we consider the case of auto-associative neural networks like the Hopfield model (Hopfield 1982), in which each memory is represented by a specific pattern of neural activity (typically a pattern of mean firing rates) that is imposed to the network at the time the pattern is memorized. The memory is recalled by reactivating the same or a similar pattern of activity at a later time, letting the network dynamics relax, and observing the stable, self-sustaining pattern of activity that is eventually reached. This pattern represents the stored memory, and it can be recalled by reactivating only a small fraction of the stored memory (pattern completion). In this simple example, the task is to implement an auto-associative content addressable memory; the type of learning we need is ‘supervised’ (in the sense that we know the desired neural output for each given input), the memories are represented by stable patterns of mean firing rates, and they are retrieved by reactivating a certain fraction of the initially activated neurons.

In order to design a synaptic circuit that implements this auto-associative memory, we need first to understand how the synapses should be modified when the patterns are memorized. In this case, the prescription has been found by Hopfield (1982), and it basically says that for each memory we need to add to the synaptic weight the product of the pre- and post-neural activities (under the simplifying assumption that the neurons are either active if they fire at high rate, or inactive if they fire at low rate, and the neural activity is represented by a +1 activity). The rule is basically a covariance rule and it is similar to the perceptron rule (see Section 6.4 for more details).

There are several similar prescriptions for different learning problems and they can be found in any textbook on neural network theory (see, e.g., Hertz et al. 1991 and Dayan and Abbott 2001 for examples of supervised, unsupervised, and reinforcement learning).

From Learning Prescriptions to Neuromorphic Hardware Implementations

Most of the learning prescriptions known in the neural network literature have been designed for simulated neural systems in which the synaptic weights are unbounded and/or continuous valued variables. As a consequence it is difficult to implement them in neuromorphic hardware. As we have seen in the previous section on memory retention, this is not a minor problem. In general, there is no standard procedure to go from the theoretical learning prescription to the neuromorphic circuit. We will describe one successful example which illustrates the typical steps that lead to the design of a synaptic circuit.

Consider again the problem of the auto-associative memory. If we denote by ξ_j the pre-synaptic neural activity, and by ξ_i the post-synaptic neural activity, then the Hopfield

prescription says that every time we store a new memory we need to modify the synaptic weight w_{ij} as follows:

$$w_{ij} \rightarrow w_{ij} + \xi_i \xi_j,$$

where $\xi = \pm 1$ correspond to high and low firing rates (e.g., $\xi = 1$ corresponds to $v = 50$ Hz and $\xi = -1$ corresponds to $v = 5$ Hz). As we will show later in this chapter, it is possible to implement bistable synapses in neuromorphic hardware. In such a case w_{ij} has only two states, and every time we store a new memory we need to decide whether to modify the synapse and in which direction. As explained in the previous section on memory retention, bistable synapses can encode only one memory, unless the transitions between the two states occur stochastically. A simple rule that implements this idea is the following:

- Potentiation: $w_{ij} \rightarrow +1$ with probability q if $\xi_i \xi_j = 1$.
- Depression: $w_{ij} \rightarrow -1$ with probability q if $\xi_i \xi_j = -1$.

In all the other cases the synapse remains unchanged. This new learning prescription, proposed by Amit and Fusi (1992 1994) and Tsodyks (1990), implements an auto-associative memory and it behaves similarly to the Hopfield model. However, the memory capacity goes from $p = \alpha N$ to $p \sim \sqrt{N}$ because of the limitations described in the previous section. How can we now translate this abstract rule into a real synaptic dynamics? This is described in Section 6.3.2.

Spike-Driven Synaptic Dynamics

Once we have a learning prescription for a synapse that is implementable in hardware (e.g., a bistable synapse), we need to define the synaptic dynamics. The details of the dynamics will in general depend on how memories are encoded in the patterns of neural activity. In the case of the example of an auto-associative memory, the memories are assumed to be represented by patterns of mean firing rates. However, most of the hardware implementations are based on spiking neurons and hence we need to consider a synapse that can read out the spikes of pre- and post-synaptic neurons, estimate the mean firing rates, and then encode them in the synaptic weight.

Here we consider one model (Fusi et al. 2000) which is compatible with STDP, but it was designed to implement the rate-based auto-associative network described in Section 6.2 in neuromorphic hardware. Interestingly, noisy pulsed neural activity can naturally implement the stochastic selection mechanism described in the previous section, even when the synaptic dynamics are inherently deterministic. We illustrate the main ideas by presenting a simple paradigmatic example. The same principles are applied to most of the current hardware implementations.

Designing a Stochastic Spike-Driven Bistable Synapse

We consider now the scenario described in the previous section in which the relevant information about a memory is encoded in a pattern of neural firing rates. Each memory is represented

by a pattern of activity in which, on average, half of the neurons fire at high rates and half of the neurons fire at low rates. Inspired by the cortical recordings *in vivo*, we assume that the neural activity is noisy. For example, the pre- and post-synaptic neurons emit spikes according to a Poisson process. If these stochastic processes are statistically independent, each synapse will behave in a different way, even if it experiences the same pre- and post-synaptic average activities. These processes can implement the simple learning rule described in the previous section, in which the synapses make a transition to a different stable state only with some probability q .

Indeed, consider a synapse which is bistable in the absence of spikes, that is, if its internal state variable X is above a threshold, then the synapse is attracted to the maximal value, otherwise it decays to the minimal value (see the mid plots in Figure 6.3). The maximum and the minimum are the only two stable synaptic values. The synaptic efficacy is strong if X is above the threshold and weak if X is below the threshold.

Every pre-synaptic spike (top plots) kicks X up and down depending on whether the post-synaptic depolarization (bottom plots) is above or below a certain threshold, respectively. When a synapse is exposed to stimulation, X can cross the threshold or remain on the same side where it was at the beginning of the stimulation. In the first case the synapse makes a transition to a different state (Figure 6.3, left), while in the second case, no transition occurs (Figure 6.3, right). Whether the transition occurs or not depends on the specific realization of the stochastic processes which control the pre- and the post-synaptic activities. In some cases there are enough closely spaced pre-synaptic spikes that coincide with elevated post-synaptic depolarization and the synapse can make a transition. In other cases the threshold is never crossed and the synapse returns to the initial value. The fraction of cases in which the synapse

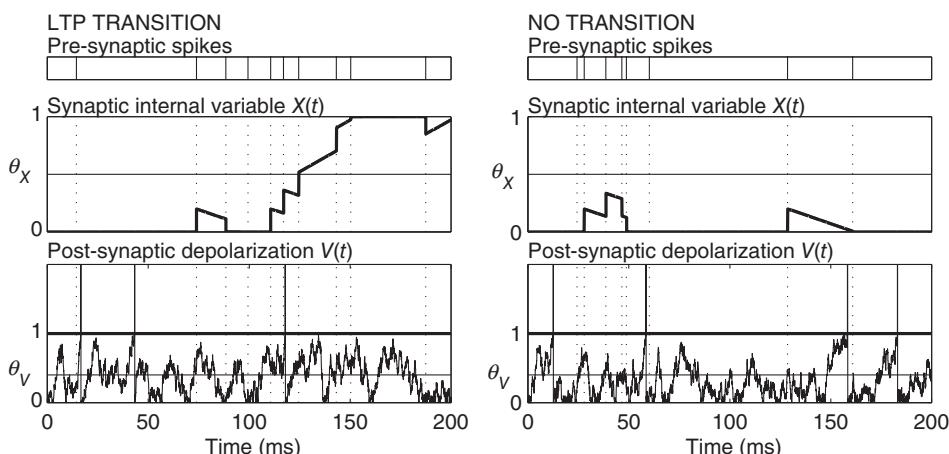


Figure 6.3 Spike-driven synaptic dynamics implementing stochastic selection. From top to bottom: pre-synaptic spikes, synaptic internal state, and post-synaptic depolarization as a function of time. Left: the synapse starts from the minimum (depressed state) but between 100 and 150 ms crosses the synaptic threshold θ_X and ends up in the potentiated state. A transition occurred. Right: the synapse starts from the depressed state, it fluctuates above it, but it never crosses the synaptic threshold, ending in the depressed state. No transition occurred. Notice that in both cases the average firing rates of the pre- and post-synaptic neurons are the same. Adapted from Amit and Fusi (1994). Reproduced with permission of MIT Press

makes a transition determine the probability q which controls the learning rate. Notice that the synaptic dynamics are entirely deterministic and the load of generating stochasticity is transferred outside the synapse. Such a mechanism has been introduced in Fusi et al. (2000) and more recently has been applied to implement the stochastic perceptron (Brader et al. 2007; Fusi 2003). More details of the VLSI synaptic circuit can be found in Section 8.3.2 of Chapter 8. Interestingly, deterministic networks of randomly connected neurons can generate chaotic activity (van Vreeswijk and Sompolinsky 1998) and in particular the proper disorder which is needed to drive the stochastic selection mechanism (Chicca and Fusi 2001).

The Importance of Integrators in Bistable Synapses

Spikes are events that are highly local in time, especially in the electronic implementations (they last 1 ms in biology, significantly less in most VLSI implementations). One of the main difficulties of dealing with these short events is related to the necessity of implementing a stochastic transition between synaptic states. In the relatively long intervals between two successive spikes, no explicit information about the activity of the pre- and post-synaptic neurons is available to the synapse. If the stochastic mechanism is implemented by reading out the instantaneous random fluctuations of the neural activity, then we need a device which constantly updates an estimate of the instantaneous neural activity. Any integrator can bridge the gap between two consecutive spikes or related events, and it can naturally measure quantities like the inter-spike intervals (e.g., in an RC circuit driven by the pre-synaptic spikes, the charge across the capacitor is an estimate of the instantaneous firing rate of the pre-synaptic neuron). The presence of an integrator seems to be a universal requirement for all hardware implementations of a synapse (electronic or biological). It is known that in biological synapses a few coincidences of pre- and post-synaptic spikes are needed to occur within a certain time interval to start the processes that lead to long-term modifications. Analogously, in all VLSI implementations, it seems to be necessary to accumulate enough ‘events’ before consolidating the synaptic modification. In the case of the described synapse, the events are the coincidences of pre-synaptic spikes and high post-synaptic depolarization (for LTP). In the case of pure STDP implementations (Arthur and Boahen 2006), they are the occurrence of pre- and post-synaptic spikes within a certain time window. In all these cases, an integrator (the variable X in the example) is not only important for implementing the stochastic transitions, but also protects the memory from the spontaneous activity, which otherwise would erase all previous synaptic modifications in a few minutes.

Spike-Driven Synaptic Models Implementing the Perceptron

The synaptic model described in Figure 6.3 can be extended to implement the perceptron algorithm described above. The dynamics already described essentially implement the synaptic modification expressed by

$$w_i \rightarrow w_i + \xi_i^\mu o^\mu.$$

The perceptron algorithm is based on the same synaptic modifications, but it is complemented by the condition that the synapse should be updated only when the output neuron does not already respond as desired. In other words, the synapse should not be modified every time

the total synaptic input I^μ is above the activation threshold and the desired output is active, that is, $o^\mu = 1$. An analogous situation occurs when the total synaptic input $I^\mu < \theta$ and $o^\mu = -1$.

This condition can be implemented in synaptic dynamics as suggested in Brader et al. (2007). During training, the post-synaptic neuron receives two inputs, one from the plastic synapses, I^μ , and other from the teacher I_t^μ . The teacher input steers the activity of the output neuron toward the desired value. Hence it is strongly excitatory when the neuron should be active and weak or inhibitory when the neuron should be inactive ($I_t^\mu = \alpha o^\mu$, where α is a constant, and $o^\mu = \pm 1$ depending on the desired output). After training, the plastic synapses should reach a state for which $I^\mu > \theta$ when $o^\mu = 1$ and $I^\mu < \theta$ when $o^\mu = -1$. These conditions should be verified for all input patterns $\mu = 1, \dots, p$. Every time the desired output is achieved for an input pattern, the synapses should not be updated and the next input pattern should be considered. When these conditions are verified for all input patterns, then learning should stop.

The main idea behind the implementation of the perceptron algorithm suggested in Brader et al. (2007) is that it is possible to detect the no-update condition by monitoring the activity of the post-synaptic neuron. Specifically, the synapses should not be updated every time the total synaptic input $I + I_t$ is either very high or very low. Indeed, under these conditions, it is likely that I and I_t already match and that the neuron would respond as desired also in the absence of the teacher input: the maximum of the total synaptic input is reached when both I and I_t are large, and the minimum when both I and I_t are small. In both these cases the synapses should not be modified (e.g., when I_t is large it means that $o = +1$ and the synapses should not be modified whenever $I > \theta$).

The mechanism is implemented by introducing an additional variable, called the calcium variable $c(t)$, by Brader et al. (2007) for its analogies with the calcium concentration in the post-synaptic biological neuron. This variable integrates the post-synaptic spikes, and it represents an instantaneous estimate of the mean firing rate. When $\theta_c < c(t) < \theta_p$, then the synaptic efficacy is pushed toward the potentiated value whenever a pre-synaptic spike arrives and the post-synaptic depolarization is above a threshold V_θ , as in the synapse described in Figure 6.3. When $\theta_c < c(t) < \theta_d$, then the synaptic efficacy is depressed every time a pre-synaptic spike arrives and the post-synaptic depolarization is below V_θ .

Values of $c(t)$ outside these ranges gate the synaptic modification and leave the synapse unchanged. The dynamics can be summarized by saying that for too large or too small values of the post-synaptic mean firing rate, the synapses are not modified. Otherwise they are modified according to the dynamics of Figure 6.3. The transitions between stable states remain stochastic, as in the simpler synaptic model, and the main source of stochasticity remains in the irregularity of the pre- and post-synaptic spike trains. The only difference is the gating operated by the calcium variable, which basically implements the perceptron condition of no-update whenever the output neuron responds correctly to the input.

6.4 Toward Associative Memories in Neuromorphic Hardware

We discussed in the previous sections how memories are stored and then retained in the pattern of synaptic states. We now discuss the process of memory retrieval, which unavoidably involves neural dynamics. Indeed, the synaptic states can be read out only by activating the pre-synaptic neuron and ‘observing’ the effects on the post-synaptic side. This is what is continuously done when the neural dynamics develop in response to a sensory input, or

spontaneously, when driven by internal interactions. The repertoire of dynamic collective states that neural networks can express is rich (see Vogels et al. 2005 for a review of models of neural dynamics) and, depending on the experimental context of reference, the emphasis has been put on global oscillations (see Buzsaki 2006 for a recent overview), possibly including states of synchronous neural firing; chaotic states (see, e.g., Freeman 2003); stationary states of asynchronous firing (see, e.g., Amit and Brunel 1997; Renart et al. 2010). In what follows we will focus on a specific example in which memories are retrieved in the form of attractors of the neural dynamics (stable states of asynchronous firing). Many of the problems that we will discuss are encountered in the other memory retrieval problems.

6.4.1 Memory Retrieval in Attractor Neural Networks

Attractors can take many forms, such as limit cycles for oscillatory regimes, strange attractors for chaotic systems, or point attractors for systems endowed with a Lyapunov function. It is the latter point attractor case that will concern us in what follows. A point attractor for a neural network is a collective dynamic state in which neurons fire asynchronously with a stationary pattern of constant firing rates (up to fluctuations). If neurons are transiently excited by a stimulus, the network state relaxes to the nearest (in a sense to be made precise) attractor state (a point in the abstract network state space, whence the name).

The specific set of attractor states available to the network is dictated by the synaptic configuration. The idea of point attractors most naturally lends itself to the implementation of an associative memory, the attractor states being the *memories* that are retrieved given a hint in the form of an induced initial state of neural activities. In early models of associative memory the goal was to devise appropriate *synaptic prescriptions* ensuring that a prescribed set of patterns of neural activities were attractors of the dynamics.

Though suggestive of *learning*, this was much less, indeed a kind of existence proof for a dynamic scenario that experimental neuroscience strongly seemed to support (more on this later); the deep issue remained, of how to model the supervised, unsupervised or reinforcement-based learning mechanisms driven by the interaction of the brain with its environment, and the generation of related expectation.

The Basic Attractor Network and the Hopfield Model

It is instructive to provide a few details on the working of such models of associative memory (of which the Hopfield model is of course the first successful example), as an easy context to grasp the key interplay between the dynamics of neural states and the structure of the synaptic matrix. In doing this, we will take a step back from the description level of the previous section; we will later consider attractor networks of spiking neurons.

First, in the original Hopfield model, neurons are binary, time-dependent variables. Two basic biological properties are retained in a very simple form: the neuron performs a spatial integration of its inputs, and its activity state can go from low to high if the integrated input exceeds a threshold. In formulae,

$$s_i(t) = \pm 1, \quad i = 1 \dots N \quad s_i(t + \delta t) = \text{sign} \left(\sum_{j \neq i}^{1,N} w_{ij} s_j(t) - \theta \right), \quad (6.2)$$

where N is the number of neurons in the (fully connected) network, w_{ij} is the efficacy of the synapse connecting pre-synaptic neuron j to the post-synaptic neuron i ; the dynamics describe a deterministic change of neural state upon crossing the threshold θ .

The dynamics tend to ‘align’ the neural state with the ‘local field’:

$$h_i = \sum_{j \neq i}^{1,N} w_{ij} s_j(t). \quad (6.3)$$

The dynamics of the network can be described by introducing a quantity, the energy, that depends on the collective state of the network and that decreases at every update. If such a quantity exists, and this is always the case for symmetric synaptic matrices, then the network dynamics can be described as a descent toward the states with lower energy, eventually relaxing at the minima. In the case of the Hopfield model, the energy is

$$E(t) = -\frac{1}{2} \sum_{i \neq j}^{1,N} w_{ij} s_i(t) s_j(t) = \frac{1}{2} \sum_i^{1,N} s_i(t) h_i(t). \quad (6.4)$$

For the deterministic asynchronous dynamics, at each time step at most one neuron (say s_k) changes its state, whereupon, writing

$$E(t) = -s_k(t) h_k(t) - \frac{1}{2} \sum_{\substack{i \neq j \\ i,j \neq k}}^{1,N} w_{ij} s_i(t) s_j(t), \quad (6.5)$$

we get

$$\Delta E(t) \equiv E(t + \delta t) - E(t) = -(s_k(t + \delta t) - s_k(t)) h_k(t), \quad (6.6)$$

so if s_k does not change its state, $\Delta E(t) = 0$; if it does, $\Delta E(t) < 0$. If noise is added to the dynamics, the network does not settle into a fixed point corresponding to a minimum of E , but it will fluctuate in its proximity.

The key step was to devise a prescription for the $\{w_{ij}\}$ such that pre-determined patterns are fixed points of the dynamics. This brings us to the seminal proposal by Hopfield (1982); to approach it let us consider first the case in which we choose just one pattern (one specific configuration of neural states) $\{\xi_i\}$, $i = 1, \dots, N$, with $\xi_i = \pm 1$ chosen at random, and we set the synaptic matrix as

$$w_{ij} = \frac{1}{N} \xi_i \xi_j.$$

Substituting into Eq. (6.4) we see that the configuration $\{s_i = \xi_i\}$, $i = 1, \dots, N$, is a minimum of E , which attains its smallest possible value, $-\frac{1}{2} \frac{N-1}{N}$. The configuration is also easily verified to be a fixed point of the dynamics (Eq. 6.2), since

$$\text{sign} \left(\sum_{j \neq i}^{1,N} \frac{1}{N} \xi_i \xi_j \xi_j \right) = \xi_i.$$

The network state $\{\xi_i\}$ attracts the dynamics even if some (less than half) $s_i \neq \xi_i$, which implements a simple instance of the key ‘error correction’ property of the attractor network: a stored pattern is retrieved from a partial initial information.

The celebrated Hopfield prescription for the synaptic matrix extends the above suggestion to multiple uncorrelated patterns $\{\xi_i^\mu\}, i = 1, \dots, N, \mu = 1, \dots, P$ (i.e., $\xi_i = \pm 1$ independently for all i, μ):

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^P \xi_i^\mu \xi_j^\mu, \quad w_{ii} = 0. \quad (6.7)$$

Of course the question is whether the P patterns $\{\xi_i^\mu\}$ act simultaneously as attractors of the dynamics; as expected, this will depend on the ratio P/N . A simple argument based on a signal-to-noise analysis gives an idea of the relevant factors.

Analogously to the case of one pattern, one can ask about the stability of a given pattern ξ^ν , for which the pattern has to be aligned with its local field ($\xi_i^\nu h_i > 0$):

$$\xi_i^\nu h_i = \xi_i^\nu \sum_{j \neq i}^{1,N} \frac{1}{N} \sum_{\mu=1}^P \xi_i^\mu \xi_j^\mu \xi_j^\nu = \frac{N-1}{N} + \frac{1}{N} \sum_{j \neq i}^{1,N} \sum_{\mu \neq \nu}^{1,P} \xi_i^\nu \xi_i^\mu \xi_j^\nu \xi_j^\mu. \quad (6.8)$$

The first term is a ‘signal’ component of order 1; the second term is zero mean ‘noise.’ It is the sum of $\simeq NP$ uncorrelated terms of order 1, and that can be estimated to be of order $\simeq \sqrt{NP}$. Therefore, the noise term is of order $\sqrt{P/N}$ and is negligible as long as $P \ll N$.

The prescription for w_{ij} can be transformed into a learning rule that is local in time by simply adding to the w_{ij} one pattern at the time. In other words, one can imagine a scenario in which a specific pattern of activity ξ^μ is imposed to the neurons. This pattern represents one of the memories that should be stored, and it induces the following synaptic modifications $w_{ij} \rightarrow w_{ij} + \xi_i^\mu \xi_j^\mu$. When the procedure is iterated for all memories, we get the synaptic matrix prescribed by Hopfield. This form of online learning is what has been discussed in the previous section on memory storage.

It should be noted that the prescription, Eq. (6.7), is by no means a unique solution to the problem; for example, choosing a matrix w_{ij} as a solution of $\sum_j w_{ij} \xi_j^\mu = \lambda \xi_i^\mu$ would do for patterns that are linearly independent. However, this would be a nonlocal prescription, in that it would involve the inverse of the correlation matrix of the patterns ξ . The Hopfield prescription has the virtue of conforming to the general idea put forward by Hebb (1949) that a synapse connecting two neurons which are simultaneously active would get potentiated.

In the retrieval phase, the network states in the Hopfield model populate a landscape with as many valleys as there are stored patterns (plus possibly the spurious ones mentioned above). The multiplicity of minima of E in the high-dimensional network state space suggests the celebrated landscape metaphor, a series of valleys and barriers separating them, in which the system evolves. In the deterministic case the representative point of the network state rolls down the closest valley floor from its initial condition, while noise can allow the system to cross barriers (with a typically exponential dependence of the crossing probabilities on the barrier height for given noise).

One valley is the ‘basin of attraction’ of the point attractor sitting at its floor: the set of initial states that lead ultimately to that attractor under the deterministic dynamics. Suggestive as it is, the usual one-dimensional landscape representation must be taken with caution, for the high dimensionality makes the intuitive grasp weaker; for example, the landscape gets populated with saddles, and the frontiers of the basins of attraction are high-dimensional themselves.

Attractors in Spiking Networks

In going from networks of binary neurons to networks of spiking neurons, most key features of the above picture are preserved, with some relevant differences. The workhorse in models of spiking neurons is the IF neuron: the membrane potential is a linear integrator with a threshold mechanism as a boundary condition on the membrane dynamics (see Chapter 7). The theory of diffusion stochastic processes provides the appropriate tools to analytically describe the IF neuron dynamics with noisy input (Burkitt 2006; Renart et al. 2004). The same formalism also provides the key to the description of a homogeneous recurrent population of IF neurons under the ‘mean field’ approximation: neurons in the population are assumed to share the same statistical properties (mean and variance) of the afferent currents; these are in turn determined by the average neurons’ firing rates and by the average recurrent synaptic efficacy. In an equilibrium state, any given neuron must generate spikes at an average rate which is the same as the other neurons’ one; this establishes a self-consistency condition that (at parity of other single-neuron parameters) determines the values of the average synaptic efficacy that allow for an equilibrium state. The mean field equations implementing such self-consistency condition equate the average firing rate to the single neuron response function, whose arguments (mean and variance of the input current) are now functions of the same average firing rate. For a typical, sigmoid-shaped response function, those equations posses either one stable solution (the firing rate of the only equilibrium state) or three solutions (fixed points), of which the two corresponding to the lowest and highest firing rates are stable.

In the latter case, if the spiking network sits in the lower fixed point and receives a sufficiently strong transient input, it can jump over the unstable state and, after the stimulus terminates, it will relax to the higher fixed point (see Figure 6.4).

The network can host multiple selective sub-populations, each of which is endowed with recurrent synapses that are strong with respect to the cross-sub-population ones. In a typical situation of interest, from a symmetric state of low activity, a stimulus selectively directed to a sub-population can activate an attractor state in which the transiently stimulated sub-population relaxes to a high-rate state, while the others are back to a low-rate state.

In a simulation, the sequence of events that lead to the activation of a selective attractor state would appear as in Figure 6.4. We remark that in the dynamic coupling between sub-populations that lead to the above scenario, a prominent role is played by the interplay between excitatory and inhibitory neurons.

Early examples proving the feasibility of such dynamic processes, generating attractor states in networks of spiking neurons with spike-driven synapses, were provided in simple settings by Del Giudice et al. (2003) and Amit and Mongillo (2003). More complex scenarios have been recently considered (Pannunzi et al. 2012). This study aims at modeling experimental findings from recordings during a visual classification task: a differential modulation of neural activity is observed for visual features defined as relevant or irrelevant for the classification

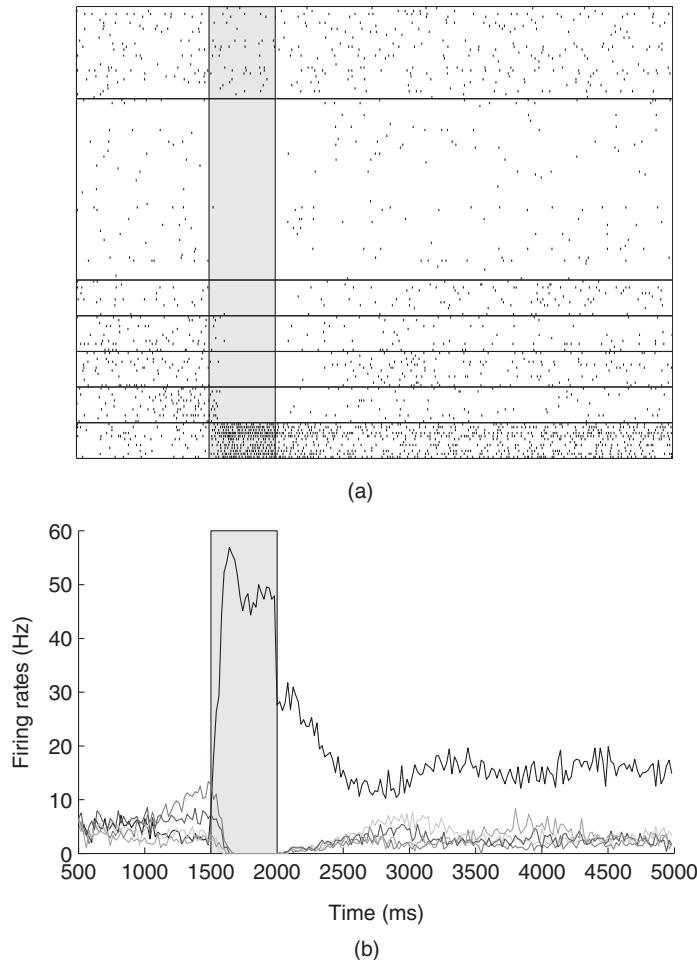


Figure 6.4 Activation of persistent delay activity, selective for a familiar stimulus, in a simulation of synaptically coupled spiking neurons. Potentiated synapses connect excitatory neurons encoding the same stimulus. The raster plot (a) shows the spikes emitted by a subset of cells in the simulated network, grouped in sub-populations with homogeneous functional and structural properties. The bottom five raster strips refer to neurons selective to five uncorrelated stimuli; the raster in the upper strip refers to a subset of inhibitory neurons, and the large middle strip to background excitatory neurons. (b) The emission rates of the selective sub-populations are plotted (the fraction of neurons emitting a spike per unit time). The population activity is such that before the stimulation all the excitatory neurons emit spikes at low rate (global spontaneous activity) almost independently of the functional group they belong to. When one of the selective sub-populations is stimulated (for the time interval indicated by the gray vertical strips), upon releasing the stimulation, that sub-population relaxes to a self-sustaining state of elevated firing activity, an expression of attractor dynamics. Adapted from Del Giudice et al. (2003), Copyright 2003, with permission from Elsevier

task. Under the assumption that such modulation would emerge due to a selective feedback signal from areas implementing the categorical decision, a multimodular model was set up, with mutual interactions between feature-selective populations and decision-coding ones. It was possible to demonstrate learning histories bringing the initially unstructured network to a complex synaptic structure supporting successful performance on the task and also the observed task-dependent modulation of activity in the IT populations.

6.4.2 Issues

The Complex Interactions Between Neural and Synaptic Dynamics

In attractor models based on a *prescription* for the synaptic matrix, the implicit assumption is that the synaptic matrix can be made to emerge through some forms of biologically plausible learning mechanisms. This, however, conceals several subtleties and pitfalls. In fact, it is perhaps surprising that to date few efforts have been devoted to studying how attractor states can dynamically emerge in a neural population as the result of the ongoing neural spiking activity, determined by the flow of external stimuli and the feedback in the population, and biologically relevant spike-driven synaptic changes (Amit and Mongillo 2003; Del Giudice et al. 2003; Pannunzi et al. 2012).

We now consider the familiar scenario of attractor states as memory states in an associative memory, though most remarks have more general validity. A very basic observation is that in a recurrent network the neural and synaptic dynamics are coupled in a very intricate way: synaptic modifications are (slowly) driven by neural activities, which are in turn determined by the pattern of synaptic connections at any given time. An incoming stimulus will both provoke the relaxation of the network to an attractor state, selective for that stimulus (if an appropriate synaptic structure exists at that point), and will also promote synaptic changes depending on the spiking activity it imposes on the network; learning and retrieval are now intertwined processes.

Even well before facing problems of memory capacity, the stability of the relevant neural states are challenged by the above dynamic loop between neural and synaptic dynamics. In fact, once the neural activities repeatedly evoked by a stimulus have left a trace in the synaptic matrix, sufficient to support a selective persistent attractor state, the following spectrum of neural activities emerge: a ‘spontaneous’ state of low firing activity (v_{spon}); a state of high firing corresponding to the response to the external stimulus (v_{stim}); the state of intermediate and selective firing activity, supported by feedback, which persists after the stimulus is gone in the absence of large perturbations (v_{sel}).

A natural request is that in the absence of incoming stimuli, the structured network should be able to keep its memory stable for long times; if spontaneous activity could modify the synapses, or the synaptic matrix could significantly change in the attractor state, any memory state would be doomed to fade away, or the last one to be evoked would dramatically interfere with other memories embedded in the same synaptic matrix at that stage (if the attractor state is left undisturbed for long and the representations of different memories overlap). Hence it is important that the synaptic dynamics is designed so that synaptic changes occur only during stimulation, driven by spikes at average rate v_{stim} (see Section 6.3 on memory storage). But as the synapses go on changing, so do both v_{stim} and v_{sel} , in general by different amounts, and in order for the above constraint to be met all along the learning process, we must have, comparing early and late stages of learning, $v_{\text{sel}}^{\text{late}} < v_{\text{stim}}^{\text{early}}$.

In the regime of slow learning that is of interest, a ‘good’ learning trajectory of the network (i.e., the sequence of synaptic configurations traveled by the network on its way to the onset of selective attractor states) should be a sequence of quasi-equilibrium states. These sequences would have the additional feature that can be studied and controlled by mean field theory approximations. Finally, the parameters of the synaptic model must be tuned to guarantee a balance between LTD and LTP at each learning stage. The introduction of a suitable regulatory mechanism would probably relax this strict requirement.

Finite-Size Effects

Any real network has a finite number of neurons N . Finite-size effects are important and bring about deviations from the predictions of the mean field theory. In particular, for finite N each population has a distribution of emission rates. Let us consider the synapses connecting populations of neurons stimulated by the same stimulus, and therefore supposed to get potentiated: the high- and low-rate tails of the actual frequency distribution corrupt the homogeneity of the pattern of synaptic transition probabilities, such that in the same synaptic group some synapses will have too high LTP probability, while others will be unexpectedly unchanged. Similarly, finite-size effects can provoke unwanted synaptic transitions where they are not expected and harmful (such as a potentiation of synapses involving post-synaptic background neurons, which can become the seed of instability for the spontaneous state).

One ingredient which makes finite-size effects more or less harmful is the character of the ‘synaptic transfer function’, that is the function giving the LTP/LTD transition probabilities as functions of the pre- and post-synaptic emission rates. The sensitivity of this function in the critical region where the rate distributions involved overlap is an important factor in determining how serious finite-size effects are going to be.

6.5 Attractor States in a Neuromorphic Chip

In a way that cannot be articulated in the present context, point attractors can be viewed as the basic, discrete ‘symbols’ of complex and dynamic ‘words’ that constitute the brain’s inner dialog. This is, at best, the identification of the elements for the definition of a computational paradigm, the latter being still far from reach. However, the recognition of the potential computational role of attractors is a good motivation to embody attractor dynamics in neuromorphic VLSI chips, which after all purport ultimately to compute as the brain does.

We sketch here some recent results obtained in the attempt to establish and control attractor dynamics in neuromorphic chips of spiking neurons. We give in sequence an example of *memory retrieval* (i.e., the activation of a pattern of firing rates which is correlated with a previously embedded synaptic structure), and a simple example in which an attractor representation is dynamically constructed in the VLSI network, from the repeated presentation of visual stimuli acquired by a neuromorphic sensor in real time.

6.5.1 Memory Retrieval

We first summarize recent work which, for the first time, demonstrates robust working memory states based on attractor dynamics in a bistable spiking neural network implemented with neuromorphic VLSI hardware (Giulioni et al. 2011). The on-chip network (see Figure 6.5)

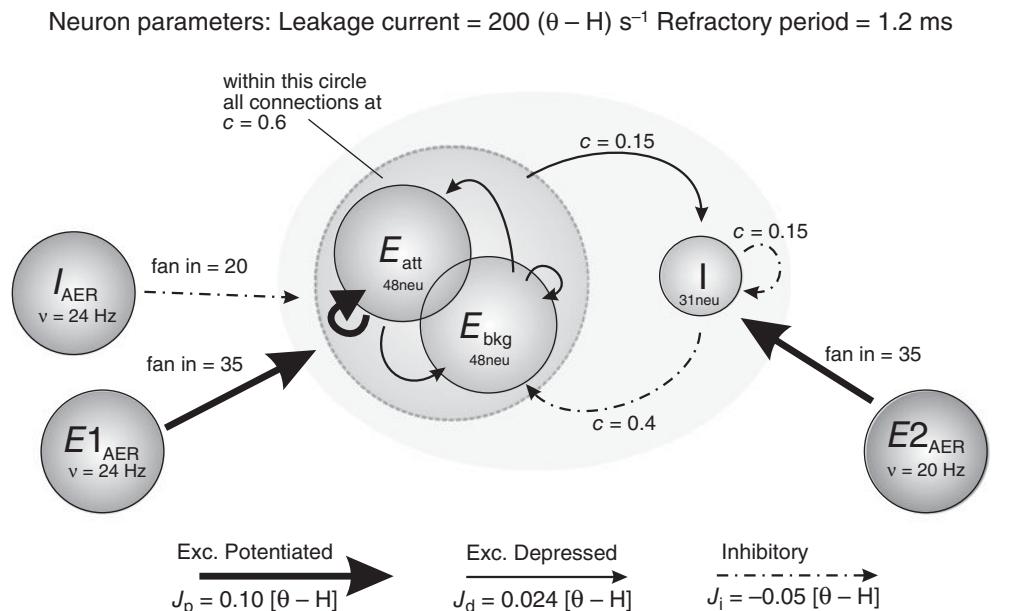


Figure 6.5 Architecture of the network implemented on a VLSI chip with 128 neurons and 16,384 plastic reconfigurable synapses, designed after the model described in Section 6.3.2. Besides the populations E_{att} , E_{bkg} , and I of excitatory and inhibitory neurons, physically residing in the chip, input to the network is provided by further populations emulated on a PC, whose activity is forwarded in real time by a suitable AER communication infrastructure. Adapted from Giulioni et al. (2011). Reproduced with permission of the authors

consists of three interacting populations (two excitatory, one inhibitory) of leaky integrate-and-fire (LIF) neurons. One excitatory population (the E_{att} population in the network of Figure 6.5) has strong synaptic self-excitation, which selectively sustains meta-stable states of ‘high’- and ‘low’-firing activity (the remaining excitatory neurons compose an unselective background). Depending on the overall excitability, transitions of E_{att} activity from the ‘low’ to the ‘high’ state may be evoked by a transient external stimulation (with reference to Figure 6.5, a temporary increase in the external mean firing rate of $E1_{AER}$ and/or I_{AER}); the ‘high’ state retains a ‘working memory’ of a stimulus until well after its release. In a small network such as the one implemented in the chip, state switches can also occur spontaneously due to random activity fluctuations; such cases are interesting but will not be examined here.

The effect of excitatory and inhibitory input transients to E_{att} is illustrated in Figure 6.6. The average firing of E_{att} and E_{bkg} neurons is shown during four successive stimuli, two excitatory and two inhibitory. The first excitatory stimulus is weak; the network reacts in a quasi-linear regime, with a slight activity increase of E_{att} . After the stimulus is released E_{att} returns to the ‘low’ meta-stable state.

The second, stronger excitatory stimulus causes a nonlinear response in the strongly self-coupled E_{att} population and dramatically increases its activity (to a lesser degree E_{bkg} activity is also affected). The recurrent interactions of the network (far more than the external stimulus

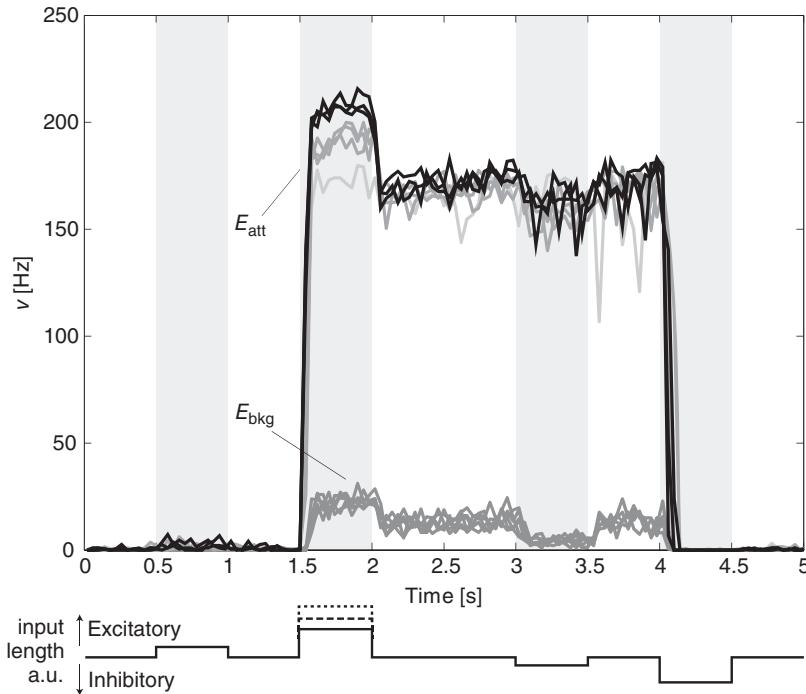


Figure 6.6 Firing rates of E_{att} and E_{bkg} in response to multiple input transients. Excitatory (inhibitory) transients are created by square-pulse increments of $E1_{AER}$ (I_{AER}) activity. The timing of input transients is illustrated beneath the main panel: 0.5 and 1.5 s mark the onset of subthreshold ($v_E = 34$ Hz) and supra-threshold ($v_E = 67, 84$, or 115 Hz) excitatory ‘kicks’, 3 and 4 s that of sub- and supra-threshold inhibitory ‘kicks.’ Subthreshold ‘kicks’ merely modulate activity of the current meta-stable state. Supra-threshold ‘kicks’ additionally trigger a transition to the other meta-stable state. Adapted from Giulioni et al. (2011). Reproduced with permission of the authors

itself) drive the network to a ‘high’ state. After the stimulus is removed the network relaxes to its ‘high’ meta-stable state thereby preserving a ‘working memory’ of the earlier stimulus.

In a similar manner, inhibitory stimuli can induce a return transition to the ‘low’ meta-stable state. In Figure 6.6, a weak inhibitory stimulus was applied at $t = 3$ s and a strong inhibitory one at $t = 4$ s. The former merely shifted temporarily the ‘high’ state, to which E_{att} is attracted back after the end of the stimulus, thus leaving the ‘working memory’ state intact. The latter was sufficiently hefty to destabilize the ‘high’ state, thus forcing the system to return to the ‘low’ state.

The ‘working memory’ state is therefore robust against activity fluctuations and small perturbations, a manifestation of the attractor property.

6.5.2 Learning Visual Stimuli in Real Time

While it is interesting to show that the VLSI network is able to exhibit robust attractor dynamics for a suitable preset synaptic structure, it is by no means obvious that such synaptic structure

can be autonomously generated by the dynamic coupling between the ongoing, stimulus-induced neural activity and the consequent changes in the plastic synapses, which in turn affects the neural response to stimuli (see the discussion in Section 6.4.2). As already noted, this is difficult to achieve for theoretical models, let alone for neuromorphic chips. In this section we demonstrate the learning ability of our on-chip network in a simple, but nontrivial, example (a very preliminary account has been given in Giulioni and Del Giudice 2011). The network architecture is similar to the one depicted in Figure 6.5, but here we use a total of 196 excitatory neurons and 50 inhibitory ones. The network is distributed over two identical chips hosted in the setup shown in Figure 6.7, the inter-chip connectivity being defined by a writable look-up-table in the PCI-AER board. External stimuli come from the dynamic vision sensor retina (Lichtsteiner et al. 2008) which encodes in patterns of AER spikes variations of light intensity in its visual fields. Spikes emitted by the retina are routed toward the chips through the PCI-AER board. The entire system runs in real time and both neural activities and synaptic states are monitored (again through the PCI-AER board). Plastic synapses between excitatory neurons are designed after the model introduced in Fusi et al. (2000) and discussed above.

The visual field of the retina is divided in 196 macropixels, each corresponding to a square of 9×9 retina pixels, and each macropixel is the input source to a single neuron in the on-chip network. From this mapping it is possible to visualize the network firing rates to match the geometric arrangement of the macropixels (see the matrices in Figure 6.8). All synapses are set to be depressed at the beginning of the experiment; given a chosen stimulus structure, learning is expected to selectively potentiate synapses connecting neurons which are co-active for that stimulus. Three visual stimuli were chosen for the experiment illustrated in Figure 6.8, each activating about one-fourth of the neurons in the network (one-fourth of the neurons constitute a background, unselective population). Since the retina is sensitive only to temporal variations of luminosity contrast, a stimulus is presented to the retina as a patterns of black dots flickering on a white background. The portion of visual field not occupied by the stimulus is filled with a sparse noisy background eliciting low activity in the silicon retina.

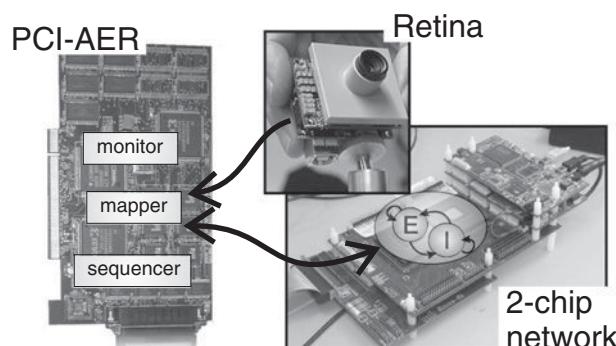


Figure 6.7 Hardware setup for the learning experiments. A PCI-AER board implements the mapping between the neuromorphic retina and the neurons in the recurrent network, in terms of macro-pixels, and manages the asynchronous communication between the retina and the network, which is distributed on two chips. Adapted from Giulioni and Del Giudice (2011). Reproduced with permission of the authors

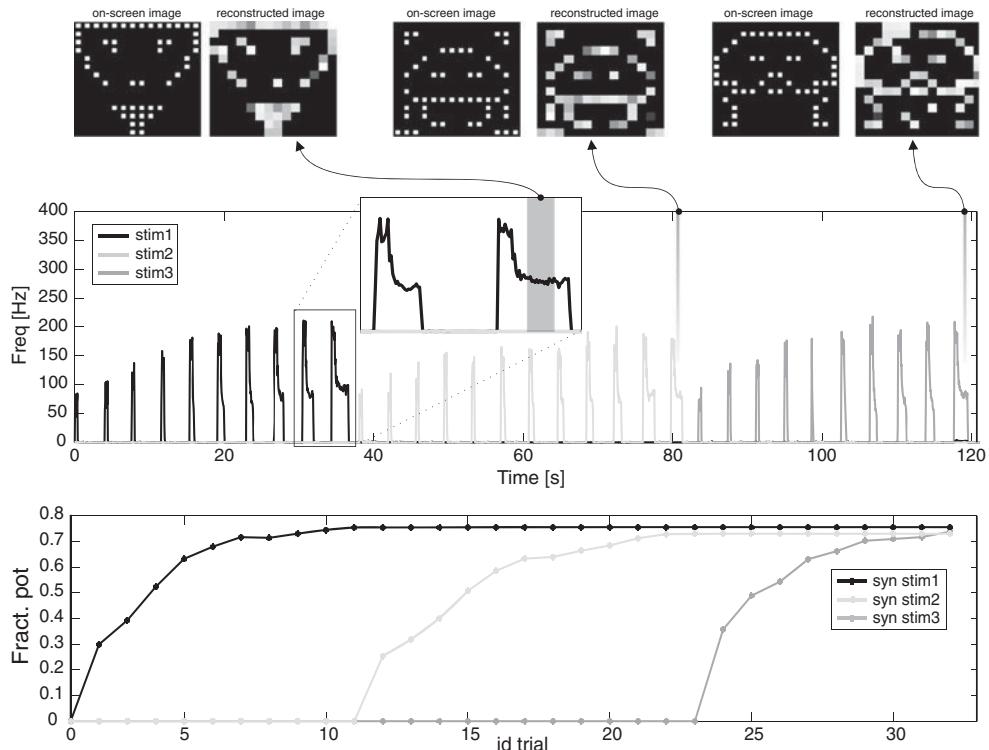


Figure 6.8 Top panel: input visual stimuli shown on the screen (on the left for each pair) and their reconstruction from the network activity after the learning period. Input stimuli are active per 0.4 s per trial. Data for the image reconstruction are recorded during the working memory phase, after the stimulus removal. Each macropixel corresponds to a neuron in the network. Gray levels encode for neurons' firing rates. Central panel: mean firing rates of the three neuronal groups activated by the three (orthogonal) input stimuli. Lower panel: fraction of potentiated synapses, read before every learning trial. Each line reports the time course of the fraction of potentiated recurrent synapses, among those connecting neurons encoding the same input stimulus. Above about 70% potentiation we observe self-sustaining, working memory states in the neural activity. All the synapses connecting neurons encoding different stimuli, and neuron encoding stimuli to background neurons, remain depressed throughout the entire experiment (not shown)

The learning protocol consists of repetitive presentations of the visual stimuli to the retina, as illustrated in Figure 6.8. In the same figure we also report the network response and the evolution of the synapses during learning.

In the absence of visual stimulation, the initial network is in a low-activity state. In the early stages of learning, when the stimulus is removed, the network goes back to this low-activity state.

After repeated presentations, a selective synaptic structure builds up, such that the mature network is able, after the removal of the stimulus, to sustain a noisy but recognizable representation of the learned visual pattern. From Figure 6.8 it is seen that for the mature network

($t > 120$ s) the stimulus elicits a fast response of the selective excitatory sub-population; after stimulus removal the selective activity is kept elevated (because of the strong self-excitation generated by learning). After the high state persists for more than about 2 s, it decays spontaneously back to the low state; indeed, finite-size fluctuations are large in such small systems, and they affect the stability of attractor states in an important way. We stress that, in the chip, inhomogeneities and mismatch cause a wide dispersion of effective synaptic parameters, in the face of which we obtain robust learning histories.

We also confirmed (not shown) that the mature network exhibits the expected pattern completion ability: upon presenting a degraded versions of a learned stimulus, the network is able to retrieve the activity pattern corresponding to the complete stimulus; learning is also robust with respect to the choice of the initial condition for the synaptic matrix.

6.6 Discussion

The theory covered by this chapter shows that there is a fundamental problem of scalability in neuromorphic devices with memory and online learning. The problem, related to the boundedness of the variables that represent the synaptic weights, is important as learning neuromorphic chips are mostly occupied by plastic synapses. Unfortunately the problem is only partially solved, with the consequence that current neuromorphic devices suffer from major limitations in their memory capacity. In the most efficient models that are available, the number p of storable dense memories scales with the square root of the number N of synapses, which is sufficient for performing a variety of different and interesting tasks, but it is far from what is achievable in neural networks with off-line learning, where p can scale linearly with N . In large-scale neural systems, which are now becoming implementable and further described in Chapter 16, neuromorphic devices with online learning are greatly disadvantaged in terms of memory capacity.

Some of the limitations of neuromorphic systems derive from the simplicity of the switch-like mechanisms used to implement plastic synapses. Not only does this simplicity contrast with the complexity of biological synapses, it also makes large-scale neuromorphic devices highly inefficient. Properly designed complex synapses can greatly increase the amount of information stored per memory (Fusi et al. 2005) and the number of storable memories (Benna and Fusi 2013), closing the gap between neural systems with online and off-line learning. However, these types of synapses are difficult to implement in hardware, and silicon area occupied by complex plastic synapses can be so large that it might be more efficient to use a larger number of simple synapses. It is unclear whether silicon synapses can actually benefit from complexity as much as their biological counterparts.

An alternative approach is to utilize simple bistable synapses in more structured memory systems. Memories can be distributed across multiple brain regions, each one characterized by a limited memory capacity (see, e.g., Roxin and Fusi 2013). This strategy can lead to improved memory performance, however it usually requires a complex neural machinery to transfer and organize information. More theoretical studies will be needed to determine the optimal implementation strategies for large-scale neural systems and most likely every problem will have a different solution.

A final challenge is to deal with heterogeneities (mismatch in electrical circuits). Biological systems can tolerate large variations in the parameters that characterize the dynamics of their

elements. They may even take advantage of the diversity of their components. Neuromorphic analog devices have also to deal with heterogeneities which are not under control in the design of the chip. At the moment the electronic mismatch can be highly disruptive and it greatly limits the performance of neuromorphic devices. It is not inconceivable that in the future the diversity generated by mismatch will actually be harnessed to perform computation. This will require new theoretical studies for understanding the computational role of heterogeneity (see, e.g., Shamir and Sompolinsky 2006).

Whatever the form is of future models of plastic synapses which are amenable to microelectronic implementation, it will also determine constraints on the relevant theoretical description of the ensuing neural dynamics and its impact on our ability to control neuromorphic chips. We gave in Section 6.4 of this Chapter an example of the way mean field theory can be used as a conceptual guide to build a procedure for a model-driven ‘system identification’ approach to find relevant regions in the chip’s parameter space. The case shown, however, was still simple in that the system comprised few homogeneous populations. On the other hand, a theoretical compass to navigate the parameter space will be even more needed for neural systems with complex structured synaptic connectivity, unless the synaptic and neural elements will be endowed with self-tuning mechanisms, which will require additional complexity. The control of these new neural systems will probably require a new theory which will be able to deal with complex and highly heterogeneous neural networks.

References

- Amit D. 1989. *Modeling Brain Function*. Cambridge University Press.
- Amit DJ and Brunel N. 1997. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb. Cortex* **7**(3), 237–252.
- Amit DJ and Fusi S. 1992. Constraints on learning in dynamic synapses. *Netw.: Comput. Neural Syst.* **3**(4), 443–464.
- Amit DJ and Fusi S. 1994. Learning in neural networks with material synapses. *Neural Comput.* **6**(5), 957–982.
- Amit DJ and Mongillo G. 2003. Spike-driven synaptic dynamics generating working memory states. *Neural Comput.* **15**(3), 565–596.
- Anthony M and Bartlett PL. 1999. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.
- Arthur JV and Boahen K. 2006. Learning in silicon: timing is everything. In: *Advances in Neural Information Processing Systems 18 (NIPS)* (eds Weiss Y, Schölkopf B, and Platt J). MIT Press Cambridge, MA. pp. 75–82.
- Babadi B and Abbott LF. 2010. Intrinsic stability of temporally shifted spike-timing dependent plasticity. *PLoS Comput. Biol.* **6**(11), e1000961.
- Ben Dayan Rubin DD, and Fusi S. 2007. Long memory lifetimes require complex synapses and limited sparseness. *Front. Comput. Neurosci.* **1**(7), 1–14.
- Benna M and Fusi S. 2013. Long term memory ... now longer than ever. *Cosyne 2013 II-1*, 102–103.
- Bi GQ and Poo MM. 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* **18**(24), 10464–10472.
- Block HD. 1962. The perceptron: a model for brain functioning. I. *Rev. Mod. Phys.* **34**(1), 123–135.
- Brader JM, Senn W, and Fusi S. 2007. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* **19**(11), 2881–2912.
- Burkitt A. 2006. A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biol. Cybern.* **95**(1), 1–19.
- Buzsaki G. 2006. *Rhythms of the Brain*. Oxford University Press.
- Chicca E and Fusi S. 2001. Stochastic synaptic plasticity in deterministic aVLSI networks of spiking neurons. In: *Proceedings of the World Congress on Neuroinformatics* (ed. Rattay F). ARGESIM/ASIM Verlag, Vienna. pp. 468–477.
- Dayan P and Abbott LF. 2001. *Theoretical Neuroscience*. MIT Press.

- Del Giudice P, Fusi S, and Mattia M. 2003. Modelling the formation of working memory with networks of integrate-and-fire neurons connected by plastic synapses. *J. Physiol. Paris* **97**(4–6), 659–681.
- Freeman WJ. 2003. Evidence from human scalp EEG of global chaotic itinerancy. *Chaos* **13**(3), 1067–1077.
- Fusi S. 2002. Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. *Biol. Cybern.* **87**(5–6), 459–470.
- Fusi S. 2003. Spike-driven synaptic plasticity for learning correlated patterns of mean firing rates. *Rev. Neurosci.* **14**(1–2), 73–84.
- Fusi S and Abbott LF. 2007. Limits on the memory storage capacity of bounded synapses. *Nat. Neurosci.* **10**(4), 485–493.
- Fusi S and Senn W. 2006. Eluding oblivion with smart stochastic selection of synaptic updates. *Chaos* **16**(2), 026112.
- Fusi S, Annunziato M, Badoni D, Salamon A, and Amit DJ. 2000. Spike-driven synaptic plasticity: theory, simulation, VLSI implementation. *Neural Comput.* **12**(10), 2227–2258.
- Fusi S, Drew PJ, and Abbott LF. 2005. Cascade models of synaptically stored memories. *Neuron* **45**(4), 599–611.
- Gerstner W, Kempter R, van Hemmen JL, and Wagner H. 1996. A neuronal learning rule for sub-millisecond temporal coding. *Nature* **383**(6595), 76–81.
- Giulioni M and Del Giudice P. 2011. A distributed VLSI attractor network learning visual stimuli in real time and performing perceptual decisions. In: *Frontiers in Artificial Intelligence and Applications – Proceedings of WIRN 2011* (eds Apolloni B, Bassis S, Esposito A, and Morabito CF), vol. 234. IOS press, pp. 344–352.
- Giulioni M, Camilleri P, Mattia M, Dante V, Braun J, and Del Giudice P. 2011. Robust working memory in an asynchronously spiking neural network realized with neuromorphic VLSI. *Front. Neuromorphic Eng.* **5**, 149.
- Graupner M and Brunel N. 2012. Calcium-based synaptic plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. *Proc. Natl. Acad. Sci. USA* **109**(10), 3991–3996.
- Gütig R and Sompolinsky H. 2006. The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* **9**(3), 420–428.
- Hebb DO. 1949. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York.
- Hertz J, Krogh A, and Palmer RG. 1991. *Introduction to the Theory of Neural Computation*. Addison Wesley.
- Hinton G and Sejnowski TJ. 1999. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, Cambridge, MA.
- Hopfield JJ. 1982. Neural networks and physical systems with emergent selective computational abilities. *Proc. Natl. Acad. Sci. USA* **79**(8), 2554–2558.
- Legenstein R, Naeger C, and Maass W. 2005. What can a neuron learn with spike-timing-dependent plasticity? *Neural Comput.* **17**(11), 2337–2382.
- Levy WB and Steward O. 1983. Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience* **8**(4), 791–797.
- Lichtsteiner P, Posch C, and Delbrück T. 2008. A 128×128 120 dB 15 us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* **43**(2), 566–576.
- Markram H, Lubke J, Frotscher M, and Sakmann B. 1997. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* **275**(5297), 213–215.
- Minsky ML and Papert SA. 1969. *Perceptrons*. MIT Press, Cambridge. Expanded edition 1988.
- Nägerl UV, Köstinger G, Anderson JC, Martin KAC, and Bonhoeffer T. 2007. Protracted synaptogenesis after activity-dependent spinogenesis in hippocampal neuron. *J. Neurosci.* **27**(30), 8149–8156.
- O'Connor DH, Wittenberg GM, and Wang SSH. 2005. Graded bidirectional synaptic plasticity is composed of switch-like unitary events. *Proc. Natl. Acad. Sci. USA* **102**(27), 9679–9684.
- Pannunzio M, Gigante G, Mattia M, Deco G, Fusi S, and Del Giudice P. 2012. Learning selective top-down control enhances performance in a visual categorization task. *J. Neurophys.* **108**(11), 3124–3137.
- Parisi G. 1986. A memory which forgets. *J. Phys. A.* **19**, L617.
- Petersen CCH, Malenka RC, Nicoll RA, and Hopfield JJ. 1998. All-or-none potentiation at CA3-CA1 synapses. *Proc. Natl. Acad. Sci. USA* **95**(8), 4732–4737.
- Renart A, Brunel N, and Wang X. 2004. Mean-field theory of irregularly spiking neuronal populations and working memory in recurrent cortical networks. In: *Computational Neuroscience: A Comprehensive Approach* (eds Maass W and Bishop CM). Chapman and Hall/CRC, pp. 431–490.
- Renart A, de la Rocha J, Bartho P, Hollender L, Parga N, Reyes A, and Harris KD. 2010. The asynchronous state in cortical circuits. *Science* **327**(5965), 587–590.

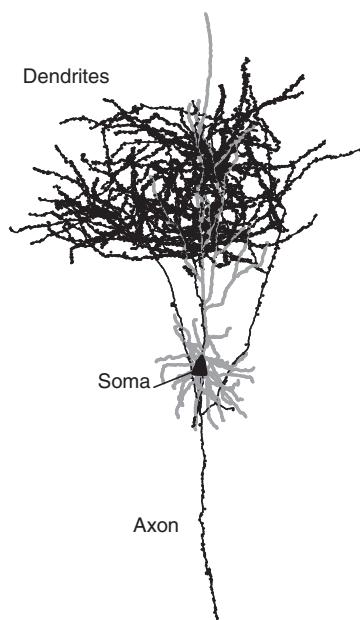
- Rosenblatt F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**(6), 386–408. Reprinted in: *Neurocomputing: Foundations of Research* (eds Anderson JA and Rosenfeld E).
- Rosenblatt F. 1962. *Principles of Neurodynamics*. Spartan Books, New York.
- Roxin A and Fusi S. 2013. Efficient partitioning of memory systems and its importance for memory consolidation. *PLoS Comput. Biol.* **9**(7), e1003146.
- Sejnowski TJ. 1977. Storing covariance with nonlinearly interacting neurons. *J. Math. Biol.* **4**, 303–321.
- Senn W and Fusi S. 2005a. Convergence of stochastic learning in perceptrons with binary synapses. *Phys. Rev. E. Stat Nonlin. Soft Matter Phys.* **71**(6 Pt 1), 061907.
- Senn W and Fusi S. 2005b. Learning only when necessary: better memories of correlated patterns in networks with bounded synapses. *Neural Comput.* **17**(10), 2106–2138.
- Shamir M and Sompolinsky H. 2006. Implications of neuronal diversity on population coding. *Neural Comput.* **18**(8), 1951–1986.
- Shouval HZ, Wang SS, and Wittenberg GM. 2010. Spike timing dependent plasticity: a consequence of more fundamental learning rules. *Front. Comput. Neurosci.* **4**(19), 1–13.
- Sjöström PJ, Turrigiano GG, and Nelson SB. 2001. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* **32**(6), 1149–1164.
- Sompolinsky H. 1986. Neural networks with non-linear synapses and static noise. *Phys. Rev. A* **34**, 2571.
- Song S and Abbott LF. 2001. Cortical development and remapping through spike timing-dependent plasticity. *Neuron* **32**(2), 339–350.
- Song S, Miller KD, and Abbott LF. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* **3**(9), 919–926.
- Sutton RS and Barto AG. 1998. *Reinforcement Learning: An Introduction*. MIT Press.
- Tsodyks M. 1990. Associative memory in neural networks with binary synapses. *Mod. Phys. Lett. B* **4**, 713–716.
- van Vreeswijk C and Sompolinsky H. 1998. Chaotic balanced state in a model of cortical circuits. *Neural Comput.* **10**(6), 1321–1371.
- Vogels TP, Rajan K, and Abbott LF. 2005. Neural network dynamics. *Annu. Rev. Neurosci.* **28**(1), 357–376.

Part II

Building Neuromorphic Systems

7

Silicon Neurons



The event-based communication circuits of Chapter 2, the sensors described in Chapters 3 and 4, and the learning rules described in Chapter 6 all involved the use of spiking neurons. There are many different models of spiking neurons and many ways of implementing them using electronic circuits. In this chapter we present a representative subset of such neuromorphic circuits, showing implementations of both simple models and biologically faithful ones, following different circuit design approaches.

The figure shows a reconstruction of a cat Layer 6 neuron with the dendrites and cell body in gray and the axon and boutons in black. Reproduced with permission of Nuno da Costa, John Anderson, and Kevan Martin.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

7.1 Introduction

Biological neurons are the primary components of networks in the brain. The neuronal membranes of these cells have active conductances which control the flow of ionic current between the various ionic reversal potentials and the membrane voltage on the membrane capacitance. These active conductances are usually sensitive to either the trans-membrane potential or the concentration of a specific ion. If these concentrations or voltages change by a large enough amount, a voltage pulse is generated at the axon hillock, a specialized region of the soma that connects to the axon. This pulse, called a ‘spike’ or ‘action potential,’ is propagated along the cell’s axon and activates synaptic connections with other neurons as it reaches the pre-synaptic terminals. Neuromorphic silicon neurons (SiNs) (Indiveri et al. 2011) are complementary metal oxide semiconductor (CMOS), very large-scale integration (VLSI) circuits that emulate the electro-physiological properties of biological neurons. The emulation uses the same organizational technique as traditional digital numerical simulations of biological neurons. Depending on the complexity and degree of emulation, different types of neuron circuits can be implemented, ranging from implementations of simple integrate-and-fire (I&F) models to sophisticated circuits that implement the functionality of full conductance-based models.

Spike-based computational models of neurons can be very useful for both investigating the role of spike timing in the computational neuroscience field and implementing event-driven computing systems in the neuromorphic engineering field. Several spike-based neural network simulators have been developed within this context, and much research has focused on software tools and strategies for simulating spiking neural networks (Brette et al. 2007). Digital tools and simulators are convenient and practical for exploring the quantitative behavior of neural networks. However they are not ideal for implementing real-time behaving systems or detailed large-scale simulations of neural systems. Even with the most recent super-computing systems or custom digital systems that exploit parallel graphical processing units (GPUs) or field-programmable gate arrays (FPGAs), it is not possible to obtain real-time performance when running simulations large enough to accommodate multiple cortical areas, yet detailed enough to include distinct cellular properties. Conversely, hardware emulations of neural systems that use SiNs operate in real time, and the speed of the network is independent of the number of neurons or their coupling. SiNs offer a medium in which neuronal networks can be emulated *directly* in hardware rather than simply simulated on a general purpose computer. They are much more energy efficient than simulations executed on general purpose computers, or custom digital integrated circuits, so they are suitable for real-time large-scale neural emulations (Schemmel et al. 2008; Silver et al. 2007). On the other hand, SiN circuits provide only a qualitative approximation to the exact performance of digitally simulated neurons, so they are not ideal for detailed quantitative investigations. Where SiN circuits provide a tangible advantage is in the investigation of questions concerning the strict real-time interaction of the system with its environment (Indiveri et al. 2009; Le Masson et al. 2002; Mitra et al. 2009; Vogelstein et al. 2007). Within this context, the circuits designed to implement these real-time, low-power neuromorphic systems can be used to build hardware, brain-inspired, computational solutions for practical applications.

SiN circuits represent one of the main building blocks for implementing neuromorphic systems. Although in the original definition, the term neuromorphic was restricted to the set of analog VLSI circuits that operate using the same physics of computation used by the nervous system (e.g., silicon neuron circuits that exploit the physics of the silicon medium to directly reproduce the bio-physics of nervous cells), the definition has now been broadened to include analog/digital hardware implementations of neural processing systems, as well as

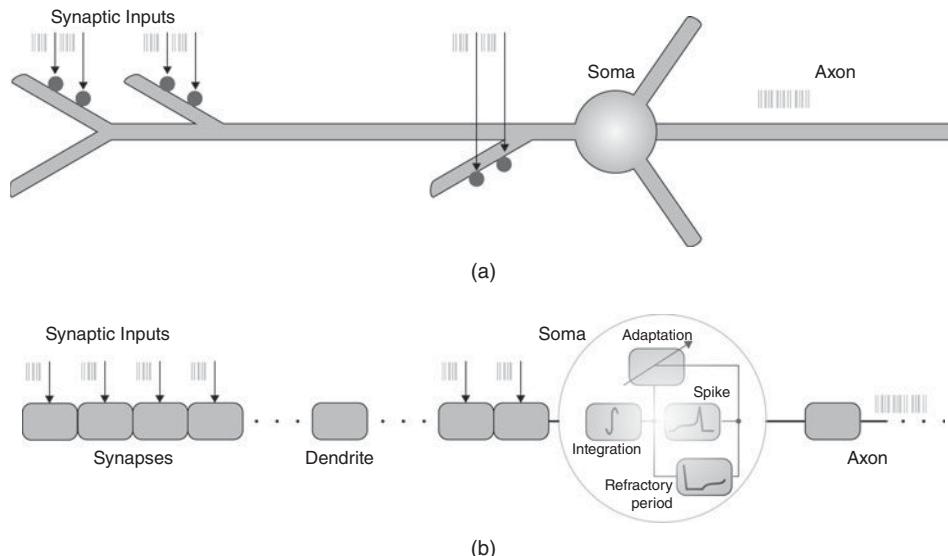


Figure 7.1 (a) Neural model diagram, with main computational elements highlighted; (b) corresponding SiN circuit blocks

spike-based sensory processing systems. Many different types of SiNs have been proposed that model the properties of real neurons at many different levels: from complex bio-physical models that emulate ion channel dynamics and detailed dendritic or axonal morphologies to basic integrate-and-fire (I&F) circuits. Depending on the application domain of interest, SiN circuits can be more or less complex, with large arrays of neurons all integrated on the same chip, or single neurons implemented on a single chip, or with some elements of the neuron distributed across multiple chips.

From the functional point of view, silicon neurons can all be described as circuits that have one or more *synapse* blocks, responsible for receiving spikes from other neurons, integrating them over time and converting them into currents, as well as a *soma* block, responsible for the spatiotemporal integration of the input signals and generation of the output analog action potentials and/or digital spike events. In addition both synapse and soma blocks can be interfaced to circuits that model the neuron's spatial structure and implement the signal processing that takes place in dendritic trees and axons, respectively (see Figure 7.1).

The Synapse

The synapse circuits of a SiN can carry out linear and nonlinear integration of the input spikes, with elaborate temporal dynamics, and short- and long-term plasticity mechanisms. The temporal integration circuits of silicon synapses, as well as those responsible for converting voltage spikes into excitatory or inhibitory post-synaptic currents (EPSCs or IPSCs, respectively), share many common elements with those used in the soma integration and adaptation blocks (see Chapter 8).

The Soma

The soma block of a SiN can be further subdivided into several functional blocks that reflect the computational properties of the theoretical models they implement. Typically SiNs comprise

one or more of the following stages: a (linear or nonlinear) temporal integration block, a spike generation block, a refractory period block, and a spike-frequency or spiking threshold adaptation block. Each of these functional sub-blocks can be implemented using different circuit design techniques and styles. Depending on which functional blocks are used, and how they are combined, the resulting SiN can implement a wide range of neuron models, from simple linear-threshold units to complex multicompartmental models.

The Dendrites and Axon

The dendrites and axon circuit blocks can be used to implement the cable equation for modeling signal propagation along passive neuronal fibers (Koch 1999). These circuits allow the design of multicompartment neuron models that take into account neuron spatial structure. We will describe examples of such circuits in Section 7.2.5.

7.2 Silicon Neuron Circuit Blocks

7.2.1 Conductance Dynamics

Temporal Integration

It has been shown that an efficient way of modeling neuron conductance dynamics and synaptic transmission mechanisms is by using simple first-order differential equations of the type $\tau \dot{y} = -y + x$, where y represents an output voltage or current, and x the input driving force (Destexhe et al. 1998). For example, this equation governs the behavior of all passive ionic channels found in nerve membranes.

The standard *voltage-mode* circuit used to model variable conductances in neuromorphic VLSI devices is the *follower-integrator* circuit. The follower-integrator comprises a transconductance amplifier configured in negative feedback mode with its output node connected to a capacitor. When used in the weak-inversion domain, this follower-integrator behaves as a first-order low-pass filter (LPF) with a tunable conductance (Liu et al. 2002). The silicon neuron circuit proposed by Mahowald and Douglas (1991) uses a series of follower-integrators to model sodium, potassium, and other proteic channel dynamics. This circuit will be briefly described in Section 7.3.1. An alternative approach is to use *current-mode* designs. In this domain, an efficient strategy for implementing the first-order differential equations is to use *log-domain* circuits (Tomazou et al. 1990). For example, Drakakis et al. (1997) showed how a log-domain circuit denoted as the ‘Bernoulli-Cell’ can efficiently implement synaptic and conductance dynamics in silicon neuron designs. This circuit has been fully characterized in Drakakis et al. (1997) and has been used to implement Hodgkin–Huxley VLSI models of neurons (Tomazou et al. 1998). An analogous circuit is the ‘Tau-Cell’ circuit, shown in Figure 7.2a. This circuit, first proposed in Edwards and Cauwenberghs (2000) as a BiCMOS log-domain filter, was fully characterized in van Schaik and Jin (2003) as a subthreshold log-domain circuit and used in Yu and Cauwenberghs (2010) to implement conductance-based synapses. This circuit has also been recently used for implementing both the Mihalas–Niebur neuron model (van Schaik et al. 2010b) and the Izhikevich neuron model (Rangan et al. 2010; van Schaik et al. 2010a).

An alternative log-domain circuit that acts as a basic LPF is shown in Figure 7.2b. Based on the filter design originally proposed by Frey (1993), this circuit acts as a voltage pulse integrator which integrates input spikes arriving at the V_{in} node to produce an output current I_{syn} with exponential rise and decay temporal dynamics. The LPF circuit time constant can be set by adjusting the V_τ bias, and the maximum current amplitude (e.g., corresponding to

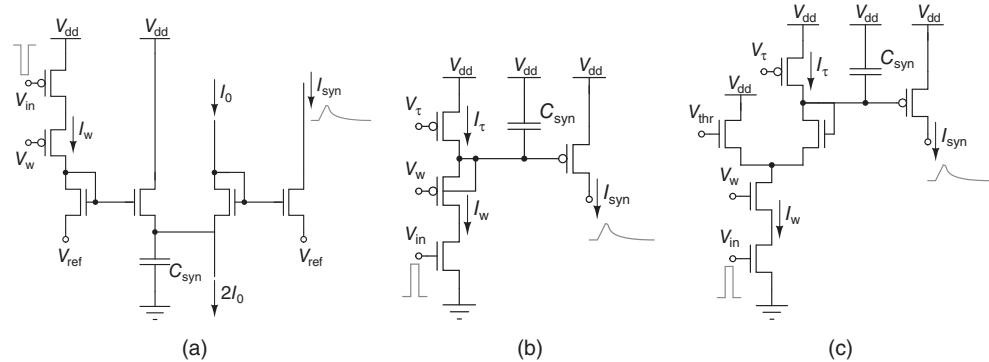


Figure 7.2 Log-domain integrator circuits. (a) Subthreshold log-domain circuit used to implement a first-order low-pass filter (LPF); (b) ‘Tau-Cell’ circuit: alternative first-order LPF design; (c) ‘DPI’ circuit: nonlinear current-mode LPF circuit

synaptic efficacy) depends on both V_τ and V_w . A detailed analysis of this circuit is presented in Bartolozzi and Indiveri (2007). A nonlinear circuit, analogous to the LPF pulse integrator, is the differential pair integrator (DPI) shown in Figure 7.2c. This circuit integrates voltage pulses, following a current-mode approach, but rather than using a single pFET to generate the appropriate I_w current, via the translinear principle (Gilbert 1975), it uses a differential pair in negative feedback configuration. This allows the circuit to achieve LPF functionality with tunable dynamic conductances: input voltage pulses are integrated to produce an output current that has maximum amplitude set by V_w , V_τ , and V_{thr} .

In all circuits of Figure 7.2 the V_w bias (the synaptic weight) can be set by local circuits to implement learning and plasticity (Fusi et al. 2000; Mitra et al. 2009). However, the DPI offers an extra degree of freedom via the V_{thr} bias. This parameter can be used to implement additional adaptation and plasticity schemes, such as intrinsic or homeostatic plasticity (Bartolozzi and Indiveri 2009).

7.2.2 Spike-Event Generation

Biophysically realistic implementations of neurons produce analog waveforms that are continuous and smooth in time, even for the generation of action potentials. In many other neuron models however, such as the I&F model, the action potential is a discontinuous and discrete event which is generated whenever a set threshold is crossed.

One of the original circuits proposed for implementing I&F neuron models in VLSI is the *Axon-Hillock* circuit (Mead 1989). Figure 7.3a shows a schematic diagram of this circuit. The amplifier block A is typically implemented using two inverters in series. Input currents I_{in} are integrated on the membrane input capacitance C_{mem} , and the analog voltage V_{mem} increases linearly until it reaches the amplifier switching threshold (see Figure 7.3b). At this point V_{out} quickly changes from 0 to V_{dd} , switching on the reset transistor and activating positive feedback through the capacitor divider implemented by C_{mem} and the feedback capacitor C_{fb} . The change in the membrane voltage induced by this positive feedback is

$$\Delta V_{mem} = \frac{C_{fb}}{C_{mem} + C_{fb}}.$$

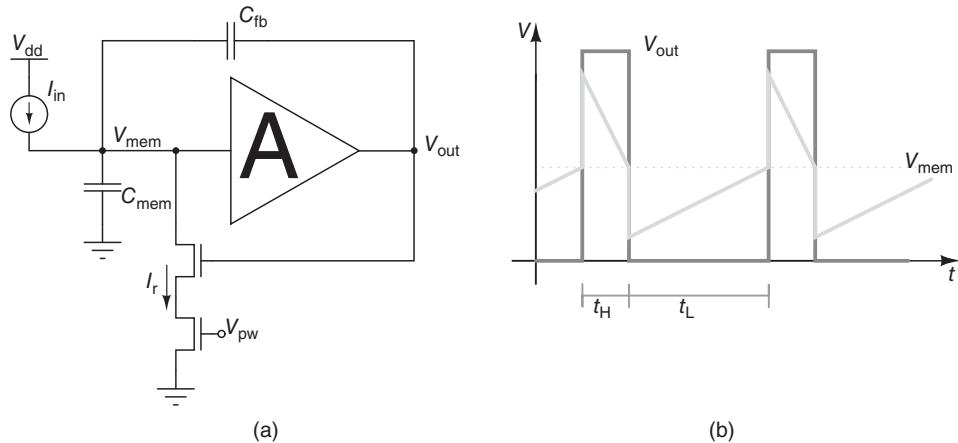


Figure 7.3 Axon-hillock circuit. (a) Schematic diagram; (b) membrane voltage and output voltage traces over time

If the reset current set by V_{pw} is larger than the input current, the membrane capacitor is discharged, until it reaches the amplifier's switching threshold again. At this point V_{out} swings back to 0, and the membrane voltage undergoes the same change in the opposite direction (see V_{mem} trace of Figure 7.3b).

The inter-spike interval t_L is inversely proportional to the input current, while the pulse duration period t_H depends on both the input and reset currents:

$$t_L = \frac{C_{fb}}{I_{in}} V_{dd}; \quad t_H = \frac{C_{fb}}{I_r - I_{in}} V_{dd}.$$

A comprehensive description of the circuit operation is presented in Mead (1989). One of the main advantages of this self-resetting neuron, which arises out of the positive feedback mechanism, is its excellent matching properties: mismatch is mostly dependent on the matching properties of capacitors rather than any of its transistors. In addition, positive feedback allows the circuit to be robust to noise and small fluctuations around the spiking threshold.

An alternative method for implementing positive feedback in these types of circuits is to copy the current produced by the amplifier to generate a spike back onto the integrating input node. This method was first proposed in a vision sensor implementing a model of the Octopus retina (Culurciello et al. 2003). An analogous silicon neuron circuit which implements this type of positive feedback is shown in Figure 7.4; as the membrane voltage V_{mem} approaches the switching threshold of the inverter M_{A1-3} , the current flowing through it is copied and sourced back into the V_{mem} node through the current mirror $M_{A1-M_{A4}}$. The pFET M_{A5} is used to delay the positive feedback effect and avoid oscillations, while the nFET M_{R3} is used to reset the neuron. In addition to eliminating fluctuations around the spiking threshold, this mechanism drastically reduces power consumption: the period spent by the first inverter in the conducting state (when all nFETs and pFETs are active) is extremely short (e.g., on timescales of nano- to microseconds), even if the membrane potential changes very slowly (e.g., on timescales of milliseconds to seconds).

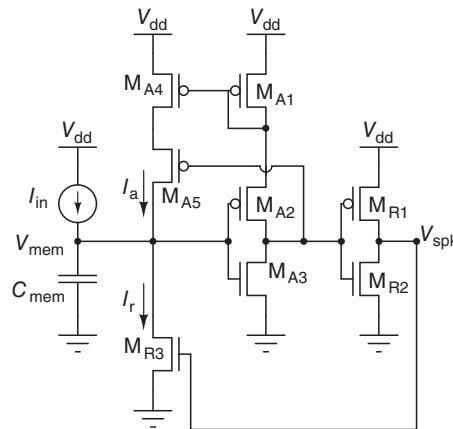


Figure 7.4 The Octopus retina neuron. The input current is generated by a photodetector, while the spike generator uses positive current feedback to accelerate input and output transitions to minimize short-circuit currents during spike production. The membrane capacitance (C_{mem}) is disconnected from the input of the spike generator to further accelerate transition and to reduce power during reset

7.2.3 Spiking Thresholds and Refractory Periods

The Axon-Hillock circuit produces a spike event when the membrane voltage crosses a voltage threshold that depends on the geometry of the transistors and on the VLSI process characteristics. In order to have better control over the spiking threshold, it is possible to use a five-transistor amplifier, as shown in Figure 7.5a. This neuron circuit, originally proposed in van Schaik (2001), comprises circuits for both setting explicit spiking thresholds and implementing an explicit refractory period. Figure 7.5b depicts the various stages that the membrane potential V_{mem} is involved in, during the generation of an action potential.

The capacitance C_{mem} of this circuit models the membrane of a biological neuron, while the membrane leakage current is controlled by the gate voltage V_{lk} , of an nFET. In the

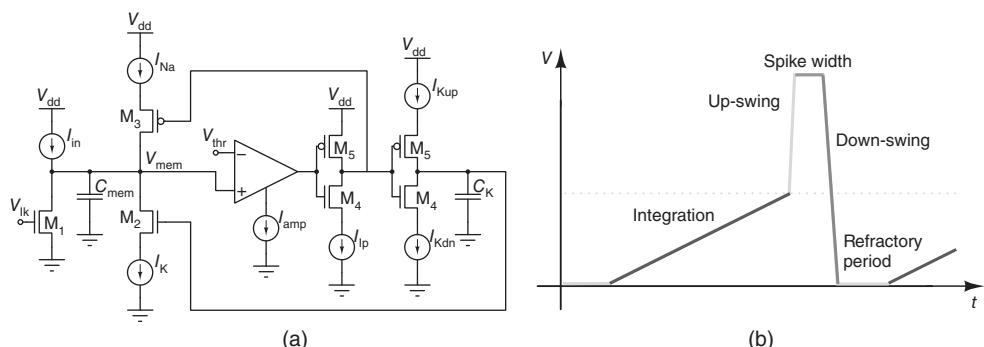


Figure 7.5 Voltage-amplifier I&F neuron. (a) Schematic diagram; (b) membrane voltage trace over time

absence of any input the membrane voltage will be drawn to its resting potential (ground, in this case) by this leakage current. Excitatory inputs (e.g., modeled by I_{in}) add charge to the membrane capacitance, whereas inhibitory inputs (not shown) remove charge from the membrane capacitance. If an excitatory current larger than the leakage current is injected, the membrane potential V_{mem} will increase from its resting potential. The voltage V_{mem} is compared with the threshold voltage V_{thr} , using a basic transconductance amplifier (Liu et al. 2002). If V_{mem} exceeds V_{thr} , an action potential is generated. The generation of the action potential happens in a similar way as in the biological neuron, where an increased sodium conductance creates the upswing of the spike, and a delayed increase of the potassium conductance creates the downswing. In the circuit this is modeled as follows: as V_{mem} rises above V_{thr} , the output voltage of the comparator will rise to the positive power supply. The output of the following inverter will thus go low, thereby allowing the *sodium current* I_{Na} to pull up the membrane potential. At the same time however, a second inverter will allow the capacitance C_K to be charged at a speed which can be controlled by the current I_{Kup} . As soon as the voltage on C_K is high enough to allow conduction of the nFET M2, the *potassium current* I_K will be able to discharge the membrane capacitance. Two different potassium channel currents govern the opening and closing of the potassium channels: the current I_{Kup} controls the spike width, as the delay between the opening of the sodium channels and the opening of the potassium channels is inversely proportional to I_{Kup} . If V_{mem} now drops below V_{thr} , the output of the first inverter will become high, cutting off the current I_{Na} . Furthermore, the second inverter will then allow C_K to be discharged by the current I_{Kdn} . If I_{Kdn} is small, the voltage on C_K will decrease only slowly, and, as long as this voltage stays high enough to allow I_K to discharge the membrane, it will be impossible to stimulate the neuron for I_{in} values smaller than I_K . Therefore I_{Kdn} controls the refractory period of the neuron.

The principles used by this design to control spiking thresholds explicitly have been used in analogous SiN implementations (Indiveri 2000; Indiveri et al. 2001; Liu et al. 2001). Similarly, the principle of using starved inverters (inverting amplifier circuits in which the current is limited by an appropriately biased MOSFET in series) and capacitors to implement refractory periods is used also in the DPI neuron described in Section 7.3.2.

An additional advantage that this circuit has over the Axon-Hillock circuit is power consumption: The Axon-Hillock circuit non-inverting amplifier, comprising two inverters in series, dissipates large amounts of power for slowly varying input signals, as the first inverter spends a significant amount of time in its fully conductive state (with both nFET and pFET conducting) when its input voltage V_{mem} slowly crosses the switching threshold. The issue of power consumption has been addressed also in other SiN designs and will be discussed in Section 7.3.1.

7.2.4 Spike-Frequency Adaptation and Adaptive Thresholds

Spike-frequency adaptation is a mechanism observed in a wide variety of neural systems. It acts to gradually reduce the firing rate of a neuron in response to constant input stimulation. This mechanism may play an important role in neural information processing and can be used to reduce power consumption and bandwidth usage in VLSI systems comprising networks of silicon neurons.

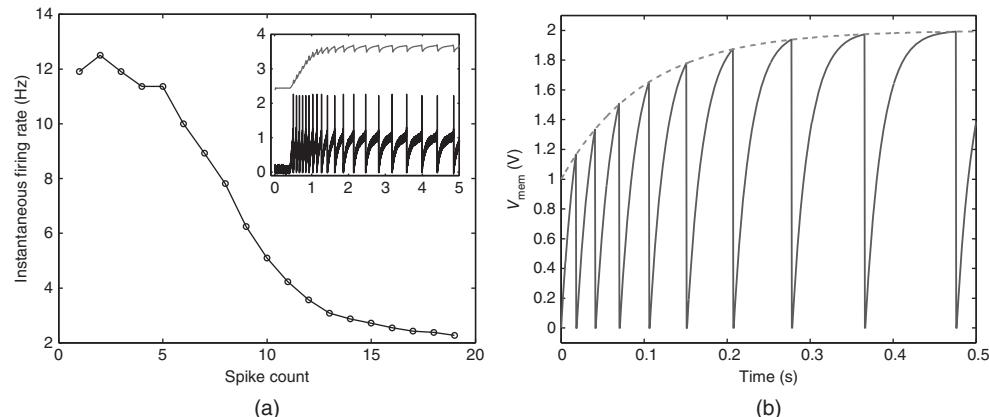


Figure 7.6 Spike-frequency adaptation is a SiN. (a) Negative slow ionic current mechanism: the plot shows the instantaneous firing rate as a function of spike count. The inset shows how the individual spikes increase their inter-spike interval with time. © 2003 IEEE. Reprinted, with permission, from Indiveri (2007). (b) Adaptive threshold mechanism: the neuron’s spiking threshold increases with every spike, therefore increasing the inter-spike interval with time

There are several processes that can produce spike-frequency adaptation. Here we will focus on the neuron’s intrinsic mechanism which produces slow ionic currents with each action potential that are subtracted from the input. This ‘negative feedback mechanism’ has been modeled differently in a number of SiNs.

The most direct way of implementing spike-frequency adaptation in a SiN is to integrate the spikes produced by the SiN itself (e.g., using one of the filtering strategies described in Section 7.2.1) and subtract the resulting current from the membrane capacitance. This would model the effect of calcium-dependent after-hyperpolarization potassium currents present in real neurons (Connors et al. 1982) and introduce a second slow variable in the model, in addition to the membrane potential variable, that could be effectively used to produce different spiking behaviors. Figure 7.6a shows measurements from a SiN with this mechanism implemented (Indiveri 2007), in response to a constant input current.

Spike-frequency adaptation and other more complex spiking behaviors can also be modeled by implementing models with adaptive thresholds, as in the Mihalas–Niebur neuron model (Mihalas and Niebur 2009). In this model a simple first-order equation is used to update the neuron’s spiking threshold voltage based on the membrane voltage variable itself: for high membrane voltage values, the spiking threshold adapts upwards, increasing the time between spikes for a constant input. Low membrane voltage values, on the other hand, result in a decrease of the spiking threshold voltage. The speed at which the threshold adapts in this model is dependent on several parameters. Tuning of these parameters determines the type of spiking behavior that is exhibited by the SiN. Figure 7.6b shows spike-frequency adaptation using an adaptive threshold. Here each time the neuron spikes the threshold voltage resets to a higher value so that the membrane voltage must grow by a larger amount and hence the time between spikes increases.

Examples of two-state variable SiNs that use either of these mechanisms will be presented in Section 7.3.

7.2.5 Axons and Dendritic Trees

Recent experimental evidence suggests that individual dendritic branches can be considered as independent computational units. A single neuron can act as a multilayer computational network, with the individually separated dendritic branches allowing for parallel processing of different sets of inputs on different branches before their outputs are combined (Mel 1994).

Early VLSI dendritic systems included the passive cable circuit model of the dendrite specifically by implementing the dendritic resistance using switched-capacitor circuits (Northmore and Elias 1998; Rasche and Douglas 2001). Other groups have subsequently incorporated some active channels into VLSI dendritic compartments (e.g., Arthur and Boahen 2004). Farquhar et al. (2004) applied their transistor channel approach for building ion channels to building active dendrite models in which ions were able to diffuse both across the membrane and axially along the length of the dendrite (Hasler et al. 2007). They used subthreshold MOSFETs to implement the conductances seen along and across the membranes and model diffusion as the macro-transport method of ion flow. The resulting single-dimensional circuit is analogous to the diffuser circuit described in Hynna and Boahen (2006), but allows the conductances of each of the MOSFETs to be individually programmed to obtain the desired neuron properties. In Hasler et al. (2007) and Nease et al. (2012) they showed how an aVLSI active dendrite model could produce action potentials down a cable of uniform diameter with active channels every five segments.

Wang and Liu (2010) constructed an aVLSI neuron with a reconfigurable dendritic architecture which includes both individual computational units and a different spatial filtering circuit (see Figure 7.7). Using this VLSI prototype, they demonstrate that the response of a dendritic component can be described as a nonlinear sigmoidal function of both input temporal synchrony and spatial clustering. This response function means that linear or nonlinear computation in a neuron can be evoked depending on the input spatiotemporal pattern (Wang and Liu 2013). They have also extended the work to a 2D array of neurons with 3×32 dendritic compartments and demonstrated how dendritic nonlinearities can contribute to neuronal computation such as reducing the accumulation of mismatch-induced response variations from the different compartments when combined at the soma and reducing the timing jitter of the output spikes (Wang and Liu 2011).

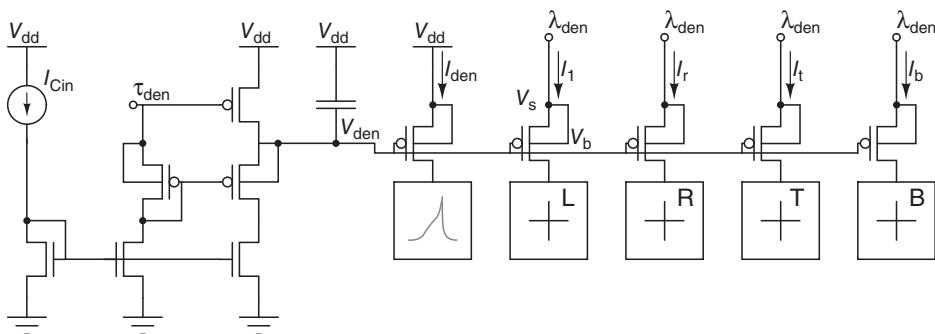


Figure 7.7 Dendritic membrane circuit and cable circuit connecting the compartments. The ‘+’ blocks indicate neighboring compartments. The block to which I_{den} flows into is similar to the circuit in Figure 7.10a

7.2.6 Additional Useful Building Blocks

Digi-MOS

Circuits that operate like an MOS transistor but with a digitally-adjustable size factor W/L are very useful in neuromorphic SiN circuits, for providing a weighted current or for calibration to compensate for mismatch. Figure 7.8 shows a possible circuit implementation based on MOS ladder structures (Linares-Barranco et al. 2003). In this example, the 5-bit control word $b_4 b_3 b_2 b_1 b_0$ is used to set the effective $(W/L)_{\text{eff}}$ ratio. As the currents flowing through each sub-branch differ significantly, this circuit does not have unique time constants. Furthermore, small currents flowing through the lower bit branches will settle to a steady state value very slowly, therefore such a circuit should not be switched at high speeds, but should rather be used to provide DC biasing currents. This circuit has been used in spatial contrast retinas (Costas-Santos et al. 2007) and charge-packet I&F neurons within event-based convolution chips (Serrano-Gotarredona et al. 2006, 2008) for mismatch calibration.

Alternative design schemes, using the same principle but different arrangements of the transistors can be used for applications in which high-speed switching is required (Leñero Bardallo et al. 2010).

Very Low Current Mirrors

Typically, the smallest currents that can be processed in conventional circuits are limited by the MOS ‘off subthreshold current,’ which is the current a MOS transistor conducts when its gate-to-source voltage is zero. However, MOS devices can operate well below this limit (Linares-Barranco and Serrano-Gotarredona 2003). To make MOS transistors operate properly below this limit, one needs to bias them with negative gate-to-source voltages, as illustrated in the current mirror circuit of Figure 7.8c. Transistors M1–M2 form the current mirror. Current I_{in} is assumed to be very small (pico- or femto-amperes), well below the ‘off subthreshold current.’ Consequently, transistors M1 and M2 require a negative gate-to-source voltage. By using the voltage-level shifter M4–M5 and connecting the source voltage of M1–M2 to $V_{nsh} = 0.4$ V, the mirror can be biased with negative gate-to-source voltages. This technique has been used to build very low frequency compact oscillators and filters (Linares-Barranco and Serrano-Gotarredona 2003) or to perform in-pixel direct photo current manipulations in spatial contrast retinas (Costas-Santos et al. 2007).

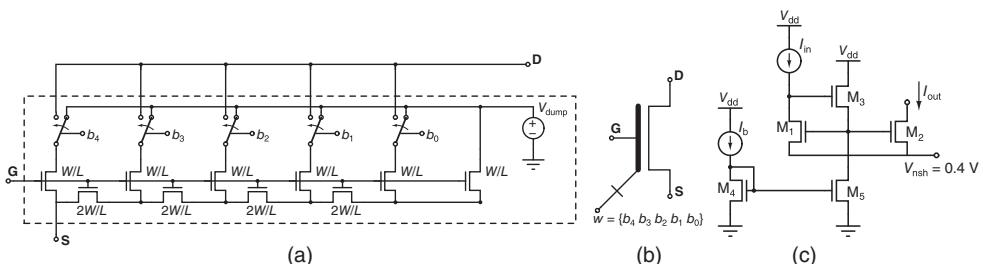


Figure 7.8 (a) Digi-MOS: MOS transistor with digitally adjustable size factor $(W/L)_{\text{eff}}$. Example 5-bit implementation using MOS ladder techniques. (b) Digi-MOS circuit symbol; (c) very low current mirror: circuit with negative gate-to-source voltage biasing for copying very low currents

7.3 Silicon Neuron Implementations

We will now make use of the circuits and techniques introduced in Section 7.2 to describe silicon neuron implementations. We organized the various circuit solutions in the following way: subthreshold biophysically realistic models; compact I&F circuits for event-based systems; generalized I&F neuron circuits; above threshold, accelerated-time, switched-capacitor, and digital designs.

7.3.1 Subthreshold Biophysically Realistic Models

The types of SiN designs described in this section exploit the biophysical equivalence between the transport of ions in biological channels and charge carriers in transistor channels. In the classical conductance-based SiN implementation described in Mahowald and Douglas (1991), the authors modeled ionic conductances using five-transistor transconductance amplifier circuits (Liu et al. 2002). In Farquhar and Hasler (2005), the authors showed how it is possible to model ionic channels using single transistors, operated in the subthreshold domain. By using two-transistor circuits, Hynna and Boahen (2007) showed how it is possible to implement complex thermodynamic models of gating variables (see also Section 7.2.1). By using multiple instances of the gating variable circuit of Figure 7.10a, it is possible, for example, to build biophysically faithful models of thalamic relay neurons.

The Conductance-Based Neuron

This circuit represents perhaps the first conductance-based silicon neuron. It was originally proposed by Mahowald and Douglas (1991), and it is composed of connected compartments, each of which is populated by modular subcircuits that emulate particular ionic conductances. The dynamics of these types of circuits is qualitatively similar to the Hodgkin–Huxley mechanism without implementing their specific equations. An example of such a silicon neuron circuit is shown in Figure 7.9.

In this circuit the membrane capacitance C_{mem} is connected to a transconductance amplifier that implements a conductance term, whose magnitude is modulated by the bias voltage G_{leak} . This passive leak conductance couples the membrane potential to the potential of the ions to which the membrane is permeable (E_{leak}). Similar strategies are used to implement the active sodium and potassium conductance circuits. In these cases, transconductance amplifiers configured as simple first-order LPFs are used to emulate the kinetics of the conductances. A current mirror is used to subtract the sodium activation and inactivation variables (I_{Naon} and I_{Naoff}), rather than multiplying them as in the Hodgkin–Huxley formalism. Additional current mirrors half-wave rectify the sodium and potassium conductance signals, so that they are never negative.

Several other conductance modules have been implemented using these principles: for example, there are modules for the persistent sodium current, various calcium currents, calcium-dependent potassium current, potassium A-current, nonspecific leak current, and an exogenous (electrode) current source. The prototypical circuits can be modified in various ways to emulate the particular properties of a desired ion conductance (e.g., its reversal potential). For example, some conductances are sensitive to calcium concentration rather than membrane voltage and require a separate voltage variable representing free calcium concentration.

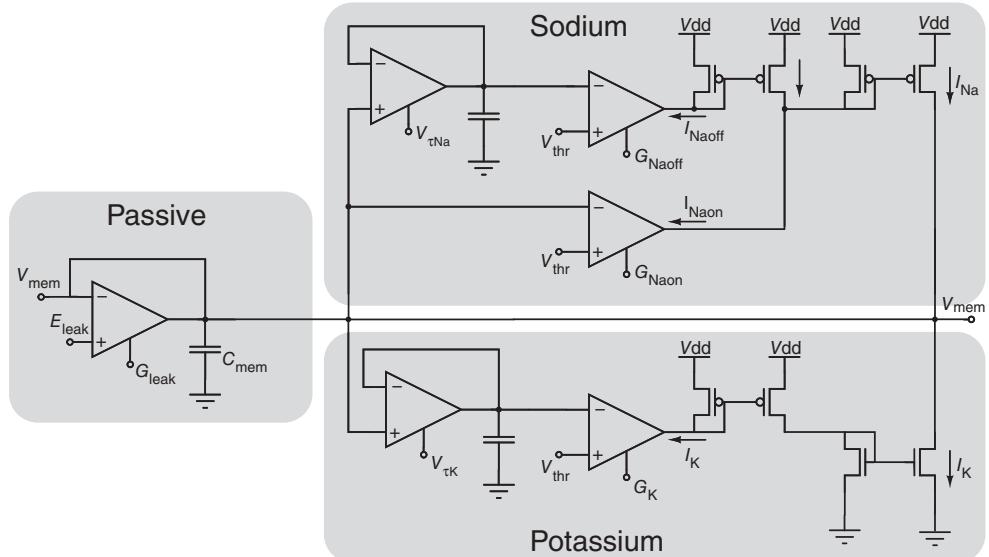


Figure 7.9 A conductance-based silicon neuron. The ‘passive’ module implements a conductance term that models the passive leak behavior of a neuron: in absence of stimulation the membrane potential V_{mem} leaks to E_{leak} following first-order LPF dynamics. The ‘sodium’ module implements the sodium activation and inactivation circuits that reproduce the sodium conductance dynamics observed in real neurons. The ‘potassium’ module implements the circuits that reproduce the potassium conductance dynamics. The bias voltages G_{leak} , $V_{\tau\text{Na}}$, and $V_{\tau\text{K}}$ determine the neuron’s dynamic properties, while G_{Naon} , G_{Naoff} , G_{K} , and V_{thr} are used to set the silicon neuron’s action potential characteristics

The Thalamic Relay Neuron

Many of the membrane channels that shape the output activity of a neuron exhibit dynamics that can be represented by state changes of a series of voltage-dependent *gating particles*, which must be open for the channel to conduct. The state transitions of these particles can be understood within the context of thermodynamic equivalent models (Destexhe and Huguenard 2000): the membrane voltage creates an energy barrier which a gating particle (a charged molecule) must overcome to change states (e.g., to open). Changes in the membrane voltage modulate the size of the energy barriers, altering the rates of opening and closing of a gating particle. The average conductance of a channel is proportional to the percentage of the population of individual channels that are open.

Since transistors also involve the movement of a charged particle through an electric field, a transistor circuit can directly represent the action of a population of gating particles (Hynna and Boahen 2007). Figure 7.10 shows a thermodynamic model of a gating variable in which the drain current of transistor M2 in Figure 7.10a represents the gating particle’s rate of opening, while the source current of M1 represents the rate of closing. The voltage V_O controls the height of the energy barrier in M2: increasing V_O increases the opening rate, shifting u_V towards u_H . Increasing V_C has the opposite effect: the closing rate increases, shifting u_V towards u_L . Generally, V_O and V_C are inversely related; that is, as V_O increases, V_C should decrease.

The source of M2, u_V , is the log-domain representation of the gating variable u . Attaching u_V to the gate of a third transistor (not shown) realizes the variable u as a modulation of a

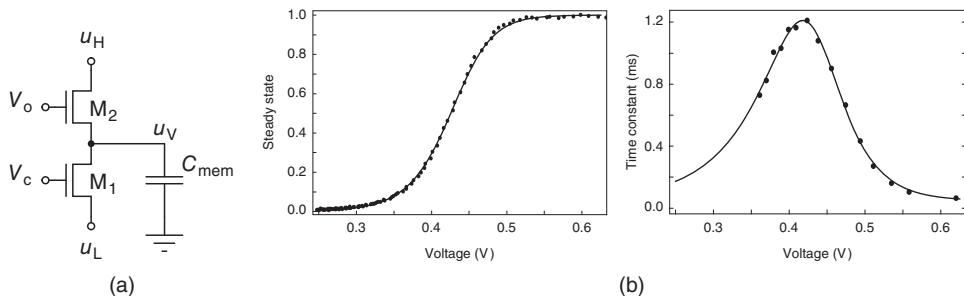


Figure 7.10 Thermodynamic model of a gating variable. (a) Gating variable circuit; (b) voltage dependence of the steady state and time constant of the variable circuit in (a). See Hynna and Boahen (2007) for details

current set by u_H . Connected as a simple activating channel – with V_o proportional to the membrane voltage (Hynna and Boahen 2007) – the voltage dependence of the steady state and time constant of u , as measured through the output transistor, match the sigmoid and bell-shaped curves commonly measured in neurophysiology (see Figure 7.10b).

Thalamic relay neurons possess a low-threshold calcium channel (also called a T-channel) and a slow inactivation variable, which turns off at higher voltages and opens at low voltages. The T-channel can be implemented using a fast activation variable, and implemented using the gating variable circuit of Figure 7.10. Figure 7.11a shows a simple two-compartment neuron circuit with a T-channel current, which can reproduce many response properties of real thalamic relay cells (Hynna and Boahen 2009). In the neuron circuit of Figure 7.11a the first block (on the left) integrates input spikes and represents the dendritic compartment, while the second block (on the right) produces output voltage spikes and represents the somatic compartment.

The dendritic compartment contains all active membrane components not involved in spike generation – namely, the synapses (e.g., one of the LPFs described in Section 7.2.1) and the T-channel – as well as common passive membrane components – a membrane capacitance (C_{mem}) and a membrane conductance (the nFET M1).

The somatic compartment, comprising a simple I&F neuron such as the Axon-Hillock circuit described in Section 7.2.2, receives input current from the dendrites through a diode-connected transistor (M2). Though a simple representation of a cell, relay neurons respond linearly in frequency to input currents (McCormick and Feeser 1990), just as an I&F cell. Due to the rectifying behavior of the diode (the pFET M2 in Figure 7.11a), current only passes from the dendrite to the soma. As a result, the somatic action potential does not propagate back to the dendrite; only the hyper-polarization (reset) that follows is evident in the dendritic voltage trace (V_{mem}). This is a simple approximation of dendritic low-pass filtering of the back-propagating signal.

When V_{mem} rests at higher voltages, the T-channel remains inactivated, and a step change in the input current simply causes the cell to respond with a constant frequency (see Figure 7.11c). If an inhibitory current is input into the cell, lowering the initial membrane voltage, then the T-channel deactivates prior to the step (see Figure 7.11b). Once the step occurs, V_{mem} begins to slowly increase until the T-channel activates, which excites the cell and causes it to burst.

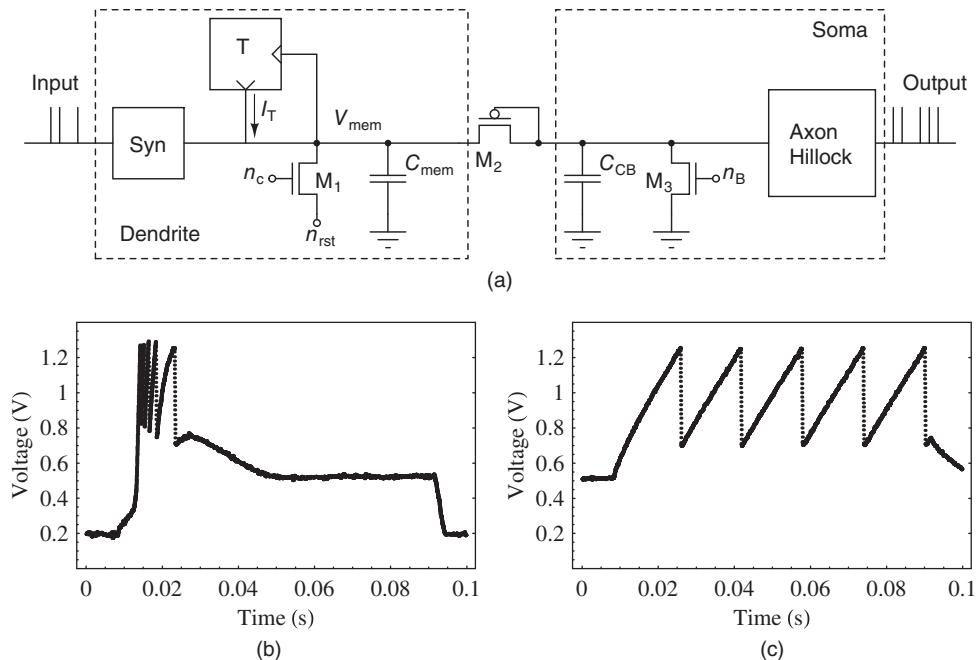


Figure 7.11 Two-compartment thalamic relay neuron model. (a) Neuron circuit. (b,c) Dendritic voltage (V_{mem}) measurements of the relay cell’s two response modes: burst (b) and tonic (c). An 80-ms-wide current step is injected into the dendritic compartment at 10 ms in both cases

Since V_{mem} is now much higher, the T-channel begins to inactivate, seen in the decrease of spike frequency within the burst on successive spikes, leading eventually to a cessation in spiking activity. In addition to the behavior shown here, this simple model also reproduces the thalamic response to sinusoidal inputs (Hynna and Boahen 2009).

The approach followed for this thalamic relay SiN can be extended by using and combining multiple instances of the basic building blocks described in Section 7.2.1.

7.3.2 Compact I&F Circuits for Event-Based Systems

We have shown examples of circuits used to implement faithful models of spiking neurons. These circuits can require significant amounts of silicon real estate. At the other end of the spectrum are compact circuits that implement basic models of I&F neurons. A common goal is to integrate very large numbers of these circuits on single chips to create large arrays of spiking elements, or large networks of neurons densely interconnected (Merolla et al. 2007; Schemmel et al. 2008; Vogelstein et al. 2007), which use the *Address-Event Representation* (AER) (Boahen 2000; Deiss et al. 1999; Lazzaro et al. 1993) to transmit spikes off-chip (see Chapter 2). It is therefore important to develop compact low-power circuits that implement useful abstractions of real neurons, but that can also produce very fast digital pulses required by the asynchronous circuits that manage the AER communication infrastructure.

As outlined in Chapter 3, a common application of basic I&F spiking circuits is their use in neuromorphic vision sensors. In this case the neuron is responsible for encoding the signal

measured by the photoreceptor and transmitting it off-chip using AER. In Azadmehr et al. (2005) and Olsson and Häfliger (2008), the authors used the *Axon-Hillock* circuit described in Section 7.2.2 to produce AER events. In Olsson and Häfliger (2008) the authors showed how this circuit can be interfaced to the AER interfacing circuits in a way to minimize device mismatch. Conversely, in Culurciello et al. (2003) the authors used a spiking neuron equivalent to the one of Figure 7.4, while in Lichtsteiner et al. (2008), the authors developed a compact ON/OFF neuron with good threshold matching properties, which is described in Section 3.5.1.

7.3.3 Generalized I&F Neuron Circuits

The simplified I&F neuron circuits described in the previous section require far fewer transistors and parameters than the biophysically realistic models of Section 7.3.1. But they do not produce a rich enough repertoire of behaviors useful for investigating the computational properties of large neural networks (Brette and Gerstner 2005; Izhikevich 2003). A good compromise between the two approaches can be obtained by implementing conductance-based or *generalized I&F* models (Jolivet et al. 2004). It has been shown that these types of models capture many of the properties of biological neurons, but require fewer and simpler differential equations compared to HH-based models (Brette and Gerstner 2005; Gerstner and Naud 2009; Izhikevich 2003; Jolivet et al. 2004; Mihalas and Niebur 2009). In addition to being efficient computational models for software implementations, these models lend themselves to efficient hardware implementation as well (Folowosele et al. 2009a; Folowosele et al. 2009b; Indiveri et al. 2010; Livi and Indiveri 2009; Rangan et al. 2010; van Schaik et al. 2010a, 2010b; Wijekoon and Dudek 2008).

The Tau-Cell Neuron

The circuit shown in Figure 7.12, dubbed as the ‘Tau-Cell neuron’ been used as the building block for implementations of both the Mihalas–Niebur neuron (van Schaik et al. 2010b) and the Izhikevich neuron (Rangan et al. 2010; van Schaik et al. 2010a). The basic leaky integrate-and-fire functionality is implemented using the Tau-Cell log-domain circuit described in Section 7.2.1. This approach uses current-mode circuits, so the state variable, which is normally

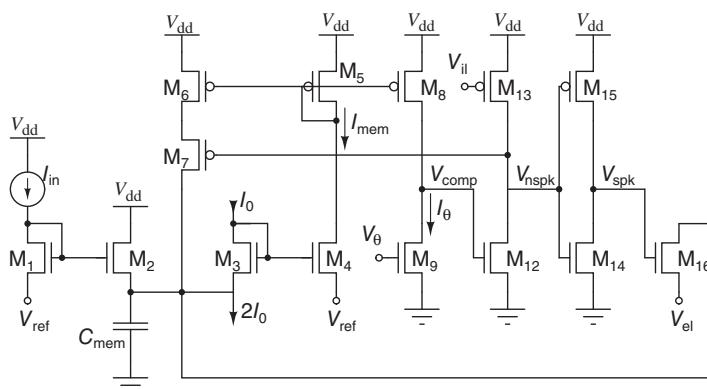


Figure 7.12 The Tau-Cell neuron circuit

the membrane voltage, V_{mem} , is transformed to a current I_{mem} . A Tau Cell, configured as a first-order LPF, is used to model the leaky integration.

In order to create a spike, I_{mem} is copied by pFETs M5 and M8 and compared with the constant threshold current I_θ . Since I_{mem} can be arbitrarily close to I_θ , a current limited inverter (M12, M13) is added to reduce power consumption while converting the result of the comparison into a digital value V_{nspk} . A positive voltage spike V_{spk} is generated with inverter M14, M15 with a slight delay with respect to V_{nspk} . pFETs M5–M7 implement positive feedback based on V_{nspk} while nFET M16 resets I_{mem} to a value determined by V_{el} . This reset causes the end of the positive feedback and the end of the spike, and the membrane is ready to start the next integration cycle.

The Log-Domain LPF Neuron

The log-domain LPF neuron (LLN) is a simple yet reconfigurable I&F circuit (Arthur and Boahen 2004, 2007) that can reproduce many of the behaviors expressed by generalized I&F models. Based on the LPF of Figure 7.2b, the LLN benefits from the log-domain design style's efficiency, using few transistors, operating with low-power (50–1000 nW), and requiring no complex configuration. The LLN realizes a variety of spiking behaviors: regular spiking, spike-frequency adaptation, and bursting (Figure 7.13b). The LLN's dimensionless membrane potential v , and adaptive conductance g variable (proportional to I_v and I_g of Figure 7.13a respectively), can be described by the following set of equations:

$$\begin{aligned} \tau \frac{d}{dt}v &= -v(1+g) + v_\infty + \frac{v^3}{3} \\ \tau_g \frac{d}{dt}g &= -g + g_{\max} r(t), \end{aligned} \quad (7.1)$$

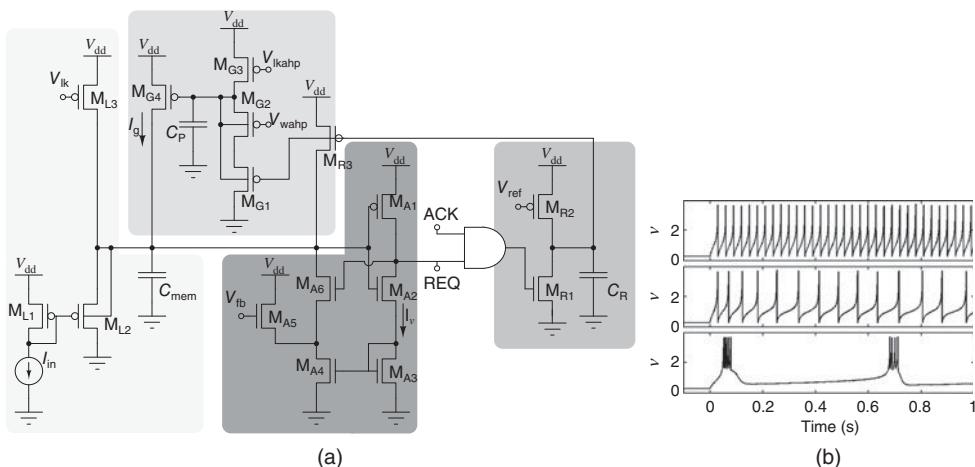


Figure 7.13 The log-domain LPF neuron (LLN). (a) The LLN circuit comprises a membrane LPF (very light gray, M_{L1}–3), a spike-event generation and positive feedback element (dark gray, M_{A1}–6), a reset-refractory pulse generator (mid gray, M_{R1}–3), and a spike-frequency adaptation LPF (light gray, M_{G1}–4). (b) Recorded and normalized traces from an LLN fabricated in 0.25 μm CMOS exhibits regular spiking, spike-frequency adaptation, and bursting (top to bottom)

where v_∞ is v 's steady state level in the absence of positive feedback and $g = 0$; τ and τ_g are the membrane and adaptive conductance time constants, respectively; and g_{\max} is the adaptive conductance's absolute maximum value. When v reaches a high level ($\gg 10$), a spike is emitted, and $r(t)$ is set high for a brief period, T_R . $r(t)$ is a reset–refractory signal, driving v low (not shown in equation).

The LLN is composed of four subcircuits (see Figure 7.13a): A membrane LPF (M_{L1-3}), a spike event generation and positive feedback element (M_{A1-6}), a reset–refractory pulse generator (M_{R1-3}), and an adaptation LPF (M_{G1-4}). The membrane LPF realizes $I_v (\propto v)$'s first-order (resistor–capacitor) dynamics in response to $I_{in} (\propto v_\infty)$. The positive feedback element drives the membrane LPF in proportion to the cube of v , analogous to a biological sodium channel population. When the membrane LPF is sufficiently driven, $\frac{v^3}{3} > v$, resulting in a run-away potential, that is, a spike. The digital representation of the spike is transmitted as an AER request (REQ) signal. After a spike (upon arrival of the AER acknowledge signal ACK), the refractory pulse generator creates a pulse, $r(t)$ with a tunable duration. When active $r(t)$ turns M_{G1} and M_{R3} on, resetting the membrane LPF (toward V_{DD}) and activating the adaptation LPF. Once activated the adaptation LPF inhibits the membrane LPF, realizing $I_g (\propto g)$, which is proportional to spike frequency.

Implementing an LLN's various spiking behaviors is a matter of setting its biases. To implement regular spiking, we set $g_{\max} = 0$ (set by M_{G2} 's bias voltage V_{wahp}) and $T_R = 1$ ms (long enough to drive v to 0, set by M_{R2} 's bias voltage V_{ref}). Spike-frequency adaptation can be obtained by allowing the adaptation LPF (M_{G1-4}) to integrate the spikes produced by the neuron itself. This is done by increasing g_{\max} and setting $\tau_g = 100$ ms (*i.e.*, by adjusting V_{lkahp} appropriately). Similarly, the bursting behavior is obtained by decreasing the duration of the $r(t)$ pulse such that v is not pulled below 1 after each spike.

The DPI Neuron

The DPI neuron is another variant of a *generalized I&F* model (Jolivet et al. 2004). This circuit has the same functional blocks used by the LLN of Figure 7.13a, but different instantiations of LPFs and current-based positive feedback circuits: the LPF behavior is implemented using instances of the tunable Diff-Pair Integrator circuit described in Section 7.2.1, while the positive feedback is implemented using the same circuits used in the octopus neuron of Figure 7.4. These are small differences from the point of view of transistor count and circuit details, but have an important effect on the properties of the SiN.

The DPI neuron circuit is shown in Figure 7.14a. It comprises an input DPI filter ($M_{L1} - M_{L3}$), a spike event generating amplifier with current-based positive feedback ($M_{A1} - M_{A6}$), a spike reset circuit with AER handshaking signals and refractory period functionality ($M_{R1} - M_{R6}$), and a spike-frequency adaptation mechanism implemented by an additional DPI filter ($M_{G1} - M_{G6}$). The input DPI filter $M_{L1} - M_{L3}$ models the neuron's leak conductance, producing exponential subthreshold dynamics in response to constant input currents. The integrating capacitor C_{mem} represents the neuron's membrane capacitance, and the positive-feedback circuits in the spike-generation amplifier model both sodium channel activation and inactivation dynamics. The reset and refractory period circuit models the potassium conductance functionality. The spike-frequency adaptation DPI circuit models the neuron's calcium conductance and produces an after hyper-polarizing current (I_g) proportional to the neuron's mean firing rate.

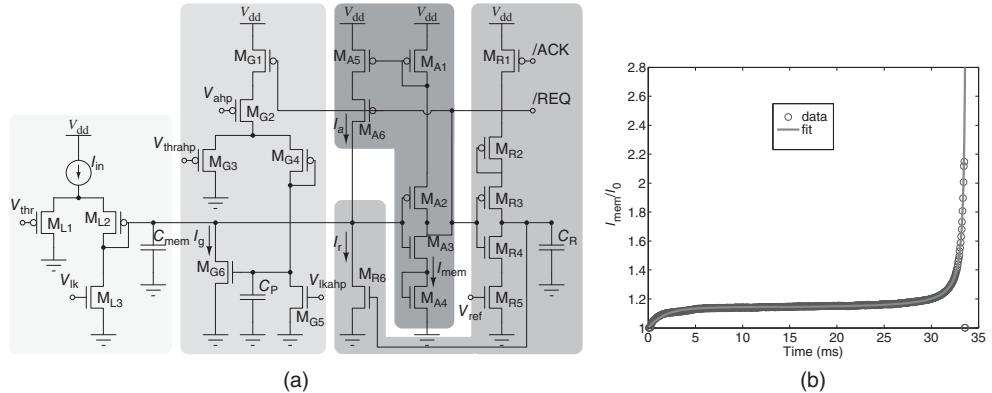


Figure 7.14 The DPI neuron circuit. (a) Circuit schematic. The input DPI LPF (very light gray, M_{L1} – M_{L3}) models the neuron’s leak conductance. A spike event generation amplifier (dark gray, M_{A1} – M_{A6}) implements current-based positive feedback (modeling both sodium activation and inactivation conductances) and produces address events at extremely low power. The reset block (mid gray, M_{R1} – M_{R6}) resets the neuron and keeps it in a reset state for a refractory period, set by the V_{ref} bias voltage. An additional DPI filter integrates the spikes and produces a slow after hyper-polarizing current I_g responsible for spike-frequency adaptation (light gray, M_{G1} – M_{G6}). (b) Response of the DPI neuron circuit to a constant input current. The measured data was fitted with a function comprising an exponential $\propto e^{-t/\tau_K}$ at the onset of the stimulation, characteristic of all conductance-based models, and an additional exponential $\propto e^{+t/\tau_{Na}}$ (characteristic of exponential I&F computational models, Brette and Gerstner 2005) at the onset of the spike. © 2010 IEEE. Reprinted, with permission, from Indiveri et al. (2010)

By applying a current-mode analysis to both the input and the spike-frequency adaptation DPI circuits (Bartolozzi et al. 2006; Livi and Indiveri 2009), it is possible to derive a simplified analytical solution (Indiveri et al. 2010), very similar to the one described in Eq. (7.1), of the form:

$$\begin{aligned} \tau \frac{d}{dt} I_{\text{mem}} &= -I_{\text{mem}} \left(1 + \frac{I_g}{I_\tau} \right) + I_{\text{mem}\infty} + f(I_{\text{mem}}) \\ \tau_g \frac{d}{dt} I_g &= -I_g + I_{g_{\max}} r(t), \end{aligned} \quad (7.2)$$

where I_{mem} is the subthreshold current analogous to the state variable v of Eq. (7.1) and I_g corresponds to the slow variable g of Eq. (7.1) responsible for spike-frequency adaptation. The term $f(I_{\text{mem}})$ accounts for the positive-feedback current I_a of Figure 7.14a and is an exponential function of I_{mem} (Indiveri et al. 2010) (see also Figure 7.14b). As for the LLN, the function $r(t)$ is unity for the period in which the neuron spikes, and zero in other periods. The other parameters in Eq. (7.2) are defined as

$$\tau \triangleq \frac{C U_T}{\kappa I_\tau}, \quad \tau_g \triangleq \frac{C_p U_T}{\kappa I_{\tau_g}}, \quad I_{\text{mem}\infty} \triangleq \left(\frac{I_{\text{in}}}{I_\tau} \right) I_0 e^{\frac{\kappa}{U_T} V_{\text{thr}}}, \quad \text{and} \quad I_{g_{\max}} \triangleq \left(\frac{I_{M_{G2}}}{I_{\tau_g}} \right) I_0 e^{\frac{\kappa}{U_T} V_{\text{thr}a}}$$

where $I_\tau = I_0 e^{\frac{\kappa}{U_T} V_{\text{lk}}}$, and $I_{\tau_g} = I_0 e^{\frac{\kappa}{U_T} V_{\text{ikahp}}}$.

By changing the biases that control the neuron's time constants, refractory period, spike-frequency adaptation dynamics, and leak behavior (Indiveri et al. 2010), the DPI neuron can produce a wide range of spiking behaviors (from regular spiking to bursting).

Indeed, given the exponential nature of the generalized I&F neuron's nonlinear term $f(I_{\text{mem}})$, the DPI neuron implements an *adaptive exponential* I&F model. This I&F model has been shown to be able to reproduce a wide range of spiking behaviors and explain a wide set of experimental measurements from pyramidal neurons (Brette and Gerstner 2005). For comparison, the LLN uses a cubic term, while the Tau-Cell-based neuron circuits proposed in van Schaik et al. (2010a) and Rangan et al. (2010) and the quadratic and the switched-capacitor SiNs described in Section 7.3.4 use a quadratic term (implementing the I&F computational models proposed by Izhikevich 2003).

7.3.4 Above Threshold, Accelerated-Time, and Switched-Capacitor Designs

The SiN circuits described up to now have transistors that operate mostly in the subthreshold or weak-inversion domain, with currents ranging typically between fractions of pico- to hundreds of nano-amperes. These circuits have the advantage of being able to emulate real neurons with extremely low-power requirements and with realistic time constants (e.g., for interacting with the nervous system, or implementing real-time behaving systems with time constants matched to those of the signals they process). However, in the weak-inversion domain, mismatch effects are more pronounced than in the strong-inversion regime (Pelgrom et al. 1989) and often require learning, adaptation, or other compensation schemes.

It has been argued that in order to faithfully reproduce computational models simulated on digital architectures, it is necessary to design analog circuits with low mismatch and high precision (Schemmel et al. 2007). For this reason, several SiN circuits that operate in the strong-inversion regime have been proposed. In this regime however, currents are four to five orders of magnitude larger. With such currents, the active circuits used to implement resistors decrease their resistance values dramatically. As passive resistors cannot be easily implemented in VLSI to yield large resistance values, it is either necessary to use large off-chip capacitors (and small numbers of neurons per chip), to obtain biologically realistic time constants, or to use 'accelerated' time scales, in which the time constants of the SiNs are a factor of 10^3 or 10^4 larger than those of real neurons. Alternatively, one can use switched-capacitors (S-C) for implementing small conductances (and therefore long time constants) by moving charge in and out of integrated capacitors with clocked switches. Taking this concept one step further, one can implement SiNs using full custom clocked digital circuits. All of these approaches are outlined in this section.

The Quadratic I&F Neuron

As for the subthreshold case, implementations of biophysically detailed models such as the one described above can be complemented by more compact implementations of simplified I&F models.

The quadratic I&F neuron circuit (Wijekoon and Dudek 2008), shown in Figure 7.15a, is an example of an above-threshold generalized I&F circuit. It was inspired by the adapting quadratic I&F neuron model proposed by Izhikevich (2003). The required nonlinear oscillatory behavior is achieved using differential equations of two state variables and a separate

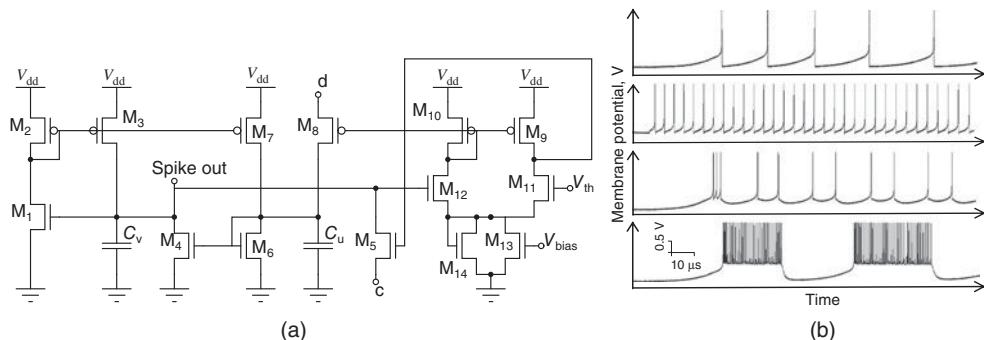


Figure 7.15 The Izhikevich neuron circuit. (a) Schematic diagram; (b) data recorded from the $0.35\text{ }\mu\text{m}$ CMOS VLSI implementation: spiking patterns in response to an input current step for various parameters of bias voltages at node c and node d: regular spiking with adaptation, fast spiking, intrinsically bursting, chattering (top to bottom)

after-spike reset mechanism, as explained in Izhikevich (2003). However, the circuit implementation does not aim to accurately replicate the nonlinear equations described in Izhikevich (2003). Instead it aims at using the simplest possible circuitry of the analog VLSI implementation that can reproduce the functional behavior of the coupled system of nonlinear equations.

The two state variables, ‘membrane potential’ (V) and ‘slow variable’ (U), are represented by voltages across capacitors C_v and C_u , respectively. The membrane potential circuit consists of transistors M1–M5 and membrane capacitor C_v . The membrane capacitor integrates post-synaptic input currents, the spike-generating positive feedback current of M3 and the leakage current generated by M4 (mostly controlled by the slow variable U). The positive feedback current is generated by M1 and mirrored by M2–M3 and depends approximately quadratically on the membrane potential. If a spike is generated, it is detected by the comparator circuit (M9–M14), which provides a reset pulse on the gate of M5 that rapidly hyperpolarizes the membrane potential to a value determined by the voltage at node c . The slow variable circuit is built using transistors M1, M2, and M6–M8. The magnitude of the current provided by M7 is determined by the membrane potential, in a way similar to the membrane circuit. The transistor M6 provides a nonlinear leakage current. The transistors and capacitances are scaled so that the potential U will vary more slowly than V . Following a membrane potential spike, the comparator generates a brief pulse to turn on transistor M8 so that an extra amount of charge, controlled by the voltage at node d , is transferred onto C_u . The circuit has been designed and fabricated in a $0.35\text{ }\mu\text{m}$ CMOS technology. It is integrated in a chip containing 202 neurons with various circuit parameters (transistor sizes and capacitances). As the transistors in this circuit operate mostly in strong inversion, the firing patterns are on an ‘accelerated’ timescale, about 10^4 faster than biological real time (see Figure 7.15b). The power consumption of the circuit is below 10 pJ/spike . A similar circuit, but operating in weak inversion and providing spike timings on a biological timescale, has been presented in Wijekoon and Dudek (2009).

The Switched-Capacitor Mihalas–Niebur Neuron

Switched-capacitors (S-C) have long been used in integrated circuit design to enable the implementation of variable resistors whose sizes can vary over several orders of magnitude.

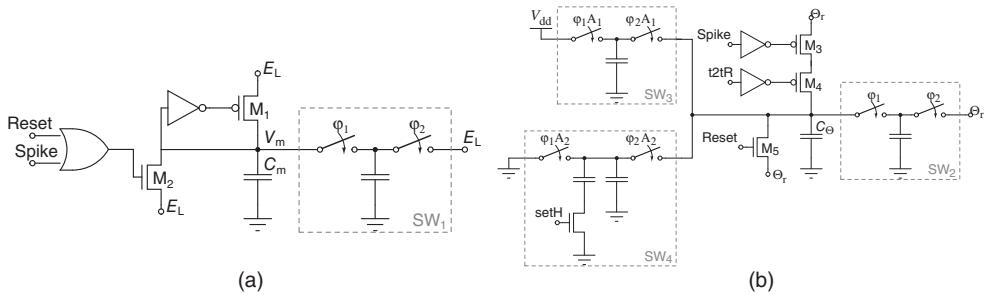


Figure 7.16 Switched capacitor Mihalas–Niebur neuron implementation. (a) Neuron membrane circuit; (b) adaptive threshold circuit

This technique can be used as a method of implementing resistors in silicon neurons, which is complementary to the methods described in the previous sections. More generally, S-C implementations of SiNs produce circuits whose behaviors are robust, predictable, and reproducible (properties that are not always observed with subthreshold SiN implementations).

The circuit shown in Figure 7.16a implements a leaky I&F neuron implemented with S-Cs (Folowosele et al. 2009a). Here the post-synaptic current is input onto the neuron membrane, V_m . The S-C, SW_1 , acts as the ‘leak’ between the membrane potential, V_m , and the resting potential of the neuron, E_L . The value of the leak is varied by changing either the capacitor in SW_1 or the frequency of the clocks ϕ_1 and ϕ_2 . A comparator (not shown) is used to compare the membrane voltage V_m with a reset voltage Θ_r . Once V_m exceeds Θ_r a ‘spike’ voltage pulse is issued and V_m is reset to the resting potential E_L .

The Mihalas–Niebur S-C neuron (Mihalas and Niebur 2009) is built by combining the I&F circuit of Figure 7.16a with the variable threshold circuit shown in Figure 7.16b. The circuit blocks are arranged in a way to implement the adaptive threshold mechanism described in Figure 7.2.4. As the circuits used for realizing the membrane and the threshold equations are identical, the density of arrays of these neurons can be doubled when simple I&F with fixed threshold properties are desired. The main drawback of this approach is the need for multiple phases of the S-C clocks which must be distributed (typically in parallel) to each neuron.

Experimental results measured from a fabricated integrated circuit implementing this neuron model (Indiveri et al. 2010) are shown in Figure 7.17. The ease with which these complex behaviors can be evoked in S-C neurons, without extensive and precise tuning, demonstrates their utility in large silicon neuron arrays.

7.4 Discussion

While the digital processing paradigm, ranging from standard computer simulations to custom FPGA designs, is advantageous for its stability, fast development times, and high precision properties, full custom VLSI solutions can often be optimized in terms of power consumption, silicon area usage, and speed/bandwidth usage. This is especially true for silicon neurons and custom analog/digital VLSI neural networks, which are the optimal solution to a large variety of applications, ranging from the efficient implementation of large-scale and real-time

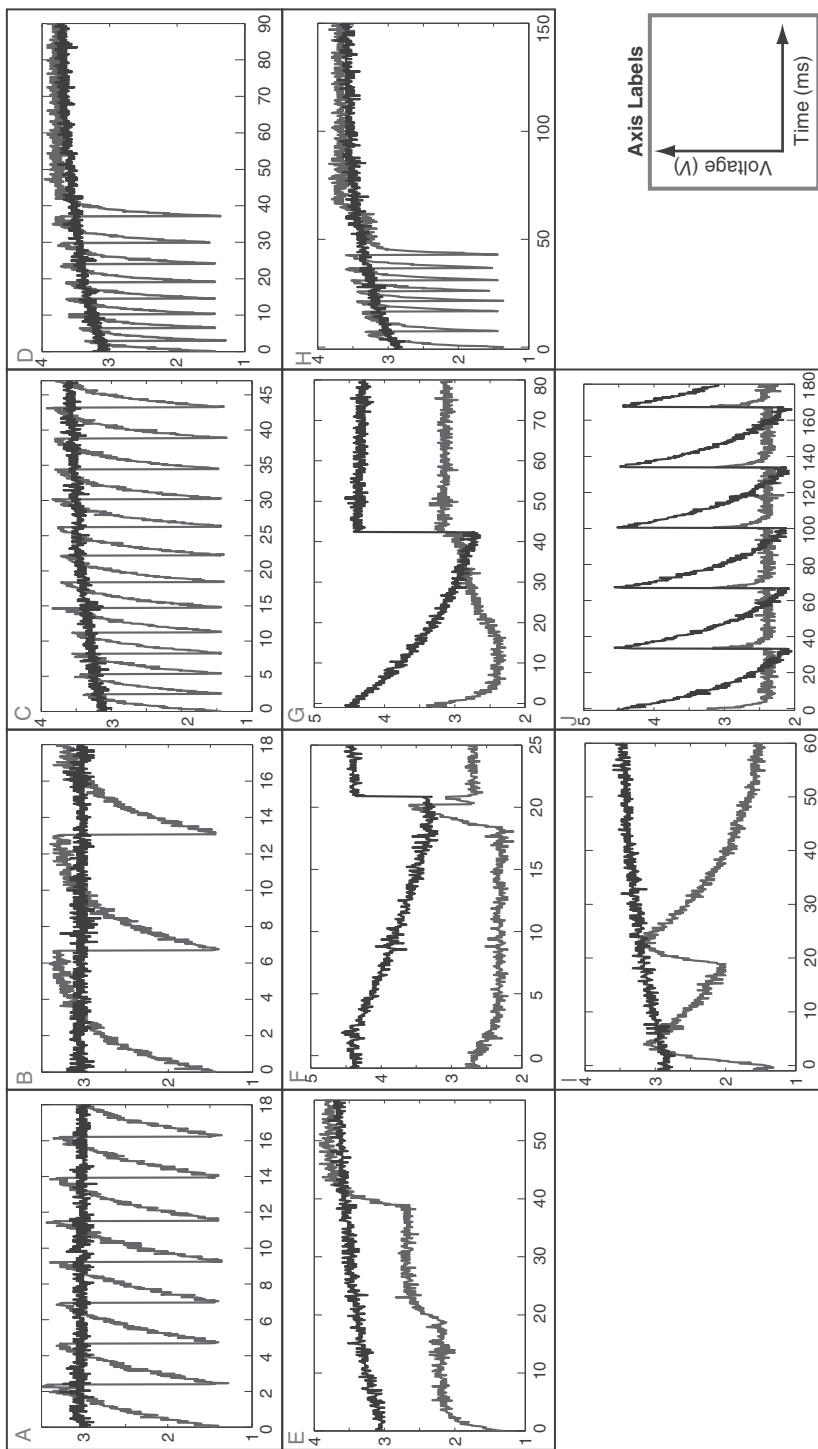


Figure 7.17 S-C Mihalas-Niebur neuron circuit results demonstrating ten of the known spiking properties that have been observed in biological neurons. Membrane voltage and adaptive threshold are illustrated in gray and black, respectively. A – tonic spiking, B – class 1 spiking, C – spike-frequency adaptation, D – phasic spiking, E – accommodation, F – rebound spiking, G – threshold variability, H – input bistability, I – integrator, J – hyper-polarized spiking

spike-based computing systems to the implementation of compact microelectronic brain-machine interfaces. In particular, even though subthreshold current-mode circuits are reputed to have higher mismatch than above-threshold circuits, they have lower noise energy (noise power times bandwidth) and superior energy efficiency (bandwidth over power) (Sarpeshkar et al. 1993). Indeed, the sources of inhomogeneities (e.g., device mismatch), which are often considered a problem, can actually be exploited in networks of SiNs for computational purposes (similar to how real neural systems exploit noise) (Chicca and Fusi 2001; Chicca et al. 2003; Merolla and Boahen 2004). Otherwise, sources of mismatch can be minimized at the device level with clever VLSI layout techniques (Liu et al. 2002) and at the system level by using the same strategies used by the nervous system (e.g., adaptation and learning at multiple spatial and temporal scales). Furthermore, by combining the advantages of synchronous and asynchronous digital technology with those of analog circuits, it is possible to efficiently calibrate component parameters and (re)configure SiN network topologies both for single chip solutions and for large-scale multichip networks (Basu et al. 2010; Linares-Barranco et al. 2003; Sheik et al. 2011; Silver et al. 2007; Yu and Cauwenberghs 2010).

Obviously, there is no absolute optimal design. As there is a wide range of neuron types in biology, there is a wide range of design and circuit choices for SiNs. While the implementations of conductance-based models can be useful for applications in which a small numbers of SiNs are required (as in hybrid systems where real neurons are interfaced to silicon ones), the compact AER I&F neurons and log-domain implementations (such as the quadratic Mihalas–Niebur neurons, the Tau-Cell neuron, the LPF neuron, or the DPI neuron) can be integrated with event-based communication fabric and synaptic arrays for very large-scale reconfigurable networks. Indeed, both the subthreshold implementations and their above-threshold ‘accelerated-time’ counterparts are very amenable for dense and low-power integration with energy efficiencies of the order of a few pico-Joules per spike (Livi and Indiveri 2009; Rangan et al. 2010; Schemmel et al. 2008; Wijekoon and Dudek 2008). In addition to continuous time, nonclocked subthreshold, and above-threshold design techniques, we showed how to implement SiNs using digitally modulated charge packet and switched-capacitor (S-C) methodologies. The S-C Mihalas–Niebur SiN circuit is a particularly robust design which exhibits the model’s generalized linear I&F properties and can produce up to ten different spiking behaviors. The specific choice of design style and SiN circuit to use depends on its application. Larger and highly configurable designs that can produce a wide range of behaviors are more amenable to research projects in which scientists explore the parameter space and compare the VLSI device behavior with that of its biological counterpart. Conversely, the more compact designs will be used in specific applications where signals need to be encoded as sequences of spikes, and where size and power budgets are critical. Figure 7.18 illustrates some of the most relevant silicon neuron designs.

The sheer volume of silicon neuron designs proposed in the literature demonstrates the enormous opportunities for innovation when inspiration is taken from biological neural systems. The potential applications span computing and biology: neuromorphic systems are providing the clues for the next generation of asynchronous, low-power, parallel computing that could bridge the gap in computing power when Moore’s law runs its course, while hybrid, silicon neuron systems are allowing neuroscientists to unlock the secrets of neural circuits, possibly leading one day, to fully integrated brain–machine interfaces. New emerging technologies (e.g., memristive devices) and their utility in enhancing spiking silicon neural networks must also be evaluated, as well as maintaining a knowledge base of the existing technologies that

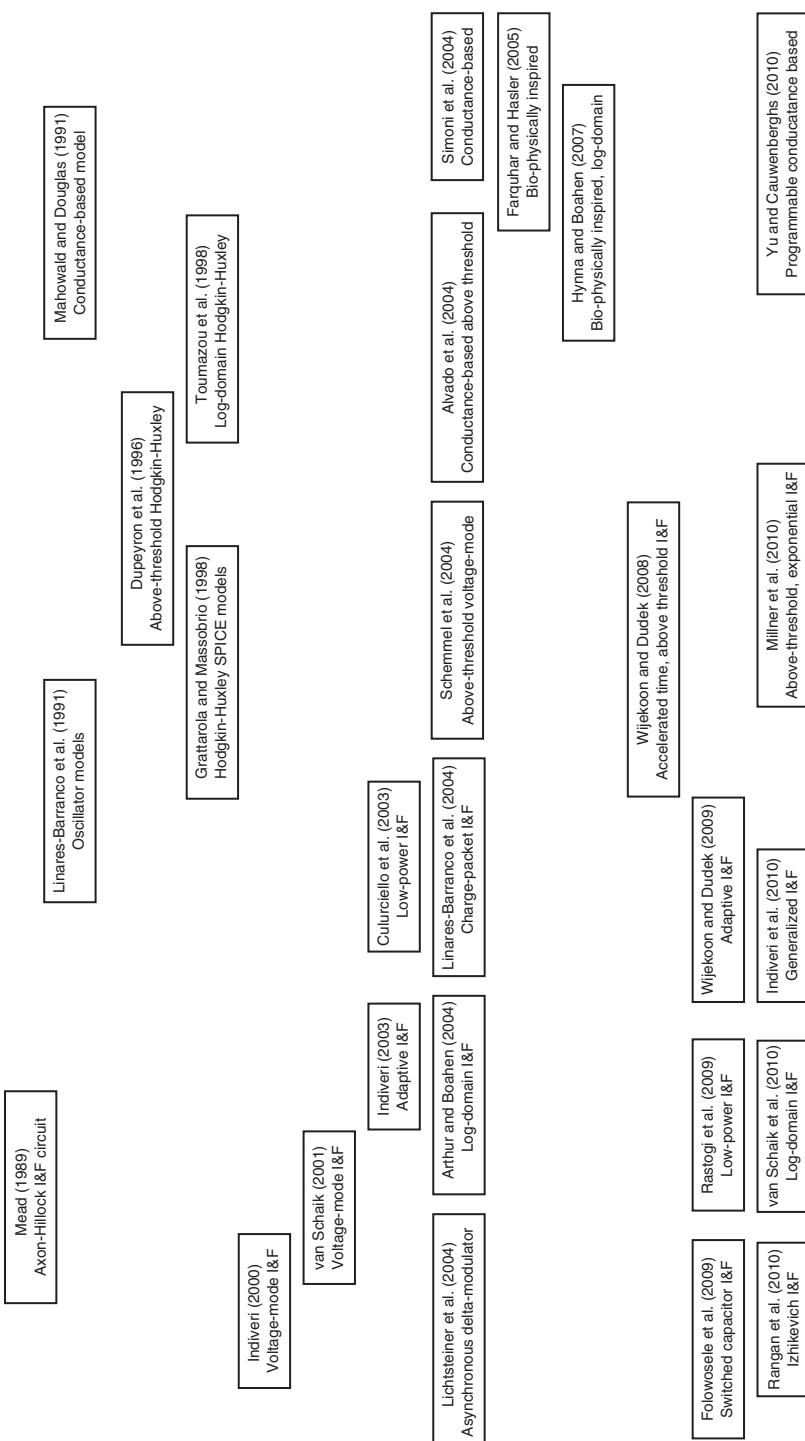


Figure 7.18 Silicon neuron design timeline. The diagram is organized such that time increases vertically, from top to bottom, and complexity increases horizontally, with basic integrate and fire models on the left and complex conductance-based models on the right. The corresponding publications with additional information on the designs are given in the References

have been proven to be successful in silicon neuron design. Furthermore, as larger on-chip spiking silicon neural networks are developed, questions of communication protocols (e.g., AER), on-chip memory, size, programmability, adaptability, and fault tolerance also become very important. In this respect, the SiN circuits and design methodologies described in this paper provide the building blocks that will pave the way for these extraordinary breakthroughs.

References

- Alvado L, Tomas J, Saighi S, Renaud-Le Masson S, Bal T, Destexhe A, and Le Masson G. 2004. Hardware computation of conductance-based neuron models. *Neurocomputing* **58–60**, 109–115.
- Arthur JV and Boahen K. 2004. Recurrently connected silicon neurons with active dendrites for one-shot learning. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)* **3**, pp. 1699–1704.
- Arthur JV and Boahen K. 2007. Synchrony in silicon: the gamma rhythm. *IEEE Trans. Neural Netw.* **18**, 1815–1825.
- Azadmehr M, Abrahamsen JP, and Hafliger P. 2005. A foveated AER imager chip. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 2751–2754.
- Bartolozzi C and Indiveri G. 2007. Synaptic dynamics in analog VLSI. *Neural Comput.* **19**(10), 2581–2603.
- Bartolozzi C and Indiveri G. 2009. Global scaling of synaptic efficacy: homeostasis in silicon synapses. *Neurocomputing* **72**(4–6), 726–731.
- Bartolozzi C, Mitra S, and Indiveri G. 2006. An ultra low power current-mode filter for neuromorphic systems and biomedical signal processing. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 130–133.
- Basu A, Ramakrishnan S, Petre C, Koziol S, Brink S, and Hasler PE. 2010. Neural dynamics in reconfigurable silicon. *IEEE Trans. Biomed. Circuits Syst.* **4**(5), 311–319.
- Boahen KA. 2000. Point-to-point connectivity between neuromorphic chips using address-events. *IEEE Trans. Circuits Syst. II* **47**(5), 416–434.
- Brette R and Gerstner W. 2005. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophys.* **94**, 3637–3642.
- Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, Goodman PH, Harris Jr. FC, Zirpe M, Natschläger T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison AP, El Boustani S, and Destexhe A. 2007. Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* **23**(3), 349–398.
- Chicca E and Fusi S. 2001. Stochastic synaptic plasticity in deterministic aVLSI networks of spiking neurons. In: *Proceedings of the World Congress on Neuroinformatics* (ed. Rattay F). ARGESIM/ASIM Verlag, Vienna. pp. 468–477.
- Chicca E, Badoni D, Dante V, D'Andreagiovanni M, Salina G, Carota L, Fusi S, and Del Giudice P. 2003. A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory. *IEEE Trans. Neural Netw.* **14**(5), 1297–1307.
- Connors BW, Gutnick MJ, and Prince DA. 1982. Electrophysiological properties of neocortical neurons in vitro. *J. Neurophys.* **48**(6), 1302–1320.
- Costas-Santos J, Serrano-Gotarredona T, Serrano-Gotarredona R, and Linares-Barranco B. 2007. A spatial contrast retina with on-chip calibration for neuromorphic spike-based AER vision systems. *IEEE Trans. Circuits Syst. I* **54**(7), 1444–1458.
- Culurciello E, Etienne-Cummings R, and Boahen K. 2003. A biomorphic digital image sensor. *IEEE J. Solid-State Circuits* **38**(2), 281–294.
- Deiss SR, Douglas RJ, and Whatley AM. 1999. A pulse-coded communications infrastructure for neuromorphic systems [Chapter 6]. In: *Pulsed Neural Networks* (eds Maass W and Bishop CM). MIT Press, Cambridge, MA. pp. 157–178.
- Destexhe A and Huguenard JR. 2000. Nonlinear thermodynamic models of voltage-dependent currents. *J. Comput. Neurosci.* **9**(3), 259–270.
- Destexhe A, Mainen ZF, and Sejnowski TJ. 1998. Kinetic models of synaptic transmission. *Methods in Neuronal Modelling, from Ions to Networks*. The MIT Press, Cambridge, MA. pp. 1–25.
- Drakakis EM, Payne AJ, and Toumazou C. 1997. Bernoulli operator: a low-level approach to log-domain processing. *Electron. Lett.* **33**(12), 1008–1009.

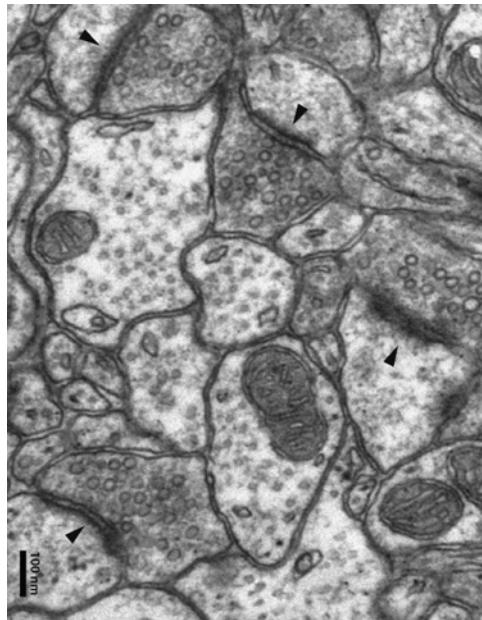
- Dupeyron D, Le Masson S, Deval Y, Le Masson G, and Dom JP. 1996. A BiCMOS implementation of the Hodgkin-Huxley formalism. In: *Proceedings of Fifth International Conference on Microelectronics Neural, Fuzzy and Bio-inspired Systems (MicroNeuro)*. IEEE Computer Society Press, Los Alamitos, CA. pp. 311–316.
- Edwards RT and Cauwenberghs G. 2000. Synthesis of log-domain filters from first-order building blocks. *Analog Integr. Circuits Signal Process.* **22**, 177–186.
- Farquhar E and Hasler P. 2005. A bio-physically inspired silicon neuron. *IEEE Trans. Circuits Syst. I: Regular Papers* **52**(3), 477–488.
- Farquhar E, Abramson D, and Hasler P. 2004. A reconfigurable bidirectional active 2 dimensional dendrite model. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **1**, pp. 313–316.
- Folowosele F, Etienne-Cummings R, and Hamilton TJ. 2009a. A CMOS switched capacitor implementation of the Mihalas-Niebur neuron. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 105–108.
- Folowosele F, Harrison A, Cassidy A, Andreou AG, Etienne-Cummings R, Mihalas S, Niebur, and Hamilton TJ. 2009b. A switched capacitor implementation of the generalized linear integrate-and-fire neuron. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2149–2152.
- Frey DR. 1993. Log-domain filtering: an approach to current-mode filtering. *IEE Proc. G: Circuits, Devices and Systems.* **140**(6), 406–416.
- Fusi S, Annunziato M, Badoni D, Salamon A, and Amit DJ. 2000. Spike-driven synaptic plasticity: theory, simulation, VLSI implementation. *Neural Comput.* **12**(10), 2227–2258.
- Gerstner W and Naud R. 2009. How good are neuron models? *Science* **326**(5951), 379–380.
- Gilbert B. 1975. Translinear circuits: a proposed classification. *Electron. Lett.* **11**, 14–16.
- Grattarola M and Massobrio G. 1998. *Bioelectronics Handbook : MOSFETs, Biosensors, and Neurons*. McGraw-Hill, New York.
- Hasler P, Kozoi S, Farquhar E, and Basu A. 2007. Transistor channel dendrites implementing HMM classifiers. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3359–3362.
- Hynna KM and Boahen K. 2006. Neuronal ion-channel dynamics in silicon. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3614–3617.
- Hynna KM and Boahen K. 2007. Thermodynamically-equivalent silicon models of ion channels. *Neural Comput.* **19**(2), 327–350.
- Hynna KM and Boahen K. 2009. Nonlinear influence of T-channels in an in silico relay neuron. *IEEE Trans. Biomed. Eng.* **56**(6), 1734.
- Indiveri G. 2000. Modeling selective attention using a neuromorphic analog VLSI device. *Neural Comput.* **12**(12), 2857–2880.
- Indiveri G. 2003. A low-power adaptive integrate-and-fire neuron circuit. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **4**, pp. 820–823.
- Indiveri G. 2007. Synaptic plasticity and spike-based computation in VLSI networks of integrate-and-fire neurons. *Neural Inform. Process. – Letters and Reviews* **11**(4–61), 135–146.
- Indiveri G, Horiuchi T, Niebur E, and Douglas R. 2001. A competitive network of spiking VLSI neurons. In: *World Congress on Neuroinformatics* (ed. Rattay F). ARGESIM / ASIM Verlag, Vienna. ARGESIM Report No. 20. pp. 443–455.
- Indiveri G, Chicca E, and Douglas RJ. 2009. Artificial cognitive systems: from VLSI networks of spiking neurons to neuromorphic cognition. *Cognit. Comput.* **1**, 119–127.
- Indiveri G, Stefanini F, and Chicca E. 2010. Spike-based learning with a generalized integrate and fire silicon neuron. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1951–1954.
- Indiveri G, Linares-Barranco B, Hamilton TJ, van Schaik A, Etienne-Cummings R, Delbrück T, Liu SC, Dudek P, Häfliger P, Renaud S, Schemmel J, Cauwenberghs G, Arthur J, Hynna K, Folowosele F, Saighi S, Serrano-Gotarredona T, Wijekoon J, Wang Y, and Boahen K. 2011. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience* **5**, 1–23.
- Izhikevich EM. 2003. Simple model of spiking neurons. *IEEE Trans. Neural Netw.* **14**(6), 1569–1572.
- Jolivet R, Lewis TJ, and Gerstner W. 2004. Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy. *J. Neurophys.* **92**, 959–976.
- Koch C. 1999. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press.
- Lazzaro J, Wawrzynek J, Mahowald M, Sivilotti M, and Gillespie D. 1993. Silicon auditory processors as computer peripherals. *IEEE Trans. Neural Netw.* **4**(3), 523–528.
- Le Masson G, Renaud S, Debay D, and Bal T. 2002. Feedback inhibition controls spike transfer in hybrid thalamic circuits. *Nature* **417**, 854–858.

- Leñero Bardallo JA, Serrano-Gotarredona T, and Linares-Barranco B. 2010. A five-decade dynamic-range ambient-light-independent calibrated signed-spatial-contrast AER retina with 0.1-ms latency and optional time-to-first-spike mode. *IEEE Trans. Circuits Syst. I: Regular Papers* **57**(10), 2632–2643.
- Lewis MA, Etienne-Cummings R, Hartmann M, Cohen AH, and Xu ZR. 2003. An in silico central pattern generator: silicon oscillator, coupling, entrainment, physical computation and biped mechanism control. *Biol. Cybern.* **88**(2), 137–151.
- Lichtsteiner P, Delbrück T, and Kramer J. 2004. Improved on/off temporally differentiating address-event imager. *Proc. 11th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 211–214.
- Lichtsteiner P, Posch C, and Delbrück T. 2008. A 128×128 120 dB 15us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* **43**(2), 566–576.
- Linares-Barranco B and Serrano-Gotarredona T. 2003. On the design and characterization of femtoampere current-mode circuits. *IEEE J. Solid-State Circuits* **38**(8), 1353–1363.
- Linares-Barranco B, Sánchez-Sinencio E, Rodríguez Vázquez A, and Huertas JL. 1991. A CMOS implementation of Fitzhugh-Nagumo neuron model. *IEEE J. Solid-State Circuits* **26**(7), 956–965.
- Linares-Barranco B, Serrano-Gotarredona T, and Serrano-Gotarredona R. 2003. Compact low-power calibration mini-DACs for neural massive arrays with programmable weights. *IEEE Trans. Neural Netw.* **14**(5), 1207–1216.
- Linares-Barranco B, Serrano-Gotarredona T, Serrano-Gotarredona R, and Serrano-Gotarredona C. 2004. Current mode techniques for sub-pico-ampere circuit design. *Analog Integr. Circuits Signal Process.* **38**, 103–119.
- Liu SC, Kramer J, Indiveri G, Delbrück T, Burg T, and Douglas R. 2001. Orientation-selective aVLSI spiking neurons. *Neural Netw.* **14**(6/7), 629–643.
- Liu SC, Kramer J, Indiveri G, Delbrück T, and Douglas R. 2002. *Analog VLSI: Circuits and Principles*. MIT Press.
- Livi P and Indiveri G. 2009. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2898–2901.
- Mahowald M and Douglas R. 1991. A silicon neuron. *Nature* **354**, 515–518.
- McCormick DA and Feeser HR. 1990. Functional implications of burst firing and single spike activity in lateral geniculate relay neurons. *Neuroscience* **39**(1), 103–113.
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Mel BW. 1994. Information processing in dendritic trees. *Neural Comput.* **6**(6), 1031–1085.
- Merolla P and Boahen K. 2004. A recurrent model of orientation maps with simple and complex cells. In *Advances in Neural Information Processing Systems 16 (NIPS)* (eds. Thrun S, Saul L, and Schölkopf B) MIT Press, Cambridge, MA. pp. 995–1002.
- Merolla PA, Arthur JV, Shi BE, and Boahen KA. 2007. Expandable networks for neuromorphic chips. *IEEE Trans. Circuits Syst. I: Regular Papers* **54**(2), 301–311.
- Mihalas S and Niebur E. 2009. A generalized linear integrate-and-fire neural model produces diverse spiking behavior. *Neural Comput.* **21**, 704–718.
- Millner S, Grübl A, Meier K, Schemmel J, and Schwartz MO. 2010. A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. In: *Advances in Neural Information Processing Systems 23 (NIPS)*. (eds Lafferty J, Williams CKI, Shawe-Taylor J, Zemel RS, and Culotta A). Neural Information Processing Systems Foundation, Inc., La Jolla, CA. pp. 1642–1650.
- Mitra S, Fusi S, and Indiveri G. 2009. Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Trans. Biomed. Circuits Syst.* **3**(1), 32–42.
- Nease S, George S, Hasler P, Kozoli S, and Brink S. 2012. Modeling and implementation of voltage-mode CMOS dendrites on a reconfigurable analog platform. *IEEE Trans. Biomed. Circuits Syst.* **6**(1), 76–84.
- Northmore DPM and Elias JG. 1998. Building silicon nervous systems with dendritic tree neuromorphs [Chapter 5]. In: *Pulsed Neural Networks* (eds. Maass W and Bishop CM). MIT Press, Cambridge, MA. pp. 135–156.
- Olsson JA and Häfliger P. 2008. Mismatch reduction with relative reset in integrate-and-fire photo-pixel array. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 277–280.
- Pelgrom MJM, Duinmaijer ACJ, and Welbers APG. 1989. Matching properties of MOS transistors. *IEEE J. Solid-State Circuits* **24**(5), 1433–1440.
- Rangan V, Ghosh A, Aparin V, and Cauwenberghs G. 2010. A subthreshold aVLSI implementation of the Izhikevich simple neuron model. In: *Proceedings of 32th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 4164–4167.
- Rasche C and Douglas RJ. 2001. Forward- and backpropagation in a silicon dendrite. *IEEE Trans. Neural Netw.* **12**(2), 386–393.

- Rastogi M, Garg V, and Harris JG. 2009. Low power integrate and fire circuit for data conversion. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2669–2672.
- Sarpeshkar R, Delbrück T, and Mead CA. 1993. White noise in MOS transistors and resistors. *IEEE Circuits Devices Mag.* **9**(6), 23–29.
- Schemmel J, Meier K, and Mueller E. 2004. A new VLSI model of neural microcircuits including spike time dependent plasticity. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)* **3**, pp. 1711–1716.
- Schemmel J, Brüderle D, Meier K, and Ostendorf B. 2007. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3367–3370.
- Schemmel J, Fieres J, and Meier K. 2008. Wafer-scale integration of analog neural networks. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, pp. 431–438.
- Serrano-Gotarredona R, Serrano-Gotarredona T, Acosta-Jimenez A, Serrano-Gotarredona C, Perez-Carrasco JA, Linares-Barranco A, Jimenez-Moreno G, Civit-Balcells A, and Linares-Barranco B. 2008. On real-time AER 2D convolutions hardware for neuromorphic spike based cortical processing. *IEEE Trans. Neural Netw.* **19**(7), 1196–1219.
- Serrano-Gotarredona T, Serrano-Gotarredona R, Acosta-Jimenez A, and Linares-Barranco B. 2006. A neuromorphic cortical-layer microchip for spike-based event processing vision systems. *IEEE Trans. Circuits Syst. I: Regular Papers* **53**(12), 2548–2566.
- Sheik S, Stefanini F, Neftci E, Chicca E, and Indiveri G. 2011. Systematic configuration and automatic tuning of neuromorphic systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 873–876.
- Silver R, Boahen K, Grillner S, Kopell N, and Olsen KL. 2007. Neurotech for neuroscience: unifying concepts, organizing principles, and emerging tools. *J. Neurosci.* **27**(44), 11807.
- Simoni MF, Cymbalyuk GS, Sorensen ME, Calabrese RL, and DeWeerth SP. 2004. A multiconductance silicon neuron with biologically matched dynamics. *IEEE Trans. Biomed. Eng.* **51**(2), 342–354.
- Toumazou C, Lidgey FJ, and Haigh DG. 1990. *Analogue IC Design: The Current-Mode Approach*. Peregrinus, Stevenage, UK.
- Toumazou C, Georgiou J, and Drakakis EM. 1998. Current-mode analogue circuit representation of Hodgkin and Huxley neuron equations. *IEE Electron. Lett.* **34**(14), 1376–1377.
- van Schaik A. 2001. Building blocks for electronic spiking neural networks. *Neural Netw.* **14**(6–7), 617–628.
- van Schaik A and Jin C. 2003. The tau-cell: a new method for the implementation of arbitrary differential equations. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **1**, pp. 569–572.
- van Schaik A, Jin C, McEwan A, and Hamilton TJ. 2010a. A log-domain implementation of the Izhikevich neuron model. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 4253–4256.
- van Schaik A, Jin C, McEwan A, Hamilton TJ, Mihalas S, and Niebur E 2010b. A log-domain implementation of the Mihalas-Niebur neuron model. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 4249–4252.
- Vogelstein RJ, Mallik U, Culurciello E, Cauwenberghs G, and Etienne-Cummings R. 2007. A multichip neuromorphic system for spike-based visual information processing. *Neural Comput.* **19**(9), 2281–2300.
- Wang YX and Liu SC. 2010. Multilayer processing of spatiotemporal spike patterns in a neuron with active dendrites. *Neural Comput.* **8**(22), 2086–2112.
- Wang YX and Liu SC. 2011. A two-dimensional configurable active silicon dendritic neuron array. *IEEE Trans. Circuits Syst. I* **58**(9), 2159–2171.
- Wang YX and Liu SC. 2013. Active processing of spatio-temporal input patterns in silicon dendrites. *IEEE Trans. Biomed. Circuits Syst.* **7**(3), 307–318.
- Wijekoon JHB and Dudek P. 2008. Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Netw.* **21**(2–3), 524–534.
- Wijekoon JHB and Dudek P. 2009. A CMOS circuit implementation of a spiking neuron with bursting and adaptation on a biological timescale. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 193–196.
- Yu T and Cauwenberghs G. 2010. Analog VLSI biophysical neurons and synapses with programmable membrane channel kinetics. *IEEE Trans. Biomed. Circuits Syst.* **4**(3), 139–148.

8

Silicon Synapses



Synapses form the connections between biological neurons and so can form connections between the silicon neurons described in Chapter 7. They are fundamental elements for computation and information transfer in both real and artificial neural systems. While modeling, the nonlinear properties and the dynamics of real synapses can be extremely onerous for software simulations, neuromorphic very large scale integration

The picture is an electron micrograph of mouse somatosensory cortex, arrowheads point to post-synaptic densities. Reproduced with permission of Nuno da Costa, John Anderson, and Kevan Martin.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

(VLSI) circuits efficiently reproduce synaptic dynamics in real-time. VLSI synapse circuits convert input spikes into analog charge which then produces post-synaptic currents that get integrated at the membrane capacitance of the post-synaptic neuron. This chapter discusses various circuit strategies used in implementing the temporal dynamics observed in real synapses. It also presents circuits that implement nonlinear effects with short-term dynamics, such as short-term depression and facilitation, as well as long-term dynamics such as spike-based learning mechanisms for updating the weight of a synapse for learning systems such as the ones discussed in Chapter 6 and the nonvolatile weight storage and update using floating-gate technology discussed in Chapter 10.

8.1 Introduction

The biological synapse is a complex molecular machine. There are two main classes of synapses: Electrical synapses and chemical synapses. The circuits in this chapter implement the latter class. A simplified picture of how the chemical synapse works is shown in Figure 8.1. In a chemical synapse, the pre-synaptic and post-synaptic membranes are separated by extracellular space called a synaptic cleft. The arrival of a pre-synaptic action potential triggers the influx of Ca^{2+} , which then leads to the release of neurotransmitter into the synaptic cleft. These neurotransmitter molecules (e.g., AMPA, GABA) bind to receptors on the post-synaptic side. These receptors consist of membrane channels with two major classes: ionic ligand-gated membrane channels such as AMPA channels with ions like Na^+ and K^+ and ionic ligand-gated and voltage-gated channels such as NMDA channels. These channels can be excitatory or inhibitory, that is the post-synaptic current can charge or discharge the membrane. The typical receptors in the cortex are AMPA and NMDA receptors which are excitatory and GABA receptors which are inhibitory. The post-synaptic currents produce a change in the post-synaptic potential and when this potential exceeds a threshold at the cell body of the neuron generates an action potential.

A framework in which to describe the neurotransmitter kinetics of the synapse was proposed by Destexhe et al. (1998). Using this approach, one can synthesize equations for a complete description of the synaptic transmission process. For a two-state ligand-gated channel model, the neurotransmitter molecules T bind to the post-synaptic receptors modeled by the first-order kinetic scheme:



where R and TR^* are the unbound and bound forms of the post-synaptic receptor, respectively, α and β are the forward and backward rate constants for transmitter binding, respectively. The fraction of bound receptors r is described by

$$\frac{dr}{dt} = \alpha[T](1 - r) - \beta r. \quad (8.2)$$

If T is modeled as a short pulse, then the dynamics of r can be described by a first-order differential equation.

If the binding of the transmitter to a post-synaptic receptor directly gates the opening of an associated ion channel, the total conductance through all channels of the synapse is r multiplied

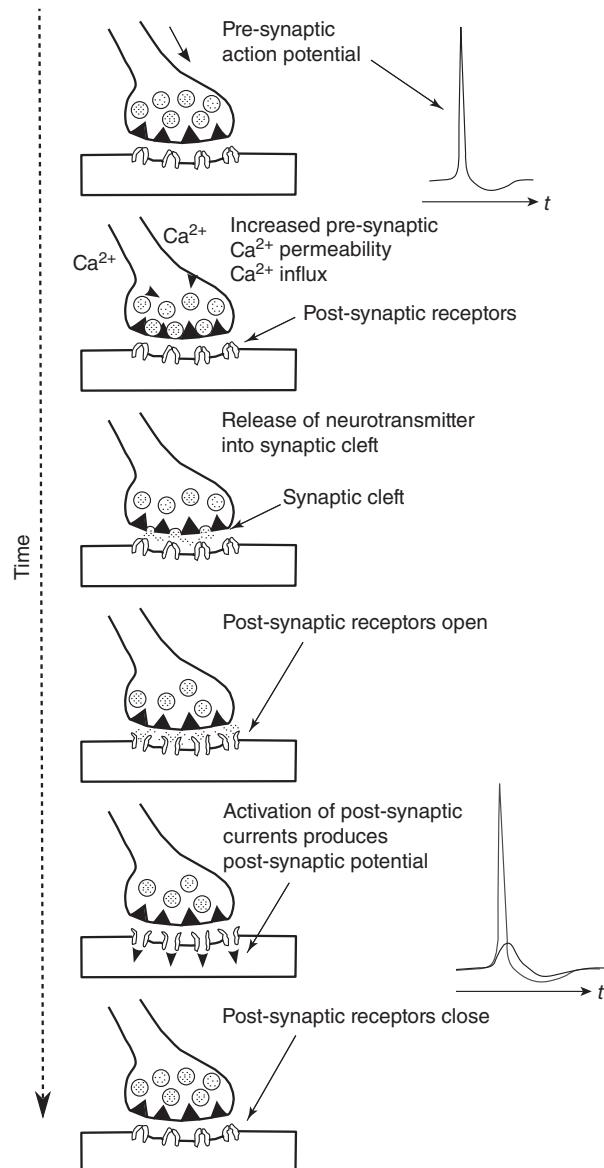


Figure 8.1 Synaptic transmission process

by the maximal conductance of the synapse \bar{g}_{syn} . Response saturation occurs naturally as r approaches one (all channels are open). The synaptic current I_{syn} is then given by

$$I_{\text{syn}}(t) = \bar{g}_{\text{syn}} r(t)(V_{\text{syn}}(t) - E_{\text{syn}}), \quad (8.3)$$

where V_{syn} is the post-synaptic potential and E_{syn} is the synaptic reversal potential.

Table 8.1 Main synapse computational blocks (left), and circuit design strategies (right)

Computational blocks		Design styles
Pulse dynamics	Subthreshold	Above threshold
Fall-time dynamics	Voltage mode	Current mode
Rise- and fall-time dynamics	Asynchronous	Switched capacitor
Short-term plasticity	Biophysical model	Phenomenological model
Long-term plasticity (learning)	Real time	Accelerated time

8.2 Silicon Synapse Implementations

In the silicon implementation, the conductance-based Eq. (8.3) is often simplified to a current that does not depend on the difference between the membrane potential and the synaptic reversal potential because the linear dependence of a current on the voltage difference across a transistor is only valid in a small voltage range of a few tenths of a volt in a subthreshold complementary metal oxide semiconductor (CMOS) circuit. Reduced synaptic models describe the post-synaptic current as a step increase in current when an input spike arrives and this current decays away exponentially when the pulse disappears.

Table 8.1 summarizes the relevant computational sub-blocks for building silicon synapses and the possible design styles that can be used to implement them. Each computational block from the first column can be implemented with circuits that adopt any of the design strategies outlined in the other column. Section 8.2.1 describes some of the more common circuits used as basic building blocks as outlined in Table 8.1.

In the synaptic circuits described in this chapter, the input is a short-duration pulse (order of μs) approximating a spike input and the synaptic output current charges the membrane capacitance of a neuron circuit.

8.2.1 Non Conductance-Based Circuits

Synaptic circuits usually implement reduced synaptic models, in part, because complex models would require many tens of transistors which would lead to a reduction in the proportional number of synapses that can be implemented on-chip. The earlier silicon synapse circuits did not usually incorporate short-term and long-term plasticity dynamics.

Pulse Current-Source Synapse

The pulsed current-source synapse is one of the simplest synaptic circuits (Mead 1989). As shown in Figure 8.2a, the circuits consist of a voltage-controlled subthreshold current source transistor in series with a switching transistor activated by an active-low input spike. The activation is controlled by the input spike (digital pulse between V_{dd} and Gnd) going to the gate of the M_{pre} transistor. When an input spike arrives, the M_{pre} transistor is switched on, and the output current I_{syn} is a current that is set by V_w (synaptic weight) of the M_w transistor. This current lasts as long as the input spike is low. Assuming that the current source transistor

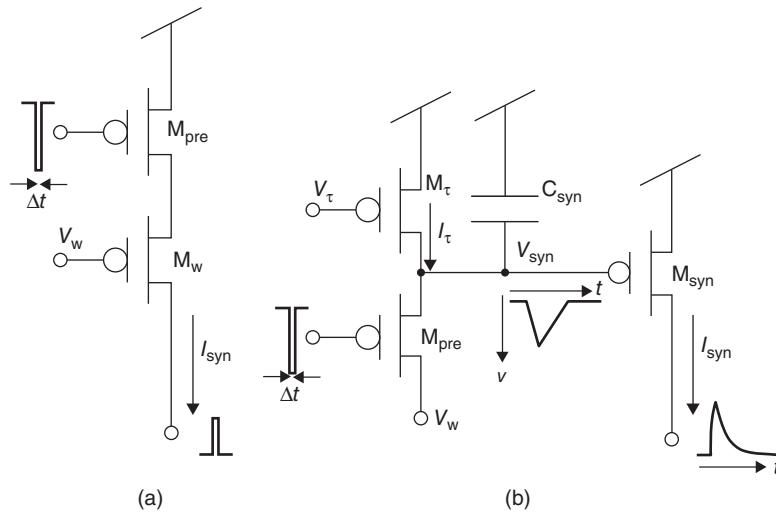


Figure 8.2 (a) Pulsed current synapse circuit and (b) reset-and-discharge circuit

M_w is in saturation and the current is subthreshold, the synaptic current I_{syn} can be expressed as follows:

$$I_{syn}(t) = I_0 \exp\left(-\frac{\kappa}{U_T}(V_w - V_{dd})\right), \quad (8.4)$$

where V_{dd} is the power supply voltage, I_0 is the leakage current, κ is the subthreshold slope factor, and U_T the thermal voltage (Liu et al. 2002). The post-synaptic membrane potential undergoes a step voltage change in proportion to $I_{syn}\Delta t$, where Δt is the pulse width of the input spike.

This circuit is compact but does not incorporate any dynamics of the current. The temporal interval between input spikes does not affect the post-synaptic integration assuming that the membrane does not have a time constant; hence, input spike trains with the same mean rates but different spike timing distributions lead to the same output neuronal firing rates. However, because of its simplicity and area compactness, this circuit has been used in a wide variety of VLSI implementations of pulse-based neural networks that use mean firing rates as the neural code (Chicca et al. 2003; Fusi et al. 2000; Murray 1998).

Reset-and-Discharge Synapse

This circuit can be extended to include simple dynamics by extending the duration of the output current I_{syn} past the input spike duration (Lazzaro et al. 1994). This synapse circuit (Figure 8.2b) comprises three pFET transistors and one capacitor; M_{pre} acts as a digital switch that is controlled by the input spike; M_τ is operated as a constant subthreshold current source to

recharge the capacitor C_{syn} ; M_{syn} generates an output current that is exponentially dependent on the V_{syn} node (assuming subthreshold operation and saturation of M_{syn}):

$$I_{\text{syn}}(t) = I_0 \exp \left(-\frac{\kappa}{U_{\text{T}}} (V_{\text{syn}}(t) - V_{\text{dd}}) \right). \quad (8.5)$$

When an active-low input spike (pulse goes from V_{dd} to Gnd) arrives at the gate of the M_{pre} transistor, this transistor is switched on and the V_{syn} node is shorted to the weight bias voltage V_w . When the input pulse ends (digital pulse goes from Gnd to V_{dd}), M_{pre} is switched off. Since I_τ is a constant current, V_{syn} charges linearly back to V_{dd} at a rate set by I_τ/C_{syn} . The output current is

$$I_{\text{syn}}(t) = I_{w0} \exp \left(-\frac{t}{\tau} \right), \quad (8.6)$$

where $I_{w0} = I_0 e^{-\frac{\kappa}{U_{\text{T}}} (V_w - V_{\text{dd}})}$ and $\tau = \frac{\kappa I_\tau}{U_{\text{T}} C_{\text{syn}}}$.

Although this synaptic circuit produces an excitatory post-synaptic current (EPSC) that lasts longer than the duration of an input pulse and decays exponentially with time, it sums nonlinearly the contribution of all input spikes. If an input spike arrives while V_{syn} is still charging back to V_{dd} , its voltage will be reset back to V_w , thus the remaining charge contribution from the last spike will be eliminated. So, given a generic spike sequence of n spikes, $\rho(t) = \sum_i^n \delta(t - t_i)$ the response of the reset-and-discharge synapse will be $I_{\text{syn}}(t) = I_{w0} \exp(-\frac{t-t_n}{\tau})$. Although this circuit produces an EPSC that lasts longer than the duration of its input spikes, its response depends mostly on the last (n th) input spike. This nonlinear property of the circuit fails to reproduce the linear summation property of post-synaptic currents often desired in synaptic models.

Linear Charge-and-Discharge Synapse

Figure 8.3a shows a modification of the reset-and-discharge synapse that has often been used in the neuromorphic engineering community (Arthur and Boahen 2004). The linear charge-and-discharge synapse circuit allows for both a rise time and fall time of the output current. The circuit operation is as follows: the input pre-synaptic pulse applied to the nFET M_{pre} is highly active. The following analysis is based on the assumption that all transistors are in saturation and operate in subthreshold. During an input pulse, the node $V_{\text{syn}}(t)$ decreases linearly at a rate set by the net current $I_w - I_\tau$, and the output current $I_{\text{syn}}(t)$ can be described as

$$I_{\text{syn}}(t) = \begin{cases} I_{\text{syn}}^- \exp \left(\frac{t-t_i^-}{\tau_c} \right) & (\text{charge phase}) \\ I_{\text{syn}}^+ \exp \left(-\frac{t-t_i^+}{\tau_d} \right) & (\text{discharge phase}), \end{cases} \quad (8.7)$$

where t_i^- is the time at which the i th input spike arrives, t_i^+ is the time right after the spike, I_{syn}^- is the output current at t_i^- , I_{syn}^+ is the output current at t_i^+ , $\tau_c = \frac{U_{\text{T}} C_{\text{syn}}}{\kappa (I_w - I_\tau)}$, and $\tau_d = \frac{U_{\text{T}} C_{\text{syn}}}{\kappa I_\tau}$. The dynamics of the output current are exponential.

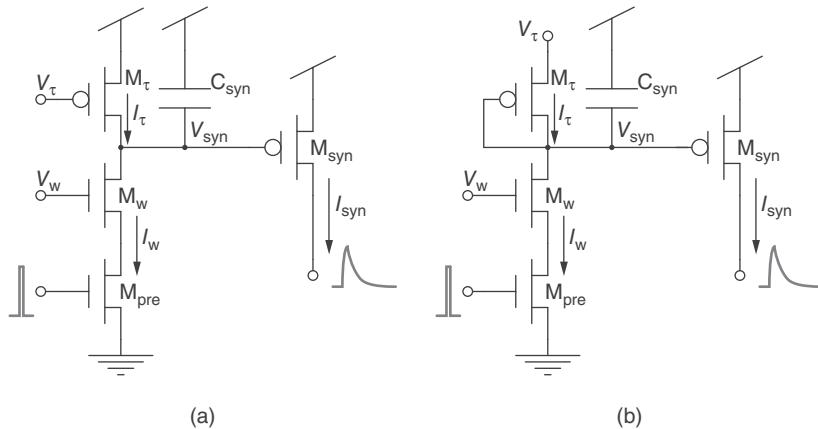


Figure 8.3 (a) Linear charge-and-discharge synapse and (b) current-mirror integrator (CMI) synapse

Assuming that each spike lasts a fixed brief period Δt , and that two successive spikes that arrive at times t_i and t_{i+1} , we can then write

$$I_{\text{syn}}(t_{i+1}^-) = I_{\text{syn}}(t_i^-) \exp \left(\Delta t \left(\frac{1}{\tau_c} + \frac{1}{\tau_d} \right) \right) \exp \left(-\frac{t_{i+1}^- - t_i^-}{\tau_d} \right). \quad (8.8)$$

From this recursive equation, we derive the response of the linear charge-and-discharge synapse to a generic spike sequence $\rho(t)$ assuming the initial condition $V_{\text{syn}}(0) = V_{\text{dd}}$. If we denote the input spike train frequency at time t as f , we can express Eq. (8.8) as

$$I_{\text{syn}}(t) = I_0 \exp \left(-\frac{\tau_c - f \Delta t (\tau_c + \tau_d)}{\tau_c \tau_d} \right). \quad (8.9)$$

The major drawback of this circuit, aside from it not being a linear integrator, is that V_{syn} is a high impedance node; so if the input frequency is too high (i.e., if $f > \frac{1}{\Delta t} \frac{I_x}{I_w}$ in Eq. (8.9)), $V_{\text{syn}}(t)$ can drop to arbitrarily low values, even all the way down to Gnd. Under these conditions, the circuit's steady-state response will not encode the input frequency. The input charge dynamics is also sensitive to the jitter in the input pulse width like all the preceding synapses.

Current-Mirror Integrator Synapse

By replacing the M_τ transistor (Figure 8.3b) with a diode-connected transistor, this modified circuit by Boahen (1997) has a dramatically different synaptic response. The two transistors M_τ and M_{syn} implement a current mirror, and together with the capacitor C_{syn} , they form a current-mirror integrator (CMI). This nonlinear integrator circuit produces a mean output current I_{syn} that increases with input firing rates and has a saturating nonlinearity with a maximum amplitude that depends on the circuit parameters: the weight, V_w , and the time constant bias, V_τ . The CMI responses have been derived analytically for steady-state conditions

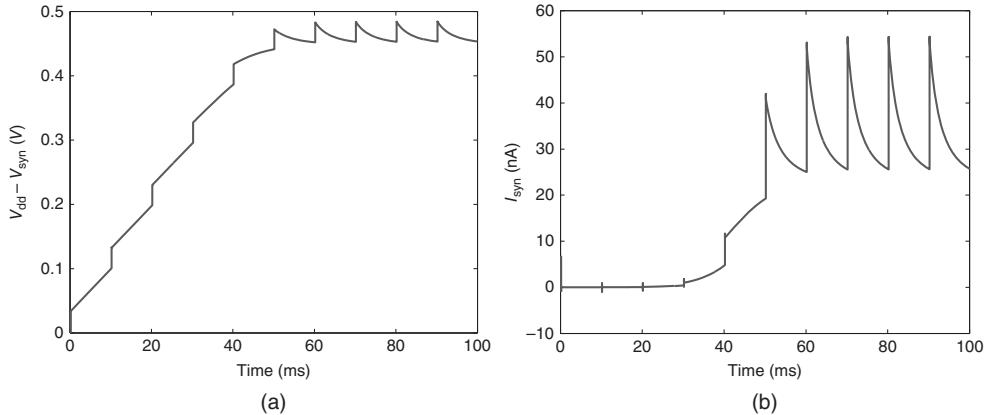


Figure 8.4 (a) CMI voltage output and (b) synaptic current output in response to a 100-Hz input spike train

(Hynna and Boahen 2001) and also for a generic spike train (Chicca 2006). The output current in response to an input spike is

$$I_{syn}(t) = \begin{cases} \frac{\alpha I_w}{1 + (\frac{\alpha I_w}{I_{syn}} - 1) \exp(-\frac{t-t_i^-}{\tau_c})} & \text{(charge phase)} \\ \frac{I_w}{\frac{I_w}{I_{syn}} + \frac{t-t_i^-}{\tau_d}} & \text{(discharge phase),} \end{cases} \quad (8.10)$$

where $t_i^-, t_i^+, I_{syn}^-, I_{syn}^+$ are previously defined in Eq. (8.7), $\alpha = e^{\frac{V_\tau - V_{dd}}{U_T}}$, $\tau_c = \frac{C_{syn} U_T}{\kappa I_w}$, and $\tau_d = \alpha \tau_c$. A simulation of the voltage (V_{syn}) and current (I_{syn}) responses from this circuit are shown in Figure 8.4.

During the charging phase, the EPSC increases over time as a sigmoidal function, while during the discharge phase it decreases with a $1/t$ profile. The discharge of the EPSC is therefore fast compared to the typical exponential decay profiles of other synaptic circuits. The parameter α (set by the V_τ source bias voltage) can be used to slow the EPSC decay. However, this parameter affects the duration of the EPSC discharge profile, the maximum amplitude of the EPSC charge phase, and the DC synaptic current; thus, longer response times (larger values of τ_d) produce higher EPSC values.

Despite these nonlinearities and the fact that the CMI cannot produce linear summation of post-synaptic currents, this compact circuit was extensively used by the neuromorphic engineering community (Boahen 1998; Horiuchi and Hynna 2001; Indiveri 2000; Liu et al. 2001).

Summarizing Exponential Synapse

The reset-and-discharge circuit was modified by Shi and Horiuchi (2004) so that the charge contribution from multiple input spikes can be summed together linearly. This summarizing synapse subthreshold circuit is shown in Figure 8.5. Input voltage spikes are applied to the

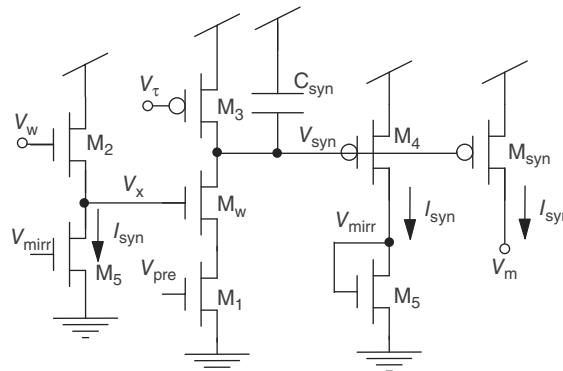


Figure 8.5 Summating synapse circuit. Input spikes are applied to V_{pre} . There are two control parameters. The voltage V_w adjusts the weight of the synapse indirectly through a source-follower circuit and V_τ sets the time constant. To reduce the body effect in the n-type source follower, all transistors can be converted to the opposite polarity

gate V_{pre} of the transistor M_1 . Here, the gate voltage V_τ sets the time constant of the synapse as in the other synapse circuits. V_w determines the synaptic weight indirectly through a source follower acting as a level shifter. The bias current in the source follower is set by a copy of the output synaptic current I_{syn} . When multiple input spikes arrive at M_1 , the increasing I_{syn} reduces the value of V_x through the source follower. By applying the translinear principle discussed in Section 8.2.1,

$$\frac{d}{dt}I_{\text{syn}} + \frac{\kappa I_\tau}{CU_T}I_{\text{syn}} = \frac{M\kappa_n}{CU_T} \exp(\kappa V_w/U_T). \quad (8.11)$$

Log-Domain Synapse

This section covers synaptic circuits which are based on the log-domain, current-mode filter design methodology proposed by Frey (1993 1996); Seevinck (1990). These filter circuits exploit the logarithmic relationship between a subthreshold metal oxide semiconductor field effect transistor (MOSFET) gate-to-source voltage and its channel current and is therefore called a log-domain filter. Even though the original circuits from Frey were based on bipolar transistors, the method of constructing such filters can be extended to MOS transistors operating in subthreshold.

The exponential relationship between the current and the gate-to-source voltage means that we can use the translinear principle first introduced by Gilbert (1975) to solve for the current relationships in a circuit or to construct circuits with a particular input–output transfer function. This principle is also outlined in Section 9.3.1 in Chapter 9. By considering a set of transistors connected in a closed-loop form, the following equation satisfies the voltages in the clockwise (CW) direction versus the voltages in the counter-clockwise (CCW) direction:

$$\sum_{n \in \text{CCW}} V_n = \sum_{n \in \text{CW}} V_n. \quad (8.12)$$

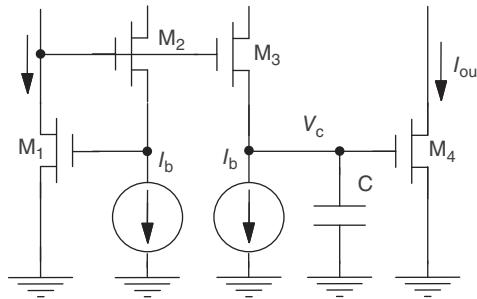


Figure 8.6 Log-domain integrator circuit

By substituting for I in Eq. (8.12) through the subthreshold exponential I–V relationship,

$$\prod_{n \in \text{CCW}} I_n = \prod_{n \in \text{CW}} I_n. \quad (8.13)$$

We can apply the translinear principle on the currents in the low-pass filter circuit in Figure 8.6:

$$I_{\text{ds}}^{\text{M}1} \cdot I_{\text{ds}}^{\text{M}2} = I_{\text{ds}}^{\text{M}3} \cdot I_{\text{ds}}^{\text{M}4}$$

$$I_{\text{ds}}^{\text{M}3} = I_{\text{b}} + C \frac{d}{dt} V_{\text{c}}$$

$$\frac{d}{dt} I_{\text{out}} = \frac{C U_{\text{T}}}{U_{\text{T}}} I_{\text{out}} \frac{d}{dt} V_{\text{c}}$$

$$I_{\text{in}} \cdot I_{\text{b}} = \left(I_{\text{b}} + \frac{C U_{\text{T}}}{\kappa I_{\text{out}}} \frac{d}{dt} I_{\text{out}} \right) \cdot I_{\text{out}}$$

leading to the following first-order differential equation between the input current I_{in} and output current I_{out} :

$$\tau \frac{d}{dt} I_{\text{out}} + I_{\text{out}} = I_{\text{in}}, \quad (8.14)$$

where $\tau = \frac{C U_{\text{T}}}{\kappa I_{\text{b}}}$.

Log-Domain Integrator Synapse

Unlike the log-domain integrator in Figure 8.6, the log-domain integrator synapse shown in Figure 8.7 has an input current that is activated only during an input spike (Merolla and Boahen 2004). The output current I_{syn} has the same exponential dependence on its gate voltage V_{syn}

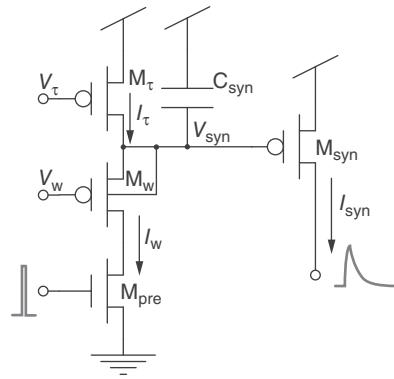


Figure 8.7 Log-domain integrator synapse circuit

as all other synaptic circuits (see Eq. (8.5)). The derivative of this current with respect to time is given by

$$\begin{aligned} \frac{d}{dt}I_{\text{syn}} &= \frac{dI_{\text{syn}}}{dV_{\text{syn}}} \frac{d}{dt}V_{\text{syn}} \\ &= -I_{\text{syn}} \frac{\kappa}{U_{\text{T}}} \frac{d}{dt}V_{\text{syn}}. \end{aligned} \quad (8.15)$$

During an input spike (charge phase), the dynamics of the V_{syn} are governed by the equation: $C_{\text{syn}} \frac{d}{dt}V_{\text{syn}} = -(I_w - I_\tau)$. Combining this first-order differential equation with Eq. (8.15), we obtain

$$\tau \frac{d}{dt}I_{\text{syn}} + I_{\text{syn}} = I_{\text{syn}} \frac{I_w}{I_\tau}, \quad (8.16)$$

where $\tau = \frac{C_{\text{syn}} U_{\text{T}}}{\kappa I_\tau}$. The beauty of this circuit lies in the fact that the term I_w is inversely proportional to I_{syn} itself:

$$I_w = I_0 \exp \left(-\frac{\kappa(V_w - V_{\text{syn}})}{U_{\text{T}}} \right) = I_{w0} \frac{I_0}{I_{\text{syn}}}, \quad (8.17)$$

where I_0 is the leakage current and I_{w0} is the current flowing through M_w in the initial condition, when $V_{\text{syn}} = V_{\text{dd}}$. When this expression of I_w is substituted in Eq. (8.16), the right term of the differential equation loses the I_{syn} dependence and becomes the constant $\frac{I_0 I_{w0}}{I_\tau}$. Therefore, the log-domain integrator function takes the form of a canonical first-order, low-pass filter equation, and its response to a spike arriving at t_i^- and ending at t_i^+ is

$$I_{\text{syn}}(t) = \begin{cases} \frac{I_0 I_{w0}}{I_\tau} \left(1 - \exp \left(-\frac{t - t_i^-}{\tau} \right) \right) + I_{\text{syn}}^- \exp \left(-\frac{t - t_i^-}{\tau} \right) & (\text{charge phase}) \\ I_{\text{syn}}^+ \exp \left(-\frac{t - t_i^+}{\tau} \right) & (\text{discharge phase}). \end{cases} \quad (8.18)$$

This is the only synaptic circuit of the ones described so far that has linear filtering properties. The same silicon synapse can be used to sum the contributions of spikes potentially arriving from different sources in a linear way. This could save significant amounts of silicon real estate in neural architectures where the synapses do not implement learning or local adaptation mechanisms and could therefore solve the chip area problem that have hindered the development of large-scale VLSI multineuron chips in the past.

However, the log-domain synapse circuit has two problems. The first problem is that the VLSI layout of the schematic shown in Figure 8.7 requires more area than the layout of other synaptic circuits, because the pFET M_w needs its own isolated well. The second, and more serious, problem is that the spike lengths used in pulse-based neural network systems, which typically last less than a few microseconds, are too short to inject enough charge in the membrane capacitor of the post-synaptic neuron to see any effect. The maximum amount of charge possible is $\Delta Q = \frac{I_w l_{w0}}{I_\tau} \Delta t$ and I_{w0} cannot be increased beyond subthreshold current limits (of the order of nano-amperes); otherwise, the log-domain properties of the filter break down (note that I_τ is also fixed, to set the desired time constant τ). A possible solution is to increase the fast (off-chip) input pulse lengths using on-chip pulse extender circuits. This solution however requires additional circuitry at each input synapse and makes the layout of the overall circuit even larger (Merolla and Boahen 2004).

Diff-Pair Integrator Synapse

By replacing the isolated pFET M_w with three nFETs, the Diff-Pair Integrator (DPI) synapse circuit (Bartolozzi and Indiveri 2007) solves the problems of the log-domain integrator synapse while maintaining its linear filtering properties, thus preserving the possibility of multiplexing in time spikes arriving from different sources. The schematic diagram of the DPI synapse is shown in Figure 8.8. This circuit comprises four nFETs, two pFETs, and a capacitor. The nFETs form a differential pair whose branch current I_{in} represents the input to the synapse

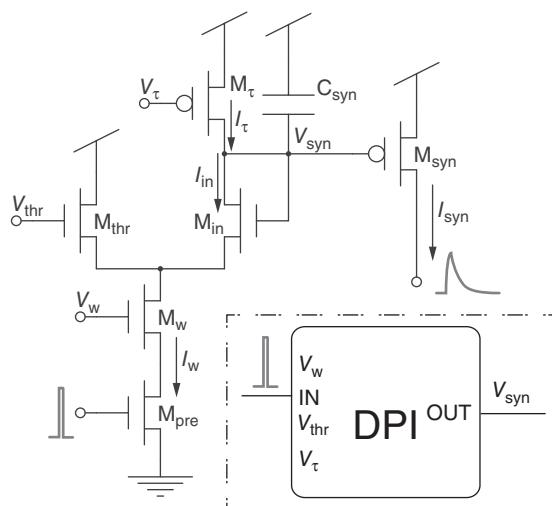


Figure 8.8 Diff-pair integrator synapse circuit

during the charging phase. Assuming both subthreshold operation and saturation regime, I_{in} can be expressed as

$$I_{\text{in}} = I_w \frac{e^{\frac{\kappa V_{\text{syn}}}{U_T}}}{e^{\frac{\kappa V_{\text{syn}}}{U_T}} + e^{\frac{\kappa V_{\text{thr}}}{U_T}}} \quad (8.19)$$

and by multiplying the numerator and denominator of Eq. (8.19) by $e^{-\frac{\kappa V_{\text{dd}}}{U_T}}$, we can re-express I_{in} as

$$I_{\text{in}} = \frac{I_w}{1 + \left(\frac{I_{\text{syn}}}{I_{\text{gain}}} \right)}, \quad (8.20)$$

where the term $I_{\text{gain}} = I_0 e^{-\frac{\kappa(V_{\text{thr}} - V_{\text{dd}})}{U_T}}$ represents a virtual p-type subthreshold current that is not tied to any pFET in the circuit.

As for the log-domain integrator, we can combine the C_{syn} capacitor equation $C_{\text{syn}} \frac{d}{dt} V_{\text{syn}} = -(I_{\text{in}} - I_{\tau})$ with Eq. (8.5) and write

$$\tau \frac{d}{dt} I_{\text{syn}} = -I_{\text{syn}} \left(1 - \frac{I_{\text{in}}}{I_{\tau}} \right), \quad (8.21)$$

where $\tau = \frac{C_{\text{syn}} U_T}{\kappa I_{\tau}}$. Replacing I_{in} from Eq. (8.20) into Eq. (8.21), we obtain

$$\tau \frac{d}{dt} I_{\text{syn}} + I_{\text{syn}} = \frac{I_w}{I_{\tau}} \frac{I_{\text{syn}}}{1 + \left(\frac{I_{\text{syn}}}{I_{\text{gain}}} \right)}. \quad (8.22)$$

This is a first-order nonlinear differential equation; however, the steady-state condition can be solved in closed form and its solution is

$$I_{\text{syn}} = I_{\text{gain}} \left(\frac{I_w}{I_{\tau}} - 1 \right). \quad (8.23)$$

If $I_w \gg I_{\tau}$, the output current I_{syn} will eventually rise to values such that $I_{\text{syn}} \gg I_{\text{gain}}$, when the circuit is stimulated with an input step signal. If $\frac{I_{\text{syn}}}{I_{\text{gain}}} \gg 1$, the I_{syn} dependence in the second term of Eq. (8.22) cancels out, and the nonlinear differential equation simplifies to the canonical first-order, low-pass filter equation:

$$\tau \frac{d}{dt} I_{\text{syn}} + I_{\text{syn}} = I_{\text{gain}} \frac{I_w}{I_{\tau}}. \quad (8.24)$$

In this case, the output current response of the circuit to a spike is

$$I_{\text{syn}} = \begin{cases} I_{\text{gain}} \frac{I_w}{I_{\tau}} \left(1 - \exp -\frac{(t-t_i^-)}{\tau} \right) & (\text{charge phase}) \\ I_{\text{syn}}^+ \exp -\frac{(t-t_i^+)}{\tau} & (\text{discharge phase}). \end{cases} \quad (8.25)$$

The dynamics of this circuit is almost identical to the log-domain integrator synapse circuit with the difference that the term I_w/I_τ is multiplied by I_{gain} here, rather than the I_0 of the log-domain integrator solution. This gain term can be used to amplify the charge phase response amplitude, therefore solving the problem of generating sufficiently large charge packets sourced into the neuron's integrating capacitor for input spikes of very brief duration, while keeping all currents in the subthreshold regime and without requiring additional pulse-extender circuits. An additional advantage of this circuit over the log-domain one is given by the fact that the layout of the circuit does not require isolated well structures for the pFETs.

Although this circuit is less compact than many synaptic circuits, it is the only one that can reproduce the exponential dynamics observed in the post-synaptic currents of biological synapses, without requiring additional input pulse-extender circuits. Moreover, the circuit has independent control of time constant, synaptic weight, and synaptic scaling parameters. The extra degree of freedom obtained with the V_{thr} parameter can be used to globally scale the efficacies of the DPI circuits that share the same V_{thr} bias. This feature could in turn be employed to implement global homeostatic plasticity mechanisms complementary to local spike-based plasticity ones acting on the synaptic weight node V_w .

8.2.2 Conductance-Based Circuits

In the synaptic circuits described until now, the post-synaptic current I_{syn} is almost independent of the membrane voltage, V_m because the output of the synapse comes from the drain node of a transistor in saturation. A more detailed synaptic model that includes the dependence of the ionic current on V_m is called the conductance-based synapse. The dynamics of V_m corresponding to currents that capture this dependence is described as follows:

$$C_m \frac{dV_m}{dt} = g_1(t)(E_1 - V_m) + g_2(t)(E_2 - V_m) + g_{\text{leak}}(E_{\text{rest}} - V_m), \quad (8.26)$$

where C_m is the membrane capacitance, V_m is the membrane potential, $g_i(t)$ is a time-varying synaptic conductance, E_i is a synaptic reversal potential corresponding to the particular ionic current (e.g., Na^+ , K^+), and each of the terms on the right corresponds to a particular ionic current.

Single-Transistor Synapse

A transistor, when operated in the ohmic region, can produce a post-synaptic current that is dependent on the difference between the membrane potential and the reversal potential. This relationship is approximately linear for a transistor operating in subthreshold and for drain-to-source voltages (V_{ds}) which are less than approximately $4kT/q$. Circuits that implement the conductance-based dependencies by operating the transistor under these conditions include the single-transistor synapses described by Diorio et al. (1996 1998), and the Hodgkin-Huxley circuits by Hynna and Boahen (2007) and Farquhar and Hasler (2005). By operating the transistors in strong inversion, the V_{ds} range over which the current grows linearly is a function of the gate-source voltage, V_{gs} . The use of the transistor in this region is implemented in the above-threshold multineuron arrays (Schemmel et al. 2007 2008).

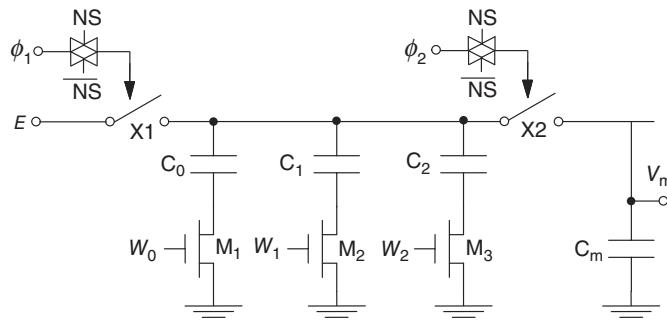


Figure 8.9 Single compartment switched-capacitor conductance-based synapse circuit. When the address of an incoming event is decoded, row- and column-select circuitry activate the cells neuron select (NS) line. The nonoverlapping clocks, ϕ_1 and ϕ_2 , control the switching and are activated by the input event. © 2007 IEEE. Reprinted, with permission, from Vogelstein et al. (2007)

Switched-Capacitor Synapse

The switched-capacitor circuit in Figure 8.9 implements a discrete-time version of the conductance-based membrane equation. The dynamics for synapse i is

$$C_m \frac{\Delta V_m}{\Delta T} = g_i(E_i - V_m). \quad (8.27)$$

The conductance, $g_i = g$ is discrete in amplitude. The amount of charge transferred depends on which of the three geometrically sized synaptic weight capacitors (C_0 to C_2) are active. These elements are dynamically switched on and off by binary control voltages W_0 – W_2 on the gates of transistors M_1 to M_3 . The binary coefficients W_0 – W_2 provide eight possible conductance values. A larger dynamic range in effective conductance can be accommodated by modulating the number and probability of synaptic events according to the synaptic conductance $g = npq$ (Koch 1999). That is, the conductance is proportional to the product of the number of synaptic release sites on the pre-synaptic neuron (n), the probability of synaptic release (p), and the post-synaptic response to a quantal amount of neurotransmitter (q). By sequentially activating switches $X1$ and $X2$, a packet of charge proportional to the difference between the externally supplied and dynamically modulated reversal potential E and V_m is added to (or subtracted from) the membrane capacitor C_m .

Figure 8.10 illustrates the response of a conductance-based neuron as it receives a sequence of events through the switched-capacitor synapse while both the synaptic reversal potential and the synaptic weight are varied (Vogelstein et al. 2007). This neuron is part of an array used in the integrate-and-fire array transceiver (IFAT) system discussed in Section 13.3.1. The membrane voltage reveals the strong impact of the order of events on the neural output due to the operation of the conductance-based model, as opposed to a standard I&F model. This circuit can also implement ‘shunting inhibition’ usually observed in Cl^- channels. This effect happens when the synaptic reversal potential is close to the resting potential of the neuron. Shunting inhibition results from the nonlinear interaction of neural activity gated by such a synapse, an effect that is unique to the conductance-based model and missing in both standard and leaky I&F models.

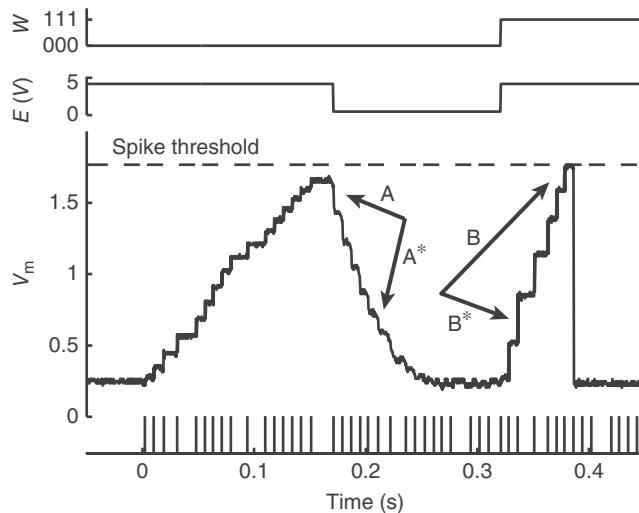


Figure 8.10 Responses from a neuron within an array of conductance-based synapses and neurons. The lower trace illustrates the membrane potential (V_m) of the neuron as a series of events sent at times marked by vertical lines at the bottom of the figure. The synaptic reversal potential (E) and synaptic weight (W) are drawn in the top two traces. An inhibitory event following a sequence of excitatory events has a greater impact on the membrane potential than the same inhibitory event following many inhibitory events (compare events indicated by arrows A versus A^*) and vice versa (compare events indicated by arrows B versus B^*). © 2007 IEEE. Reprinted, with permission, from Vogelstein et al. (2007)

The synaptic circuit in Figure 8.9 does not have any temporal dynamics. Extensions to this design (Yu and Cauwenberghs 2010) give this circuit the synaptic dynamics.

8.2.3 NMDA Synapse

The NMDA receptors form another important class of synaptic channels. These receptors allow ions to flow through the channels only if the membrane voltage is depolarized above a given threshold while in the presence of their neurotransmitter (glutamate). A circuit that demonstrates this property by exploiting the thresholding property of the differential pair circuit is shown in Figure 8.11 (Rasche and Douglas 2001). If the membrane node V_m is lower than an external bias V_{ref} , the output current I_{syn} flows through the transistor M_4 in the left branch of the diff-pair and has no effect on the post-synaptic depolarization. On the other hand, if V_m is higher than V_{ref} , the current flows also into the membrane potential node, depolarizing the neuron, and thus implementing the voltage-gating typical of NMDA synapses.

This circuit can reproduce both the voltage-gated and temporal dynamic properties of real NMDA synapses. It may be important to be able to implement these properties in our VLSI devices because there is evidence that they play a crucial role in detecting coincidence between the pre-synaptic activity and post-synaptic depolarization for inducing long-term potentiation (LTP) (Morris et al. 1990). Furthermore, the NMDA synapse could be useful for

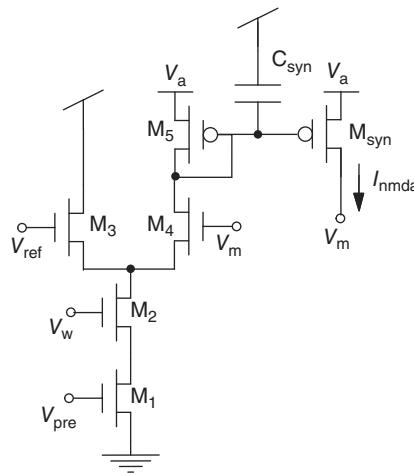


Figure 8.11 NMDA synapse circuit. V_m is the post-synaptic membrane voltage. The transistors, M_3 and M_4 , of the differential pair circuit emulate the voltage dependence of the current. Input spike that goes to V_{pre} and V_w sets the weight of the synapse

stabilizing persistent activity of recurrent VLSI networks of spiking neurons, as proposed by computational studies within the context of working memory (Wang 1999).

8.3 Dynamic Plastic Synapses

The synapse circuits discussed so far have a weight that is specified by a fixed value during operation. In many biological synapses, experiments show that the effective conductance of the membrane channels in the synapse change dynamically. This change in the conductance is dependent on the pre-synaptic activity and/or the post-synaptic activity. Short-term dynamic synapse states are determined by their pre-synaptic activity and have short time constants from around tens to hundreds of milliseconds. Long time-constant plastic synapses have weights that are modified by both the pre- and post-synaptic activity. Their time constants can be in the hundreds of seconds to hours, days, and years. Their functional roles are usually explored in models of learning. The next sections describe synapse circuits where the weight of the synapse changes following either short-term or long-term plasticity.

8.3.1 Short-Term Plasticity

Dynamic synapses can be depressing, facilitating, or a combination of both. In a depressing synapse, the synaptic strength decreases after each spike and recovers to its maximal value with a time constant τ_d . This way, a depressing synapse is analogous to a high-pass filter. It codes primarily changes in the input rather than the absolute level of the input. The output current of the depressing synapse codes primarily changes in the pre-synaptic frequency. The

steady-state current amplitude is approximately inversely dependent on the input frequency. In facilitating synapses, the strength increases after each spike and recovers to its minimum value with a time constant τ_f . It acts like a nonlinear low-pass filter to changes in spike rates. A step increase in pre-synaptic firing rate leads to a gradual increase in the synaptic strength.

Both types of synapses can be treated as time-invariant fading memory filters (Maass and Sontag 2000). Two prevalent models that are used in network simulations and also for fitting physiological data are the phenomenological models by Markram et al. (1997); Tsodyks and Markram (1997); Tsodyks et al. (1998) and the models from Abbott et al. (1997) and Varela et al. (1997).

In the model from Abbott et al. (1997), the depression in the synaptic weight is defined by a variable D , varying between 0 and 1. The synaptic strength is then $gD(t)$, where g is the maximum synaptic strength. The dynamics of D is described by

$$\tau_d \frac{dD(t)}{dt} = 1 - D(t) + d D(t)\delta(t), \quad (8.28)$$

where τ_d is the recovery time constant of the depression and D is decreased by d ($d < 1$), right after a spike, $\delta(t)$.

An example of a depressing circuit is shown in Figure 8.12. The circuit details and its response are described in Rasche and Hahnloser (2001) and Boegerhausen et al. (2003). The voltage V_a determines the maximum synaptic strength g while the synaptic strength $gD = I_{\text{syn}}$ is exponential in the voltage V_x . The subcircuit consisting of transistors M_1 to M_3 controls the dynamics of V_x . The spike input goes to the gate terminal of M_1 and M_4 . During an input spike, a quantity of charge (determined by V_d) is removed from the node V_x . In between spikes, V_x recovers toward V_a through the diode-connected transistor M_3 (see Figure 8.13). We can convert the I_{syn} current source into an equivalent current I_d with some gain and a time constant through the current-mirror circuit consisting of M_5 , M_{syn} , and the capacitor C_{syn} , and by adjusting the voltage V_{gain} . The synaptic strength is given by $I_{\text{syn}} = gD = I_0 e^{\frac{\kappa V_x}{V_T}}$.

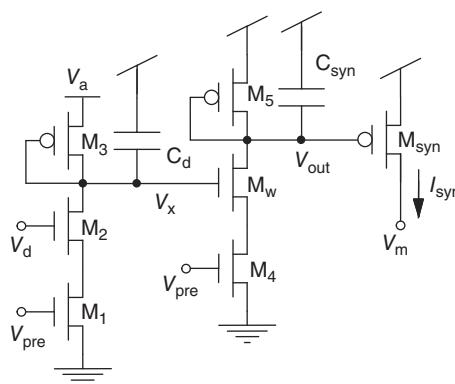


Figure 8.12 Depressing synapse circuit

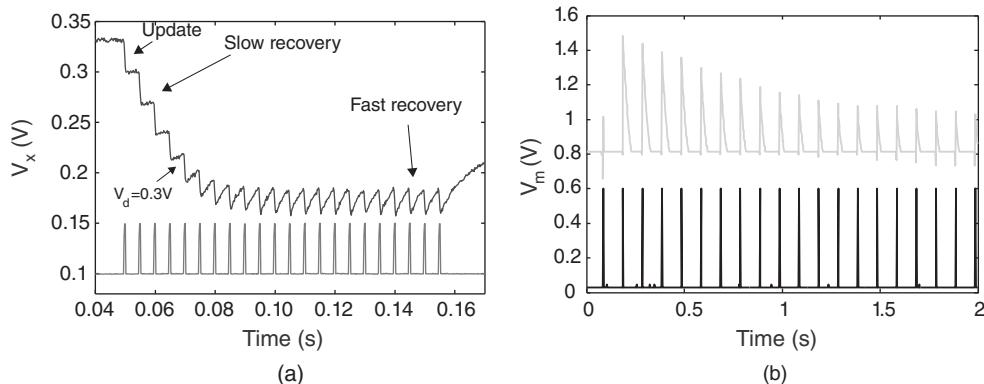


Figure 8.13 Responses measured from the depressing synapse circuit of Figure 8.12. (a) Change of V_x over time. It is decreased when an input spike arrives and it recovers back to the quiescent value at different rates dependent on its distance from the resting value, $V_a \approx 0.33$ V. Adapted from Boegershausen et al. (2003). Reproduced with permission of MIT Press. (b) Transient response of a neuron (by measuring its membrane potential, V_m) when stimulated by a regular spiking input (bottom curve) through a depressing synapse. The bottom trace in each subplot shows the input spike train

8.3.2 Long-Term Plasticity

In a synaptic model of short-term plasticity, the weight of the synapse is dependent on the short time history of the input spike activity. In the case of long-term plasticity, the weight is determined by both the pre-synaptic and post-synaptic activity. The VLSI circuits that implement such learning rules range from Hebbian rules to spike-timing dependent plasticity (STDP) rule and its variants. The various types of long-term plastic VLSI synapse circuits have been implemented following a general block diagram shown in Figure 8.14.

Analog Inputs for Weight Update

One of the earliest attempts in building learning synapses come out of the floating-gate circuits work in Mead's lab in the 1990s. (There was an earlier work (Holler et al. 1989) on trainable networks). These circuits capitalize on the availability of nonvolatile memory storage mechanisms, hot-electron injection, and electron tunneling in a normal CMOS process

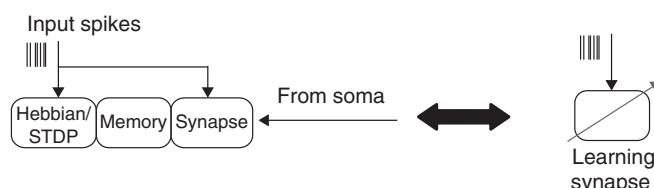


Figure 8.14 General block diagram for a learning synapse. The components are a Hebbian/STDP block which implements the learning rule, a memory block, and a synaptic circuit

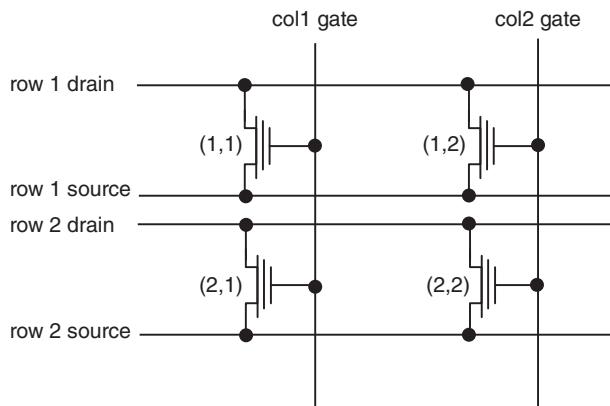


Figure 8.15 A 2×2 synaptic array of high-threshold floating-gate transistors with circuits to update weight using tunneling and injection mechanisms. The row synapses share a common drain wire, so tunneling at synapse (1,1) is activated by raising the row 1 drain voltage to a high value and by setting col1 gate to a low value. To inject at the same synapse, col1 gate is increased and the row 1 drain voltage is lowered

leading to the ability to modify the memory or voltage on a floating gate (Diorio et al. 1996, 1998; Hasler 2005; Hasler et al. 1995, 2001; Liu et al. 2002). These mechanisms are further explained in Chapter 10 and an example synaptic array is shown in Figure 8.15. These nonvolatile update mechanisms are similar to the ones used in electrically erasable programmable read only memory (EEPROM) technology except that EEPROM transistors are optimized for digital programming and binary-valued data storage. The floating-gate silicon synapse has the following desirable properties: the analog memory is nonvolatile and the memory update can be bidirectional. The synapse output is the product of the input signal and the stored value. The single transistor synapse circuit essentially implements all three components of the learning synapse in Figure 8.14. Although the initial work was centered around high-threshold nFET transistors in specialized bipolar technology, similar results were obtained from pFET transistors in normal CMOS technology (Hasler et al. 1999), hence current work is centered around arrays of pFET synapses connected to various neuron models (Brink et al. 2013). Various floating-gate learning synapses that use the explicit dynamics offered by the floating-gate dynamics include the correlation-learning rules (Dugger and Hasler 2004) and later floating-gate synapses that use spike-based inputs following a STDP rule (Häfliger and Rasche 1999; Liu and Moeckel 2008; Ramakrishnan et al. 2011). The advantage of the floating-gate technology for implementing synapses is that the parameters for the circuits can be stored using the same nonvolatile technology (Basu et al. 2010).

Spike-Based Learning Circuits using Floating-Gate Technology

The earliest spike-based learning VLSI circuit was described by Häfliger et al. (1996). This circuit implemented a spike-based version of the learning rule based on the Riccati equation as described by Kohonen (1984). This time-dependent learning rule (more thoroughly described

in Häfliger 2007) that the circuit is based on, achieves exact weight normalization, if the output mechanism is an I&F neuron (weight vector normalization here means that the weight vector length will always converge to a given constant length) and can detect temporal correlations in spike trains. (Input spike trains that show a peak around zero in their cross-correlograms with other inputs, i.e., they tend to fire simultaneously, will be much more likely to increase synaptic weights.)

To describe the weight update rule as a differential equation, the input spike train x to the synapse in question (and correspondingly the output spike train from the neuron y) is described as sum of Dirac delta function:

$$x(t) = \sum_j \delta(t - t_j), \quad (8.29)$$

where t_j is the time of the j th input spike.

Sometimes it is more convenient to use the characteristic function x' for the set of input spikes, that is, a function that is equal to 1 at firing times and 0 otherwise.

The weight update rewards a potential causal relationship between an input spike and a subsequent action potential, that is, when an action potential follows an input spike. This is very much in the spirit of Hebb's original postulate which says that an input 'causing' an action potential should lead to an increase in synaptic weight (Hebb 1949). On the other hand, if an action potential is not preceded by a synaptic input, that weight shall tend to be decreased. To implement this, a 'correlation signal' c is introduced for each synapse. Its purpose is to keep track of recent input activity. Its dynamics is described as follows:

$$\frac{dc}{dt} = x - \frac{1}{\tau} c - y(c + x'). \quad (8.30)$$

The correlation signal c at a synapse is incremented by 1 for every input spike, decays exponentially with time constant τ , and is reset to 0 with every output spike.

This equation has been approximated in Häfliger (2007) with the circuit in Figure 8.16. An active-low input spike 'nPre' activates the current source transistor (M2) biased by 'nInc' and increments the voltage on node 'corr' which represents the variable c . This voltage is subject to the leakage current through M5. This constant leakage current constitutes the main difference to the theoretical rule, because it is not a resistive current. An output spike causes the signal 'Ap' to go high for a few microseconds. During this time, 'corr' is reset through M3 and M4. These signal dynamics are illustrated in the inset in the upper right of Figure 8.16.

The theoretical weight update rule originally proposed in Häfliger et al. (1996) but more conveniently described in Häfliger (2007) rewards causal input–output relationships and punishes noncausal relationships following

$$\frac{dw}{dt} = \mu y \left(c - \frac{1}{v^2} w \right). \quad (8.31)$$

At every output spike, the weight w is incremented by μc and decremented by $\frac{\mu}{v^2} w$, where μ is the learning rate and v is the length to which the weight vector is normalized.

Transistors M6 through M9 in Figure 8.16 compute the positive term μc of that update rule, as sampled at the onset of a neuron output pulse 'Ap' (which represents a spike in the

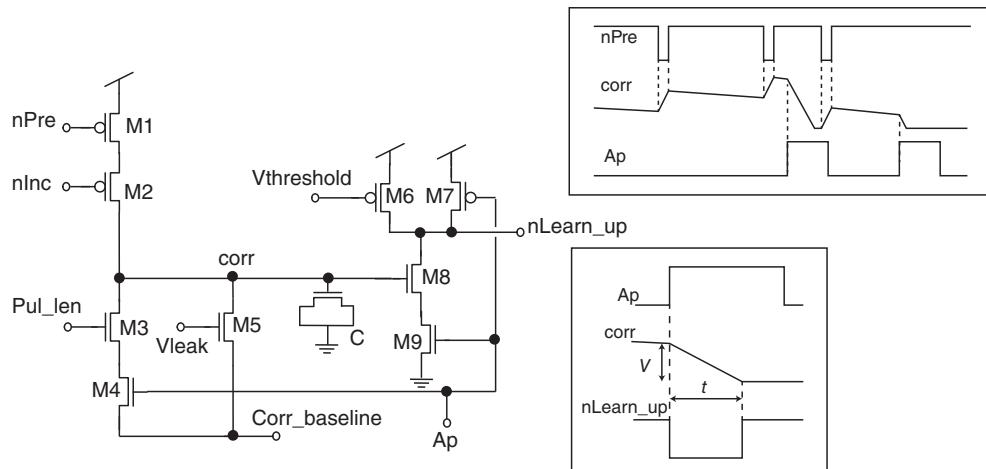


Figure 8.16 Circuit that approximates Eq. (8.30) (transistors M1, M2, and M5) and the positive weight update term of Eq. (8.31) (transistors M3 and M4, and M6 through M9). The latter is a digital pulse ‘nLearn_up’ the duration of which corresponds to term c as the neuron fires. This circuit implements the module for increasing the weight (the LTP part) in the Hebbian block of Figure 8.14

spike train y): they linearly transduce the voltage on node ‘corr’ into the time domain as an active-low pulse length as illustrated in the inset on the bottom right of the figure: the signal ‘Ap’ activates the current comparator M6/M8 and its output voltage ‘nLearn_up’ goes low for the time it takes ‘corr’ to reach its idle voltage ‘Corr_baseline’, when depleted through the current limit set by ‘Pul_len,’ that is, for a duration that is proportional to c . M6 is biased by ‘Vthreshold’ to source a current only slightly bigger than the current supplied by M8 when ‘corr’ is equal to ‘Corr_baseline.’

So the result of this circuit is a width-modulated digital pulse in the time domain. This pulse can be connected to any kind of analog weight storage cell that increments its content proportionally to the duration of that pulse, for example, a capacitor or floating gate which integrates a current from a current source or tunneling node turned on for the duration of that pulse.

The negative update term in Eq. (8.31) is implemented in a similar manner with a width-modulated output pulse, for example, activating a negative current source on a capacitive or floating gate weight storage cell. The original circuit (Häfliger et al. 1996) used capacitive weight storage, and later implementations were made with analog floating gate (Häfliger and Rasche 1999), and ‘weak’ multilevel storage cells (Häfliger 2007; Riis and Häfliger 2004).

Spike Timing-Dependent Plasticity Rule

The spike-based learning rule previously outlined in Section 6.3.1 of Chapter 6 is also called the STDP rule. It models the behavior of synaptic plasticity observed in physiological experiments (Markram et al. 1997; Song et al. 2000; van Rossum et al. 2000). The form of this learning rule is shown in Figure 8.17. The synaptic weight is increased when the pre-synaptic spike precedes a post-synaptic spike within a time window. For the reverse order of occurrence of pre- and post-synaptic spikes, the synaptic weight is decreased. In the additive form, the

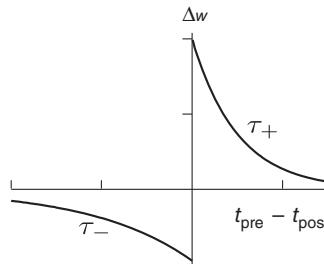


Figure 8.17 Temporally asymmetric learning curve or spike-timing-dependent plasticity curve

change in weight $A(\Delta t)$ is independent of the synaptic weight A and the weights saturate at a fixed maximum and minimum value (Abbott and Nelson 2000):

$$\Delta w = \begin{cases} W_+ e^{-\Delta t / \tau_+} & \text{if } \Delta t > 0 \\ W_- e^{-\Delta t / \tau_-} & \text{if } \Delta t < 0, \end{cases} \quad (8.32)$$

where $t^{\text{post}} = t^{\text{pre}} + \Delta t$. In the multiplicative form, the change in weight is dependent on the synaptic weight value.

STDP Silicon Circuits

Silicon STDP circuits that emulate the weight update curves in Figure 8.17 were described in Bofill-i-Petit et al. (2002), Indiveri (2003), and Indiveri et al. (2006). The circuits in Figure 8.18

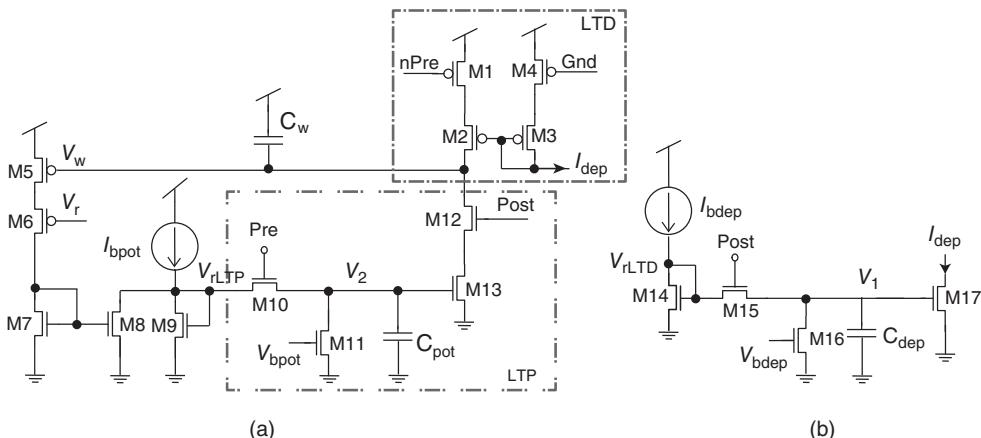


Figure 8.18 Weight change circuits. (a) Strength of the synapse is inversely proportional to the value of V_w . The higher V_w , the smaller the weight of the synapse. This section of the weight change circuit detects causal spike correlations. (b) This part of the circuit creates the decaying shape of the depression side of the learning window

can implement both the weight-dependent and weight-independent updates (Bofill-i-Petit and Murray 2004). The weight of each synapse is represented by the charge stored on its weight capacitor C_w . The strength of the weight is inversely proportional to V_w . The closer the value of V_w is to ground, the stronger is the synapse.

The spike output of the neuron driven by the plastic synapse is defined as Post. When Pre (input spike) is active, the voltage created by I_{bpot} on the diode connected transistor M9 is copied to the gate of M13. This voltage across C_{pot} decays with time from its peak value due to a leakage current set by V_{bpot} . When the post-synaptic neuron fires, Post switches M12 on. Therefore, the weight is potentiated (V_w decreased) by an amount which reflects the time elapsed since the last pre-synaptic event.

The weight-dependent mechanism is introduced by the simple linearized V - I configuration consisting of M5 and M6 and the current mirror M7 and M8 (see Figure 8.18a). M5 is a low gain transistor operated in strong inversion whereas M6 is a wide transistor made to operate in weak inversion such that it has even higher gain. When V_w decreases (weight increase) the current through M5 and M6 increases, but M5 is maintained in the linear region by the high gain transistor. Thus, a current proportional to the value of the weight is subtracted from I_{bpot} . The resulting smaller current injected into M9 will cause a drop in the peak of potentiation for large weight values.

In a similar manner to potentiation, the weight is weakened by the circuit shown in Figure 8.18b when it detects a noncausal interaction between a pre-synaptic and a post-synaptic spike. When a post-synaptic spike event is generated, a Post pulse charges C_{dep} . The accumulated charge leaks linearly through M16 at a rate set by V_{bdep} . A nonlinear decaying current (I_{dep}) is sent to the weight change circuits placed in the input synapse (see I_{dep} in Figure 8.18a). When a pre-synaptic spike reaches a synapse, M1 is turned on. If this occurs soon enough after the Post pulse, V_w is brought closer to V_{dd} (weight strength decreased).

This circuit can be used to implement weight-independent learning by setting V_r to V_{dd} . Modified versions of this circuit have been used to express variants of the STDP rule, for example, the additive and multiplicative forms of the weight update as discussed in Song et al. (2000) (Bamford et al. 2012).

Binary Weighted Plastic Synapses

Stochastic Binary

The bistable synapses described in Section 6.3.2 of Chapter 6 have been implemented in analog VLSI (Fusi et al. 2000). This synapse uses an internal state represented by an analog variable that makes jumps up or down whenever a spike appears on the pre-synaptic neuron. The direction of the jump is determined by the level of depolarization of the post-synaptic neuron. The dynamics of the analog synaptic variable is similar to the perceptron rule, except that the synapse maintains its memory on long timescales in the absence of stimulation. Between spikes, the synaptic variable drifts linearly up, toward a ceiling or down, toward a floor depending on whether the variable is above or below a synaptic threshold. The two values that delimit the synaptic variable are the stable synaptic efficacies.

Because stimulation takes place on a finite (short) interval, stochasticity is generated by the variability in both the pre- and post-synaptic neurons spike timings. Long-term potentiation

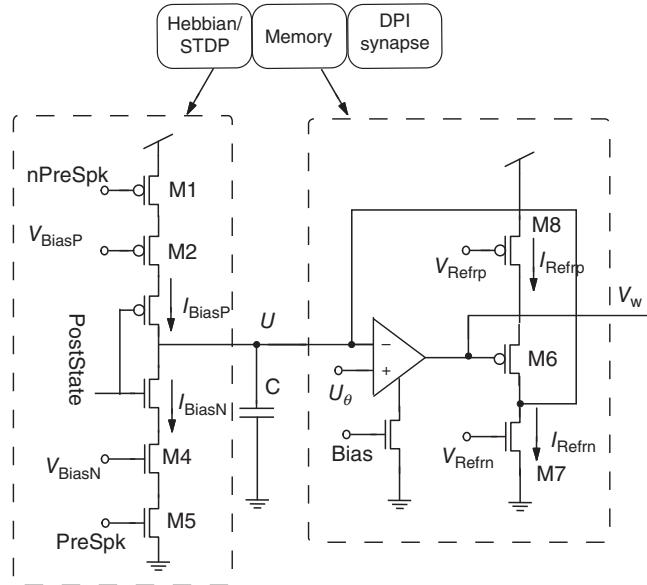


Figure 8.19 Schematic of the bistable synaptic circuit. The activity-dependent Hebbian block implements $H(t)$. The voltage across the capacitor C is $U(t)$ and is related to the internal variable $X(t)$ by Eq. (8.33). The memory block implements $R(t)$ and provides the refresh currents

(LTP) and long-term depression (LTD) transitions take place on pre-synaptic bursts, when the jumps accumulate to overcome the refresh drifts. The synapse can preserve a continuous set of values for periods of the order of its intrinsic time constants. But on long timescales, only two values are preserved: the synaptic efficacy fluctuates in a band near one of the two stable values until a strong stimulation produces enough spikes to drive it out of the band and into the neighborhood of the other stable value. The circuit shown in Figure 8.19 shows the different building blocks of the learning synapse.

The capacitor C acts as an analog memory element: the dimensionless internal synaptic state $X(t)$ is related to the capacitor voltage in the following way:

$$X(t) = \frac{U(t) - U_{\min}}{U_{\max} - U_{\min}}, \quad (8.33)$$

where $U(t)$ is the voltage across the synaptic capacitor and can vary in the region delimited by $U_{\min} \approx 0$ and $U_{\max} \approx V_{dd}$.

Hebbian Block

The Hebbian block implements $H(t)$ which is activated only on the arrival of a pre-synaptic spike. The synaptic internal state jumps up or down, depending on the post-synaptic depolarization. The binary input signal PostState determines the direction of the jump and is the outcome of the comparison between two voltage levels: the depolarization of the post-synaptic

neuron $V(t)$ and the firing threshold θ_V . If the depolarization is above the threshold, then PostState is ≈ 0 ; otherwise it is near the power supply voltage V_{dd} . (The circuit for generating PostState is not shown). In the absence of pre-synaptic spikes, M1 and M5 are not conducting and no current flows into the capacitor, C. During the emission of a pre-synaptic spike ($nPreSpk$ is low), the PostState signal controls which branch is activated. If PostState = 0 (the post-synaptic depolarization is above the threshold θ_V), current charges up the capacitor C. The charging rate is determined by V_{BiasP} , which is the current flowing through M2. Analogously, when PostState = 1, the lower branch is activated, and the capacitor is discharged to ground.

Memory Block

The refresh block implements $R(t)$, which charges the capacitor if the voltage $U(t)$ is above the threshold U_θ (corresponding to θ_X) and discharges it otherwise. It is the term that tends to damp any small fluctuation that drives $U(t)$ away from one of the two stable states. If $U(t) < U_\theta$, the voltage output of the comparator is $\approx V_{dd}$, M5 is not conducting while M6 is conducting. The result is a discharge of the capacitor by the current I_{Refrn} . If $U(t) > U_\theta$, the low output of the comparator turns on M5 and the capacitor is charged at a rate proportional to the difference between the currents I_{Refrp} and I_{Refrn} .

Stochastic Binary Stop-Learning

The spike-based learning algorithm implemented in Figure 8.19 is based on the model described in Brader et al. (2007) and Fusi et al. (2000). This algorithm can be used to implement both unsupervised and supervised learning protocols, and train neurons to act as perceptrons or binary classifiers. Typically, input patterns are encoded as sets of spike trains with different mean frequencies, while the neuron's output firing rate represents the binary classifier output. The learning circuits that implement this algorithm can be subdivided into two main blocks: a spike-triggered weight-update module with bistable weights, present in each plastic synapse, and a post-synaptic stop-learning control module, present in the neuron's soma. The 'stop-learning' circuits implement the characteristic feature of this algorithm, which stops updating weights if the output neuron has a very high or very low output firing rate, indicating the fact that the dot product between the input vector and the learned synaptic weights is either close to 1 (pattern recognized as belonging to the trained class) or close to 0 (pattern not in trained class).

The post-synaptic stop-learning control circuits are shown in Figure 8.20. These circuits produce two global signals V_{UP} and V_{DN} , shared among all synapses belonging to the same dendritic tree, to enable positive and/or negative weight updates respectively. Post-synaptic spikes produced by the I&F neuron are integrated by a DPI circuit. The DPI circuit produces the signal V_{Ca} which is related to the Calcium concentration in real neurons, and represents the recent spiking activity of the neuron. This signal is compared to three different thresholds (V_{thk1} , V_{thk2} , and V_{thk3}) by three corresponding current-mode winner-take-all circuits (Lazzaro et al. 1989). In parallel, the neuron's membrane potential V_m is compared to a fixed threshold V_{thm} by a transconductance amplifier. The values of V_{UP} and V_{DN} depend on the output of this amplifier, as well as the Calcium concentration signal V_{Ca} . Specifically, if $V_{thk1} < V_{Ca} < V_{thk3}$ and $V_m > V_{thm}$, then increases in synaptic weights ($V_{UP} < V_{dd}$) are enabled. If $V_{thk1} < V_{Ca} < V_{thk2}$

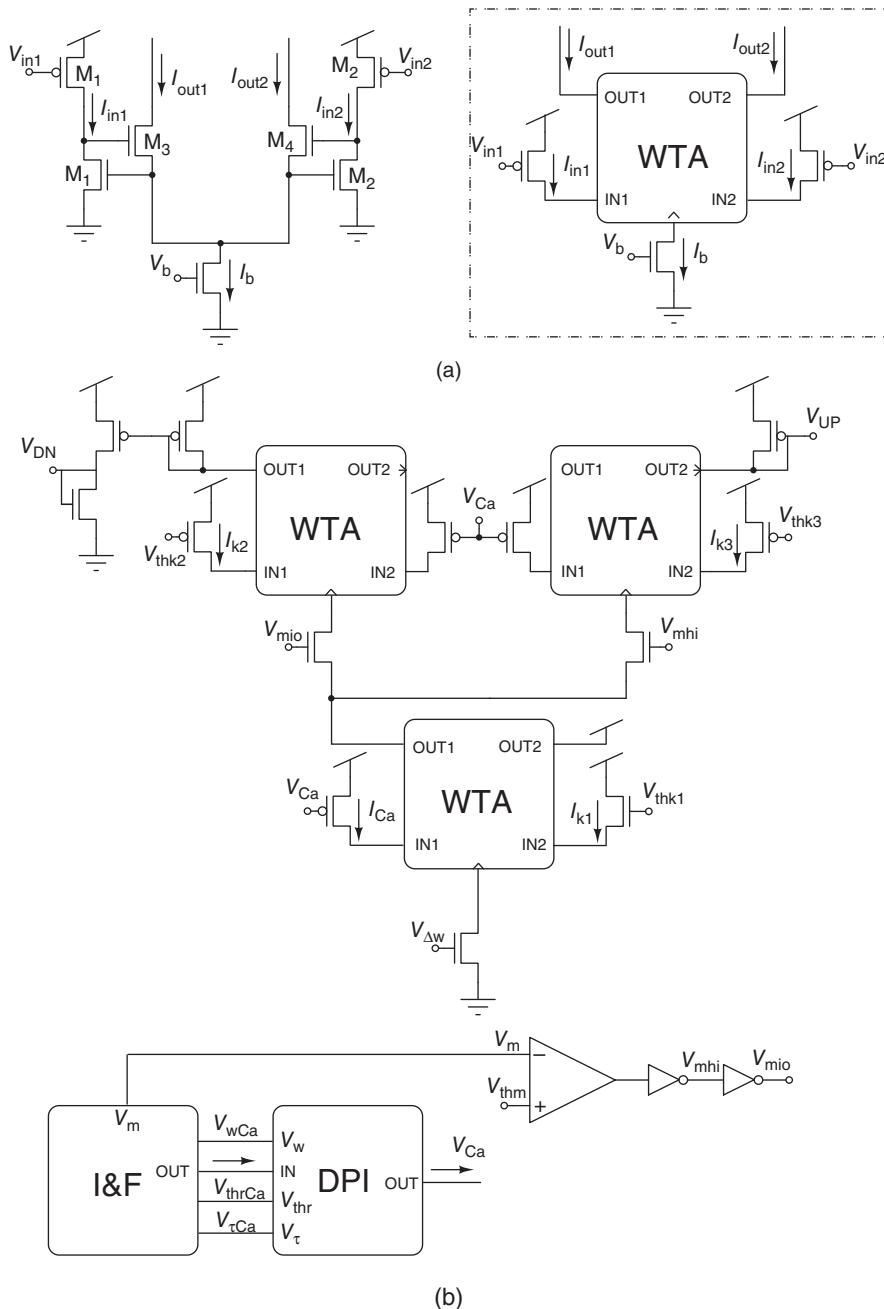


Figure 8.20 Post-synaptic stop-learning control circuits at the soma of neurons with bistable synapses. The winner-take-all (WTA) circuit in (a) and the DPI circuit are used in the stop-learning control circuit in (b)

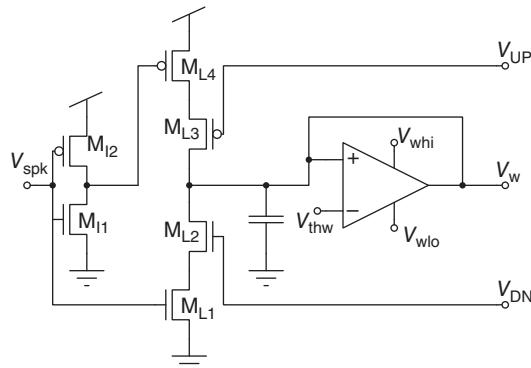


Figure 8.21 Spike-based learning circuits. Pre-synaptic weight-update module or Hebbian block of each learning synapse. The signals V_{UP} and V_{DN} are generated by the circuit in Figure 8.20

and $V_{mem} < V_{thm}$, then decreases in synaptic weights ($V_{DN} > 0$) are enabled. Otherwise no changes in the synaptic weights are allowed ($V_{UP} = V_{dd}$, and $V_{DN} = 0$).

The pre-synaptic weight-update module comprises four main blocks: an input stage (see $M_{I1} - M_{I2}$ in Figure 8.21), a spike-triggered weight update circuit ($M_{L1} - M_{L4}$ in Figure 8.21), a bistability weight refresh circuit (see transconductance amplifier in Figure 8.21), and a current-mode DPI circuit (not shown). The bistability weight refresh circuit is a positive-feedback amplifier with very small ‘slew-rate’ that compares the weight voltage V_w to a set threshold V_{thw} , and slowly drives it toward one of the two rails V_{whi} or V_{wlo} , depending whether $V_w > V_{thw}$ or $V_w < V_{thw}$, respectively. This bistable drive is continuous and its effect is superimposed to the one from the spike-triggered weight update circuit. Upon the arrival of an input address-event, two digital pulses trigger the weight update block and increase or decrease the weight, depending on the values of V_{UP} and V_{DN} : if during a pre-synaptic spike the V_{UP} signal from the post-synaptic, stop-learning control module is enabled ($V_{UP} < V_{dd}$), the synapse’s weight V_w undergoes an instantaneous increase. Similarly, if during a pre-synaptic spike the V_{DN} signal from the post-synaptic, weight control module is high, V_w undergoes an instantaneous decrease. The amplitude of the EPSC produced by the DPI block upon the arrival of the pre-synaptic spike is proportional to $V_{\Delta w}$.

Mitra et al. (2009) show how such circuits can be used to carry out classification tasks, and characterize the performance of this VLSI learning system. This system also displays the transition dynamics in Figure 6.3 of Chapter 6.

Binary SRAM STDP

A different method of updating the synaptic weight based on the STDP learning rule is the binary STDP synaptic circuit (Arthur and Boahen 2006). It is built from three subcircuits: the decay circuit, the integrator circuit, and static random access memory (SRAM) (Figure 8.22). The decay and integrator subcircuits are used to implement potentiation and depression in a symmetric fashion. The SRAM holds the current binary state of the synapse, either potentiated or depressed.

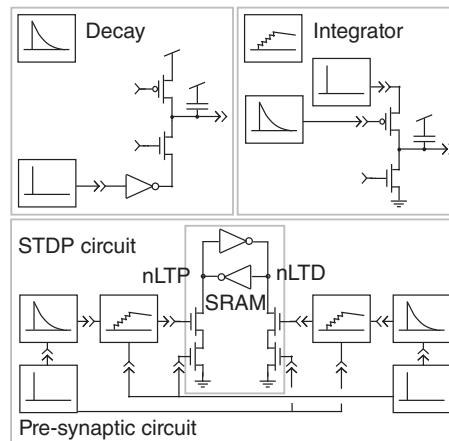


Figure 8.22 Binary static random access memory (SRAM) STDP circuit. The circuit is composed of three subcircuits: decay, integrator, and STDP SRAM. Adapted from Arthur and Boahen (2006). Reproduced with permission of MIT Press

For potentiation, the decay block remembers the last pre-synaptic spike. Its capacitor is charged when that spike occurs and discharges linearly thereafter. A post-synaptic spike samples the charge remaining on the capacitor, passes it through an exponential function, and dumps the resultant charge into the integrator. This charge decays linearly thereafter. At the time of the post-synaptic spike, the SRAM, a cross-coupled inverter pair, reads the voltage on the integrator's capacitor. If it exceeds a threshold, the SRAM switches state from depressed to potentiated ($nLTD$ goes high and $nLTP$ goes low). The depression side of the STDP circuit is exactly symmetric, except that it responds to post-synaptic activation followed by pre-synaptic activation and switches the SRAM's state from potentiated to depressed ($nLTP$ goes high and $nLTD$ goes low). When the SRAM is in the potentiated state, the pre-synaptic spike activates the principal neuron's synapse; otherwise the spike has no effect. The output of the SRAM activates a log-domain synapse circuit described in Section 8.2.1.

8.4 Discussion

The historical picture of how the various synapse circuits have evolved over the years is depicted in Figure 8.23. Synaptic circuits can be more costly (in number of transistors) than a silicon neuron circuit. As more features are added, the transistor count per synapse grows. In addition, these circuits still face the difficulty of an elegant implementation of a linear resistor especially when the circuits are operated in subthreshold. Therefore, the designer should consider the necessary features needed for the application rather than building a synapse with all possible features. This chapter focuses on mixed-signal synapse circuits. An all-digital solution for both synapses and neurons has also been considered because these circuits, while dissipating more power on average, can be designed faster using existing design tools as demonstrated by Arthur et al. (2012), Merolla et al. (2011), and Seo et al. (2011).

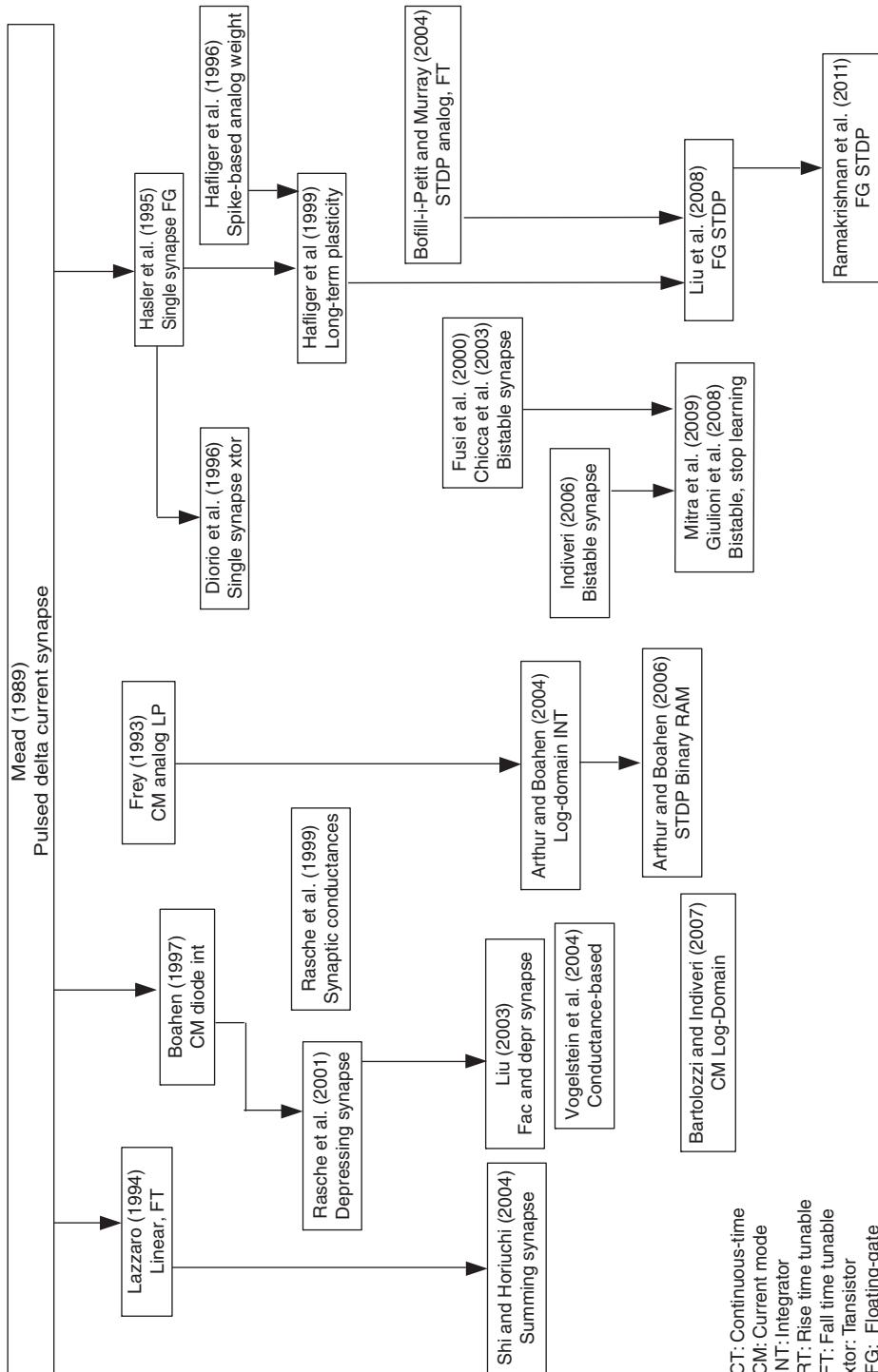


Figure 8.23 Historical tree

References

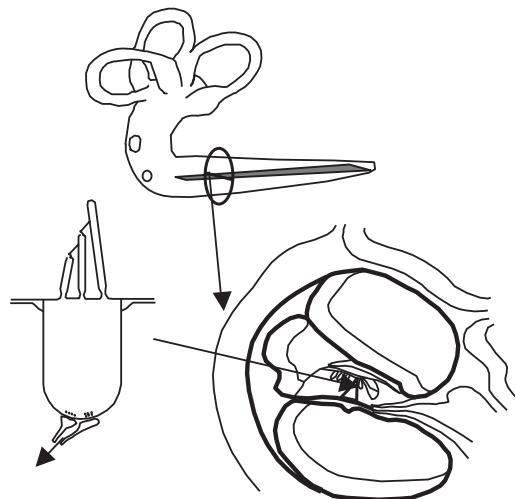
- Abbott LF and Nelson SB. 2000. Synaptic plasticity: taming the beast. *Nat. Neurosci.* **3**, 1178–1183.
- Abbott LF, Varela JA, Sen K, and Nelson SB. 1997. Synaptic depression and cortical gain control. *Science* **275**(5297), 220–223.
- Arthur JV and Boahen K. 2004. Recurrently connected silicon neurons with active dendrites for one-shot learning. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, **3**, pp. 1699–1704.
- Arthur JV and Boahen K. 2006. Learning in silicon: timing is everything. In: *Advances in Neural Information Processing Systems 18 (NIPS)* (eds Weiss Y, Schölkopf B, and Platt J). MIT Press, Cambridge, MA. pp. 75–82.
- Arthur JV, Merolla PA, Akopyan F, Alvarez R, Cassidy A, Chandra S, Esser S, Imam N, Risk W, Rubin D, Manohar R, and Modha D. 2012. Building block of a programmable neuromorphic substrate: a digital neurosynaptic core. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, pp. 1–8.
- Bamford SA, Murray AF, and Willshaw DJ. 2012. Spike-timing-dependent plasticity with weight dependence evoked from physical constraints. *IEEE Trans. Biomed. Circuits Syst.* **6**(4), 385–398.
- Bartolozzi C and Indiveri G. 2007. Synaptic dynamics in analog VLSI. *Neural Comput.* **19**(10), 2581–2603.
- Basu A, Ramakrishnan S, Petre C, Koziol S, Brink S, and Hasler PE. 2010. Neural dynamics in reconfigurable silicon. *IEEE Trans. Biomed. Circuits Syst.* **4**(5), 311–319.
- Boahen KA. 1997. *Retinomorphic Vision Systems: Reverse Engineering the Vertebrate Retina*. PhD thesis. California Institute of Technology, Pasadena, CA.
- Boahen KA. 1998. Communicating neuronal ensembles between neuromorphic chips. In: *Neuromorphic Systems Engineering* (ed. Lande TS). The International Series in Engineering and Computer Science, vol. 447. Springer. pp. 229–259.
- Boegerhausen M, Suter P, and Liu SC. 2003. Modeling short-term synaptic depression in silicon. *Neural Comput.* **15**(2), 331–348.
- Bofill-i-Petit A and Murray AF. 2004. Synchrony detection by analogue VLSI neurons with bimodal STDP synapses. In: *Advances in Neural Information Processing Systems 16 (NIPS)* (eds. Thrun S, Saul L, and Schölkopf B). MIT Press, Cambridge, MA. pp. 1027–1034.
- Bofill-i-Petit A, Thompson DP, and Murray AF. 2002. Circuits for VLSI implementation of temporally asymmetric Hebbian learning. In: *Advances in Neural Information Processing Systems 14 (NIPS)* (eds. Dietterich TG, Becker S, and Ghahramani Z). MIT Press, Cambridge, MA. pp. 1091–1098.
- Brader JM, Senn W, and Fusi S. 2007. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* **19**(11), 2881–2912.
- Brink S, Nease S, Hasler P, Ramakrishnan S, Wunderlich R, Basu A, and Degnan B. 2013. A learning-enabled neuron array IC based upon transistor channel models of biological phenomena. *IEEE Trans. Biomed. Circuits Syst.* **7**(1), 71–81.
- Chicca E. 2006. *A Neuromorphic VLSI System for Modeling Spike-Based Cooperative Competitive Neural Networks*. PhD thesis, ETH Zürich, Zürich, Switzerland.
- Chicca E, Badoni D, Dante V, D'Andreagiovanni M, Salina G, Carota L, Fusi S, and Del Giudice P. 2003. A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory. *IEEE Trans. Neural Netw.* **14**(5), 1297–1307.
- Destexhe A, Mainen ZF, and Sejnowski TJ. 1998. Kinetic models of synaptic transmission. *Methods in Neuronal Modelling, from Ions to Networks*. The MIT Press, Cambridge, MA. pp. 1–25.
- Diorio C, Hasler P, Minch BA, and Mead C. 1996. A single-transistor silicon synapse. *IEEE Trans. Electr. Dev.* **43**(11), 1972–1980.
- Diorio C, Hasler P, Minch BA, and Mead CA. 1998. Floating-gate MOS synapse transistors. *Neuromorphic Systems Engineering: Neural Networks in Silicon*. Kluwer Academic Publishers, Norwell, MA. pp. 315–338.
- Dugger J and Hasler P. 2004. A continuously adapting correlating floating-gate synapse. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 1058–1061.
- Farquhar E and Hasler P. 2005. A bio-physically inspired silicon neuron. *IEEE Trans. Circuits Syst. I: Regular Papers* **52**(3), 477–488.
- Frey DR. 1993. Log-domain filtering: an approach to current-mode filtering. *IEE Proc. G: Circuits, Devices and Systems.* **140**(6), 406–416.
- Frey DR. 1996. Exponential state space filters: a generic current mode design strategy. *IEEE Trans. Circuits Syst. II* **43**, 34–42.

- Fusi S, Annunziato M, Badoni D, Salamon A, and Amit DJ. 2000. Spike-driven synaptic plasticity: theory, simulation, VLSI implementation. *Neural Comput.* **12**(10), 2227–2258.
- Gilbert B. 1975. Translinear circuits: a proposed classification. *Electron. Lett.* **11**, 14–16.
- Giulioni M, Camilleri P, Dante V, Badoni D, Indiveri G, Braun J, and Del Giudice P. 2008. A VLSI network of spiking neurons with plastic fully configurable “stop-learning” synapses. *Proc. 15th IEEE Int. Conf. Electr., Circuits Syst. (ICECS)*, pp. 678–681.
- Häfliger P. 2007. Adaptive WTA with an analog VLSI neuromorphic learning chip. *IEEE Trans. Neural Netw.* **18**(2), 551–572.
- Häfliger P and Rasche C. 1999. Floating gate analog memory for parameter and variable storage in a learning silicon neuron. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **II**, pp. 416–419.
- Häfliger P, Mahowald M, and Watts L. 1996. A spike based learning neuron in analog VLSI. In: *Advances in Neural Information Processing Systems 9 (NIPS)* (eds. Mozer MC, Jordan MI, and Petsche T). MIT Press, Cambridge, MA. pp. 692–698.
- Hasler P. 2005. Floating-gate devices, circuits, and systems. *Proceedings of the Fifth International Workshop on System-on-Chip for Real-Time Applications*. IEEE Computer Society, Washington, DC. pp. 482–487.
- Hasler P, Diorio C, Minch BA, and Mead CA. 1995. Single-transistor learning synapses with long term storage. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **III**, pp. 1660–1663.
- Hasler P, Minch BA, Dugger J, and Diorio C. 1999. Adaptive circuits and synapses using pFET floating-gate devices. In: *Learning in Silicon* (ed. Cauwenberghs G). Kluwer Academic. pp. 33–65.
- Hasler P, Minch BA, and Diorio C. 2001. An autozeroing floating-gate amplifier. *IEEE Trans. Circuits Syst. II* **48**(1), 74–82.
- Hebb DO. 1949. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York.
- Holler M, Tam S, Castro H, and Benson R. 1989. An electrically trainable artificial neural network with 10240 ‘floating gate’ synapses. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)* **II**, pp. 191–196.
- Horiuchi T and Hynna K. 2001. Spike-based VLSI modeling of the ILD system in the echolocating bat. *Neural Netw.* **14**(6/7), 755–762.
- Hynna KM and Boahen K. 2001. Space-rate coding in an adaptive silicon neuron. *Neural Netw.* **14**(6/7), 645–656.
- Hynna KM and Boahen K. 2007. Thermodynamically-equivalent silicon models of ion channels. *Neural Comput.* **19**(2), 327–350.
- Indiveri G. 2000. Modeling selective attention using a neuromorphic analog VLSI device. *Neural Comput.* **12**(12), 2857–2880.
- Indiveri G. 2003. Neuromorphic bistable VLSI synapses with spike-timing dependent plasticity. In: *Advances in Neural Information Processing Systems 15 (NIPS)* (eds. Becker S, Thrun S, and Obermayer K). MIT Press, Cambridge, MA. pp. 1115–1122.
- Indiveri G, Chicca E, and Douglas RJ. 2006. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Trans. Neural Netw.* **17**(1), 211–221.
- Koch C. 1999. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press.
- Kohonen T. 1984. *Self-Organization and Associative Memory*. Springer, Berlin.
- Lazzaro J, Ryckebusch S, Mahowald MA, and Mead CA. 1989. Winner-take-all networks of O(n) complexity. In: *Advances in Neural Information Processing Systems 1 (NIPS)* (ed. Touretzky DS). Morgan-Kaufmann, San Mateo, CA. pp. 703–711.
- Lazzaro J, Wawrzynek J, and Kramer A. 1994. Systems technologies for silicon auditory models. *IEEE Micro* **14**(3), 7–15.
- Liu SC. 2003. Analog VLSI circuits for short-term dynamic synapses. *EURASIP J. App. Sig. Proc.*, **7**, 620–628.
- Liu SC and Moeckel R. 2008. Temporally learning floating-gate VLSI synapses. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2154–2157.
- Liu SC, Kramer J, Indiveri G, Delbrück T, Burg T, and Douglas R. 2001. Orientation-selective aVLSI spiking neurons. *Neural Netw.* **14**(6/7), 629–643.
- Liu SC, Kramer J, Indiveri G, Delbrück T, and Douglas R. 2002. *Analog VLSI: Circuits and Principles*. MIT Press.
- Maass W and Sontag ED. 2000. Neural systems as nonlinear filters. *Neural Comput.* **12**(8), 1743–1772.
- Markram H, Lubke J, Frotscher M, and Sakmann B. 1997. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* **275**(5297), 213–215.
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Merolla P and Boahen K. 2004. A recurrent model of orientation maps with simple and complex cells. In: *Advances in Neural Information Processing Systems 16 (NIPS)* (eds. Thrun S, Saul LK, and Scholkopf B). MIT Press, Cambridge, MA. pp. 995–1002.

- Merolla PA, Arthur JV, Akopyan F, Imam N, Manohar R, and Modha D. 2011. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45 nm. *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, September, pp. 1–4.
- Mitra S, Fusi S, and Indiveri G. 2009. Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Trans. Biomed. Circuits Syst.* **3**(1), 32–42.
- Morris RGM, Davis S, and Butcher SP. 1990. Hippocampal synaptic plasticity and NMDA receptors: a role in information storage? *Phil. Trans. R. Soc. Lond. B* **320**(1253), 187–204.
- Murray A. 1998. Pulse-based computation in VLSI neural networks. In: *Pulsed Neural Networks* (eds. Maass W and Bishop CM). MIT Press, pp. 87–109.
- Ramakrishnan S, Hasler PE, and Gordon C. 2011. Floating gate synapses with spike-time-dependent plasticity. *IEEE Trans. Biomed. Circuits Syst.* **5**(3), 244–252.
- Rasche C and Douglas RJ. 2001. Forward- and backpropagation in a silicon dendrite. *IEEE Trans. Neural Netw.* **12**(2), 386–393.
- Rasche C and Hahnloser R. 2001. Silicon synaptic depression. *Biol. Cybern.* **84**(1), 57–62.
- Riis HK and Häfliger P. 2004. Spike based learning with weak multi-level static memory. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **5**, pp. 393–396.
- Schemmel J, Brüderle D, Meier K, and Ostenhoff B. 2007. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3367–3370.
- Schemmel J, Fieres J, and Meier K. 2008. Wafer-scale integration of analog neural networks. *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, June, pp. 431–438.
- Seevinck E. 1990. Companding current-mode integrator: a new circuit principle for continuous time monolithic filters. *Electron. Lett.* **26**, 2046–2047.
- Seo J, Brezzo B, Liu Y, Parker BD, Esser SK, Montoye RK, Rajendran B, Tierno JA, Chang L, Modha DS, and Friedman DJ. 2011. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, September, pp. 1–4.
- Shi RZ and Horiuchi T. 2004. A summing, exponentially-decaying CMOS synapse for spiking neural systems. In: *Advances in Neural Information Processing Systems 16 (NIPS)* (eds. Thrun S, Saul L, and Schölkopf B). MIT Press, Cambridge, MA. pp. 1003–1010.
- Song S, Miller KD, and Abbott LF. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* **3**(9), 919–926.
- Tsodyks M and Markram H. 1997. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proc. Natl. Acad. Sci. USA* **94**(2), 719–723.
- Tsodyks M, Pawelzik K, and Markram H. 1998. Neural networks with dynamic synapses. *Neural Comput.* **10**(4), 821–835.
- van Rossum MC, Bi GQ, and Turrigiano GG. 2000. Stable Hebbian learning from spike timing-dependent plasticity. *J. Neurosci.* **20**(23), 8812–8821.
- Varela JA, Sen K, Gibson J, Fost J, Abbott LF, and Nelson SB. 1997. A quantitative description of short-term plasticity at excitatory synapses in layer 2/3 of rat primary visual cortex. *J. Neurosci.* **17**(20), 7926–7940.
- Vogelstein RJ, Mallik U, and Cauwenberghs G. 2004. Silicon spike-based synaptic array and address-event transceiver. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, **V**, pp. 385–388.
- Vogelstein RJ, Mallik U, Vogelstein JT, and Cauwenberghs G. 2007. Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses. *IEEE Trans. Neural Netw.* **18**(1), 253–265.
- Wang XJ. 1999. Synaptic basis of cortical persistent activity: the importance of NMDA receptors to working memory. *J. Neurosci.* **19**(21), 9587–9603.
- Yu T and Cauwenberghs G. 2010. Analog VLSI biophysical neurons and synapses with programmable membrane channel kinetics. *IEEE Trans. Biomed. Circuits Syst.* **4**(3), 139–148.

9

Silicon Cochlea Building Blocks



This chapter goes into the details of some of the circuit blocks used in the silicon cochleas discussed in Chapter 4. It looks at some of the basic circuit structures used in the design of various one-dimensional (1D) and two-dimensional (2D) silicon cochleas. Nearly all silicon cochleas are built around second-order low-pass or band-pass filters. These filters are usually described as second-order sections.

9.1 Introduction

For the 1D cochlea, voltage-domain filters have been most commonly used, largely because of historical reasons – voltage-domain circuits were well understood – and partly because there was no compelling reason for using current-domain circuits. The first 2D cochlea (Watts et al.

1992) indeed was also designed using voltage mode second-order filters, but the use of current mode second-order filters greatly simplifies the implementation of the resistive grid, modeling the fluid in the 2D cochlea (see Chapter 4). Hence, later versions of the 2D cochlea have used current-domain (i.e., log-domain) filters.

In this chapter, we will first look at the voltage-domain second-order filters, followed by the log-domain second-order filter. Both the 1D and the 2D cochleas, whether implemented in the voltage or current domain, need exponentially decreasing currents to bias the filters, which we discuss in a separate section. As discussed in Chapter 4, the inner hair cells (IHC) stimulate the auditory nerve neurons and cause them to fire. At the end of this chapter an implementation of the IHC will be presented.

9.2 Voltage-Domain Second-Order Filter

The second-order filters of the 1D voltage-domain cochlea are built from three transconductance amplifiers. We will first introduce the transconductance amplifier and then discuss the second-order filters. We also give a detailed analysis of the operation of these circuits.

9.2.1 Transconductance Amplifier

The transconductance amplifier in its most basic version is shown in Figure 9.1a. When biased in weak inversion it has a hyperbolic tangent transfer function given by

$$I_{\text{out}} = I_{\text{bias}} \tanh \left(\frac{V_+ - V_-}{2nU_T} \right) \quad (9.1)$$

with I_{bias} , V_+ , and V_- as in Figure 9.1, and n is the weak inversion slope factor which depends on the technology and normally has a value somewhere between 1 and 2. The thermal voltage U_T is given by $U_T = kT/q$, with k the Boltzmann constant, T the temperature in Kelvin, and q the charge of an electron. U_T is about 25 mV at room temperature.

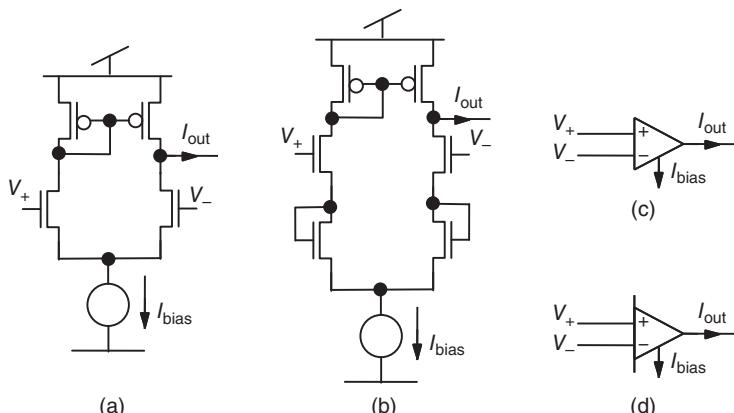


Figure 9.1 A basic transconductance amplifier: (a) schematic and (c) symbol, a transconductance amplifier with an enlarged linear range; (b) schematic and (d) symbol

The transconductance amplifier is biased in weak inversion when the current through the differential pair transistors is much smaller than the specific current of these transistors, that is,

$$I_{\text{bias}} \ll I_S = 2n\mu C_{\text{ox}} W / L U_T^2, \quad (9.2)$$

where W is the channel width of the transistor, L the length, μ the mobility of the minority carriers, and C_{ox} the gate oxide capacitance per unit area. The specific current depends on process parameters, the geometry of the transistor, and temperature (Vittoz 1994). For small inputs ($|V_+ - V_-| < 60 \text{ mV}$) we can approximate the amplifier as a linear transconductance:

$$I_{\text{out}} = g_a(V_+ - V_-) \quad (9.3)$$

with the transconductance given by

$$g_a = \frac{I_{\text{bias}}}{2nU_T}. \quad (9.4)$$

This last equation shows that the transconductance of the amplifier is proportional to the bias current when the amplifier operates in weak inversion.

As we will see in Section 9.2.4, we will also need a transconductance amplifier with an enlarged linear range as shown in Figure 9.1b. The transfer function of this amplifier is given by

$$I_{\text{out}} = I_{\text{bias}} \tanh \left(\frac{V_+ - V_-}{2n(n+1)U_T} \right) \quad (9.5)$$

and

$$g_a = \frac{I_{\text{bias}}}{2n(n+1)U_T}. \quad (9.6)$$

The linear range of this amplifier is thus enlarged by a factor $n + 1$, where n is about 1.5, and the transconductance is reduced by a factor $n + 1$ with respect to the normal transconductance amplifier with the same bias current. The transfer functions of both amplifiers are shown in Figure 9.2.

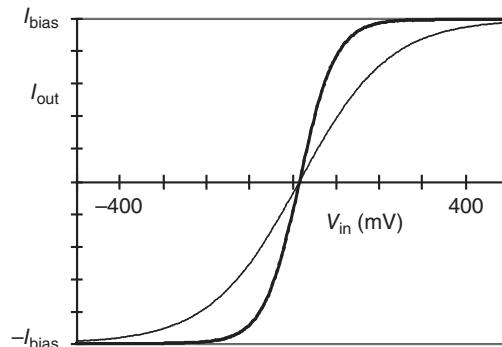


Figure 9.2 Transfer functions of the transconductance amplifier (heavy line) and the modified transconductance amplifier

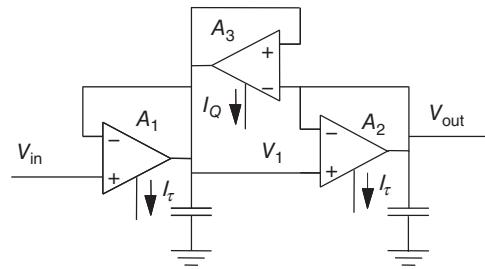


Figure 9.3 A second-order low-pass filter

9.2.2 Second-Order Low-Pass Filter

A second-order low-pass filter can be made with three transconductance amplifiers (A_1 , A_2 , A_3 as shown in Figure 9.3) and two capacitors. The transfer function of this filter is given in the Laplace domain by

$$H(s) = \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{1}{1 + \tau s/Q + (\tau s)^2}, \quad (9.7)$$

where $s = j\omega$, $j^2 = -1$, and ω is the angular frequency, the time constant $\tau = C/g_m$ when both A_1 and A_2 have a conductance g_τ and both capacitors have capacitance C . The quality factor Q of the filter can be expressed as

$$Q = \frac{1}{2 - g_Q/g_\tau} \quad (9.8)$$

where g_Q is the conductance of the amplifier A_3 . The gain and phase response of the filter described by Eq. (9.7) are shown in Figure 9.4 for two values of Q .

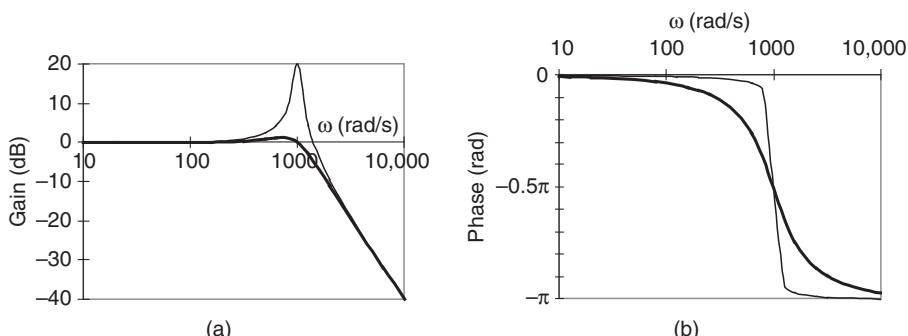


Figure 9.4 (a) Gain and (b) phase response of the second-order low-pass filter for $Q = 1$ (heavy line) and $Q = 10$ when $1/\tau = 1000 \text{ s}^{-1}$

9.2.3 Stability of the Filter

From Eq. (9.8), it is clear that Q becomes infinite when $g_Q = 2g_\tau$, and from Eq. (9.7) we can then see that when $\omega = 1/\tau$, the gain of the filter also becomes infinite and the filter will be unstable. This gives $g_Q < 2g_\tau$ as the small-signal stability limit of the filter. However, the filter of Figure 9.3 also has a large-signal stability limit, which puts a more stringent constraint on the value of Q . In order to obtain the transfer function of Eq. (9.7), we have treated the filter as a linear system. This approximation is only valid for small input signals. It has been shown by Mead (1989) that large transient input signals can create a sustained oscillation in this filter. During most of this oscillation, all three amplifiers are saturated so that their output is either plus or minus their bias current. We can therefore adopt a piece-wise linear approach to analyze this situation in which we treat the amplifiers as current sources. The following analysis is mostly adapted from Mead's, but is more general since it does not assume that the amplitude of the oscillation is equal to the supply voltage.

When V_{in} suddenly increases by a large amount, the amplifier A_1 will saturate and will charge the capacitor at its output with its maximum output current I_τ . If, at the same time, V_1 is larger than V_{out} , A_3 will also charge the capacitor with its maximum output current I_Q and we can write for V_1 :

$$\frac{dV_1}{dt} = \frac{I_Q + I_\tau}{C}, \quad (9.9)$$

where $V_{\text{out}} \ll V_1 \ll V_{\text{in}}$. V_1 will thus rise at its maximum rate. Once V_1 catches up with V_{in} , the output current of A_1 changes sign and we write for V_1 :

$$\frac{dV_1}{dt} = \frac{I_Q - I_\tau}{C}, \quad (9.10)$$

where $V_{\text{out}} \ll V_{\text{in}} \ll V_1$.

In order to have Q larger than one, I_Q has to be larger than I_τ so that in this case V_1 will continue to increase with a smaller slope until it reaches the positive power supply V_{dd} or until V_{out} catches up with V_1 . As long as V_1 stays larger than V_{out} , we can write in our piece-wise linear approach for V_{out} :

$$\frac{dV_{\text{out}}}{dt} = \frac{I_\tau}{C}, \quad (9.11)$$

where $V_{\text{out}} \ll V_1$.

Once V_{out} catches up with V_1 , the sign of the output of A_3 will change, and V_1 will start its steep descent, until V_1 goes below V_{in} , when the sign of the output of A_1 changes and V_1 descends more slowly to the negative power supply V_{ss} , or until V_{out} catches up again. Figure 9.5 sketches the behavior of the circuit according to the above equations. The thick line shows the evolution of V_1 and the thin line shows the same for V_{out} . Whenever V_{out} catches up with V_1 , the change in both voltages will switch direction.

By comparing the voltages at which V_{out} catches up with V_1 at the start and the end of a single rise and fall cycle, we can determine the nature of the oscillation. If ΔV (see Figure 9.5) is positive, the amplitude of the oscillation will decrease during each period and the oscillation

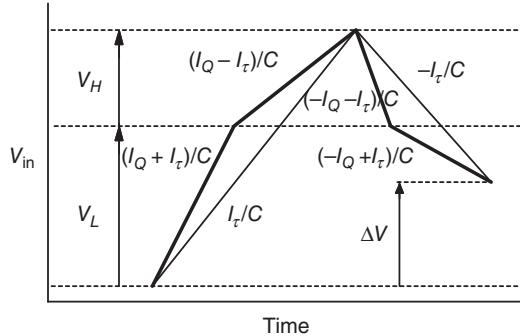


Figure 9.5 Piece-wise linear approximation of the waveform for V_1 (bold) and V_{out} . The slope of each line is indicated next to it

will cease after a certain time. The limit of stability is reached when ΔV becomes zero, so that the amplitude of the oscillation stays constant. For the rising part of the oscillation we write:

$$\frac{C}{I_Q + I_\tau} V_L + \frac{C}{I_Q - I_\tau} V_H = \frac{C}{I_\tau} (V_L + V_H), \quad (9.12)$$

where V_L and V_H are as shown in Figure 9.5. Similarly, for the falling part, we write when ΔV equals zero:

$$\frac{C}{I_Q + I_\tau} V_H + \frac{C}{I_Q - I_\tau} V_L = \frac{C}{I_\tau} (V_L + V_H). \quad (9.13)$$

Equations (9.12) and (9.13) can only be satisfied when $V_L = V_H$. Substituting V for V_L and V_H in either equation, and dividing by CV/I_τ yields

$$\frac{1}{I_Q/I_\tau + 1} + \frac{1}{I_Q/I_\tau - 1} = 2. \quad (9.14)$$

Rewriting this equation, we obtain the following solution:

$$\frac{I_Q^2}{I_\tau^2} - \frac{I_Q}{I_\tau} - 1 = 0 \Rightarrow \frac{I_Q}{I_\tau} = \frac{1 + \sqrt{5}}{2} \approx 1.62. \quad (9.15)$$

This gives the critical value for large-signal stability of the low-pass filter of Figure 9.3. Since the conductance of the amplifiers is directly proportional to the bias currents, this large-signal stability condition also limits g_Q/g_τ to this value, and thus limits Q to a maximum value of 2.62 (see Eq. 9.8). The large-signal stability limit therefore severely limits the maximum quality factor of the filter when using the basic transconductance amplifier.

9.2.4 Stabilised Second-Order Low-Pass Filter

The circuit can be improved by using two wide-range transconductance amplifiers to implement A_1 and A_2 (Figure 9.6) and a basic transconductance amplifier for A_3 (Watts et al. 1992) (see Figure 9.1b). In this case we can write for the conductance ratio:

$$\frac{g_Q}{g_\tau} = (n + 1) \frac{I_Q}{I_\tau} \quad (9.16)$$

which ensures that g_Q/g_τ becomes 2, that is, Q becomes infinite, before I_Q/I_τ becomes 1.62, since n is larger than 1. Thus the filter is always large-signal stable whenever it is small-signal stable.

With the two wide-range transconductance amplifiers A_1 and A_2 , and the one basic transconductance amplifier A_3 , we have a second-order low-pass filter for which we can set the cutoff frequency and the quality factor using the bias currents of these amplifiers. By cascading these filters and biasing the amplifiers with exponentially decreasing currents, we can then create a model of the basilar membrane. The limited input linear range of the second-order LPF in Figure 9.6 can be increased by modifying the amplifiers A_1 and A_2 so that the inputs go to the well terminals instead of the gate terminals of the input differential pair transistors (Sarpeshkar et al. 1997).

9.2.5 Differentiation

The voltage V_{out} (Figure 9.6) at the output of each second-order stage in the cochlear filter cascade represents the displacement of a small section of the basilar membrane. However, since the stimulation of the inner hair cells in the biological cochlea is proportional to the velocity of the basilar membrane, the output of each second-order stage has to be differentiated. This can be done by creating a copy of the output current I_{dif} of amplifier A_2 at every stage as in Watts et al. (1992). Since the voltage on a capacitor is proportional to the integral of the current onto the capacitor, I_{dif} is effectively proportional to the basilar membrane velocity. Yet, with equal displacement amplitudes, velocity will be much larger for high frequencies than for low frequencies, yielding output signals with amplitudes that decrease from the beginning of the cochlea to the end. This can be corrected by normalizing I_{dif} to give equal amplitude at

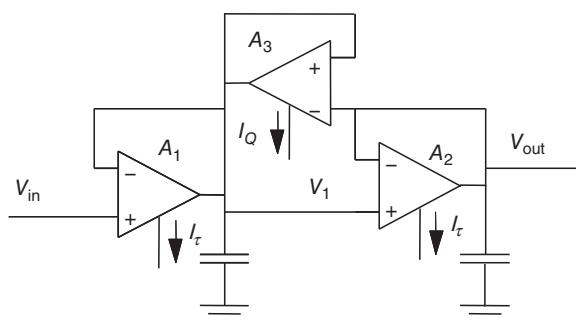


Figure 9.6 Modified second-order low-pass filter

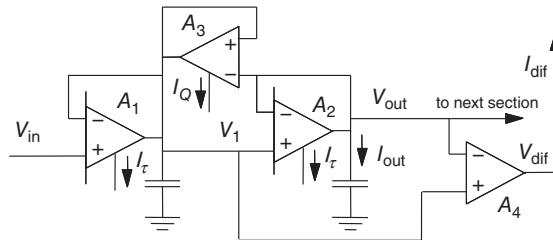


Figure 9.7 One section of the cochlear cascade, with differentiator Adapted from van Schaik et al. (1996). Reproduced with permission of MIT Press

every output. A resistive line controlling the gain of the current mirrors that create the copies of I_{dif} at each stage is used for this purpose by Watts et al. (1992). However, this resistive line introduces an extra source of mismatch in the circuit.

An alternative solution which does not need normalization is to take the difference between V_{out} and V_1 (see Figure 9.7). We can rewrite Eq. (9.3) applied to A_2 as

$$g_\tau(V_1 - V_{\text{out}}) = I_{\text{out}} \quad (9.17)$$

or

$$V_1 - V_{\text{out}} = \frac{I_{\text{out}}}{g_\tau} = \frac{sCV_{\text{out}}}{g_\tau} = \tau s V_{\text{out}}. \quad (9.18)$$

This is equivalent to differentiating V_{out} , with 0 dB gain at the cutoff frequency for all stages. Figure 9.8 shows the gain and phase response of the filter after differentiation. We can see that a single band-pass filter only has a shallow high frequency cutoff slope of 20 dB per decade. In the filter cascade, however, the 40 dB per decade cut-off slopes of the individual low-pass filters will be accumulated (Figure 9.4). This can yield very steep high frequency cut-off slopes, as we will see in the measurements later on in Figure 9.12.

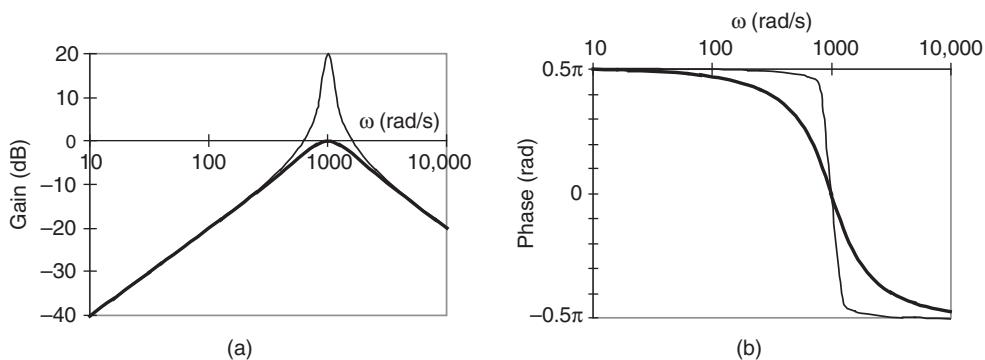


Figure 9.8 (a) Gain and (b) phase response of the second-order low-pass filter with differentiator for $Q = 1$ (heavy line) and $Q = 10$ when $1/\tau = 1000 \text{ s}^{-1}$

A current output I_{dif} can be taken from the output of the section by adding an additional transconductance amplifier A_4 as shown in Figure 9.7, which ensure that:

$$I_{\text{dif}} = g_m(V_1 - V_{\text{out}}) = g_m \tau s V_{\text{out}}, \quad (9.19)$$

where g_m is the transconductance of this amplifier.

9.3 Current-Domain Second-Order Filter

In this section, we introduce the log-domain circuits used in the implementation of the 2D silicon cochlea presented in Hamilton et al. (2008a, 2008c). While these circuits are not specific to the 2D silicon cochlea, they provide a good introduction to log-domain filters and to operating circuits in the current mode; a good counterpoint to the circuits presented in Section 9.2.

9.3.1 The Translinear Loop

Log-domain filters are based on translinear loops. The translinear loop is a fundamental concept in log-domain circuit design (Gilbert 1975). It is described in Gilbert (1990) as follows:

In a closed loop containing an even number of forward-biased junctions, arranged so that there are an equal number of clockwise-facing and counter-clockwise-facing polarities, the product of the current-densities in the clockwise direction is equal to the product of the current densities in the counter-clockwise direction.

This concept is based on the relationship

$$e^A \times e^B = e^{A+B} \quad (9.20)$$

and, as such, is only valid when current through a device is an exponential function of the voltages at the terminals of the device, as is the case for BJTs and MOSFETs operating in the sub-threshold region. This concept can be best described when looking at the circuit in Figure 9.9.

In Figure 9.9, M_1 and M_3 comprise the counter-clockwise junctions while M_2 and M_4 comprise the clockwise junctions. Transistor M_5 is necessary to correctly bias the source

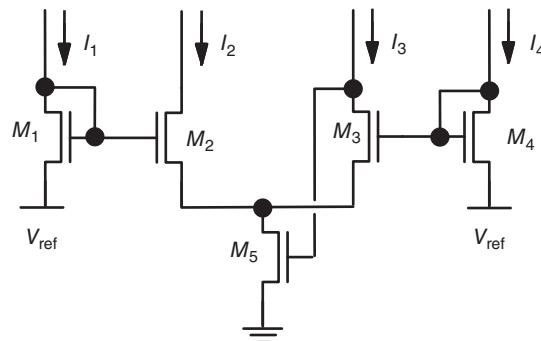


Figure 9.9 A translinear multiplier

voltage of both M_2 and M_3 and sink the current running through M_2 and M_3 . Hence, given the definition above we can say:

$$I_1 I_3 = I_2 I_4 \quad (9.21)$$

or, defining the current through M_2 as the output of the circuit, we can write:

$$I_2 = \frac{I_1 I_3}{I_4}. \quad (9.22)$$

Equation (9.21) demonstrates the ease with which a multiplier circuit can be constructed when operating in the log-domain. Understanding the principle of translinear loops can also be an important tool when analyzing the function of other log-domain circuits.

Tau Cell

The tau cell (van Schaik and Jin 2003) is a basic building block representing a class of log-domain filters. A schematic of the tau cell is shown in Figure 9.10. The tau cell is designed for complete programmability via the time constant, τ , and the current feedback, $A_i v$. It can be used as a building block to create a number of more complex, higher order filters.

The tau cell is based on the principle of translinear loops and its core structure is identical to that of Figure 9.9. In Figure 9.10, the closed loop of gate-source junctions necessary to form a translinear loop is created by transistors M_1 to M_4 and hence,

$$I_{i-1} I_0 = I_{M2} I_i. \quad (9.23)$$

The capacitor C introduces dynamics into the translinear loop resulting in a filter. The current through the capacitor is given by

$$I_C = C \frac{dV_C}{dt} = I_{M2} + I_{M3} - 2I_0 + AI_0 - \frac{AI_0 I_{i+1}}{I_i}. \quad (9.24)$$

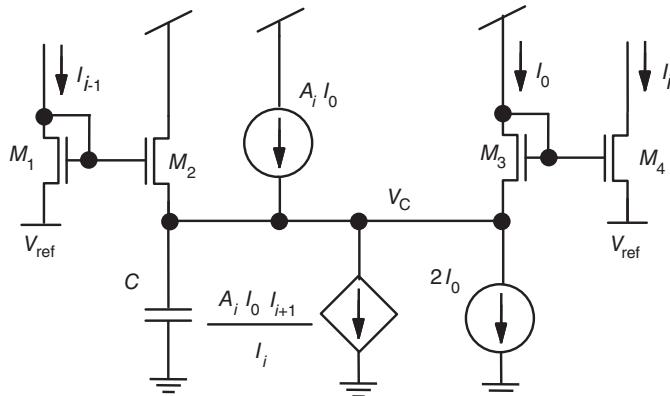


Figure 9.10 The tau cell

If we assume that the gate capacitance of M_3 and M_4 is significantly smaller than C , that is, if we assume that the dynamics at the gate of M_3 is much faster than at source of M_3 , we can simplify Eq. (9.24) with $I_{M3} = I_0$ and using Eq. (9.23) to eliminate I_{M2} :

$$I_C = C \frac{dV_C}{dt} = \frac{I_{i-1}I_0}{I_i} - I_0 + AI_0 - \frac{AI_0I_{i+1}}{I_i}. \quad (9.25)$$

In addition, since M_3 and M_4 have the same gate voltage and assuming the voltage at the drain of M_4 is at least 100 mV larger than V_{ref} , we can write:

$$I_i = I_0 e^{\frac{V_C - V_{\text{ref}}}{U_T}} \quad (9.26)$$

Equation (9.26) implies that

$$\frac{dI_i}{dt} = \frac{I_0}{U_T} e^{\frac{V_C - V_{\text{ref}}}{U_T}} \frac{dV_C}{dt} = \frac{I_i}{U_T} \frac{dV_C}{dt} \quad (9.27)$$

We can use this result to eliminate V_C from Eq. (9.25):

$$\frac{CU_T}{I_0} \frac{dI_i}{dt} = I_{i-1} - (1 - A)I_i - AI_{i+1} \quad (9.28)$$

In the Laplace domain, the transfer function for a single tau cell can thus be written as

$$T_i = \frac{I_i}{I_{i-1}} = \frac{1}{[\tau_i s + 1 - A_i] + A_i T_{i+1}}, \quad (9.29)$$

where $\tau = \frac{CU_T}{I_0}$ is the time constant, τ_i is the time constant of stage i , I_{i-1} is the input current, I_i is the output current of stage i and $I_{i+1} = T_{i+1}I_i$. If there is no next stage then $T_{i+1} = 0$ and $A_i = 0$ by definition.

9.3.2 Second-Order Tau Cell Log-Domain Filter

A second-order low-pass filter can be realized by connecting two tau cells as illustrated in Figure 9.11. Here we see that the first cell has a feedback gain A_1 , while the second tau cell has no feedback. The current feedback, $A_1 I_0 I_{i+1} / I_i$, can be implemented using the multiplier

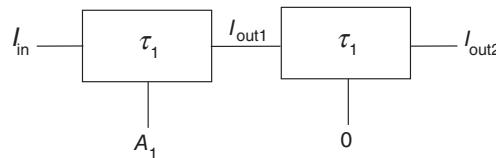


Figure 9.11 Second-order tau cell filter structure

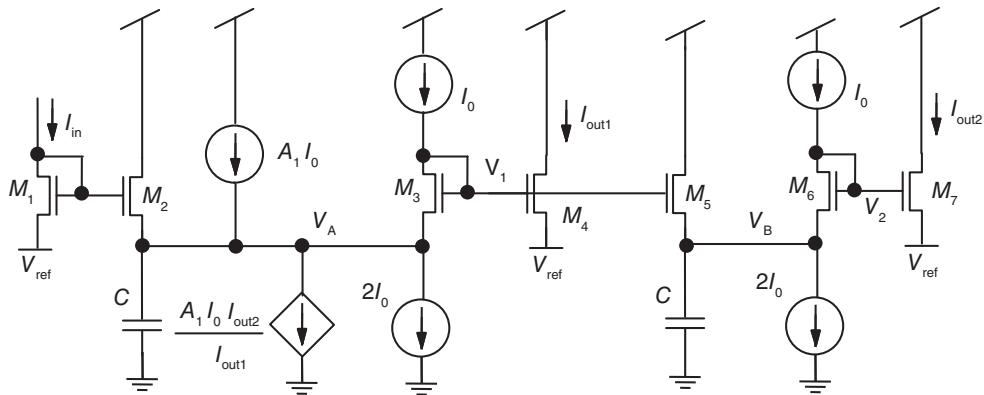


Figure 9.12 Second-order tau cell low-pass filter

of Figure 9.9. A more optimized method to implement the current feedback is described in Hamilton et al. (2008b). The schematic view of the second-order low-pass filter is given in Figure 9.12. The general equation for the second-order low-pass filter in Figure 9.12 is given by

$$T(s) = \frac{I_{\text{out}2}}{I_{\text{in}}} = \frac{1}{\tau^2 s^2 + \frac{\tau s}{Q} + 1}, \quad (9.30)$$

where τ is the time constant and Q is the quality factor.

In order to construct a band-pass filter similar to that described in Section 9.2.5, the output from the second tau cell in Figure 9.12, $I_{\text{out}2}$, must be subtracted from the output of the first tau cell in Figure 9.12, $I_{\text{out}1}$, such that

$$\frac{I_{\text{out}}}{I_{\text{in}}} = \frac{I_{\text{out}1} - I_{\text{out}2}}{I_{\text{in}}} = \frac{s\tau + 1}{\tau^2 s^2 + \frac{\tau s}{Q} + 1} - \frac{1}{\tau^2 s^2 + \frac{\tau s}{Q} + 1} = \frac{s\tau}{\tau^2 s^2 + \frac{\tau s}{Q} + 1}, \quad (9.31)$$

where I_{out} is the output from the band-pass filter. Figure 9.13 shows a band-pass filter structure that is described by Eq. (9.31).

9.4 Exponential Bias Generation

Since there is an exponential relationship between position along the basilar membrane and best frequency in the real cochlea, we will need to use filters with exponentially decreasing cut-off frequencies in our model. In all the silicon cochlear models mentioned in Chapter 4, the exponential dependency is obtained using a linear decreasing voltage on the gates of MOS transistors operating in weak-inversion. In weak-inversion, the drain current of a saturated nFET with its source tied to the bulk and its gate voltage referred to the same bulk can be expressed by

$$I_D = I_S e^{V_G - V_{TO}/nU_T} \quad (9.32)$$

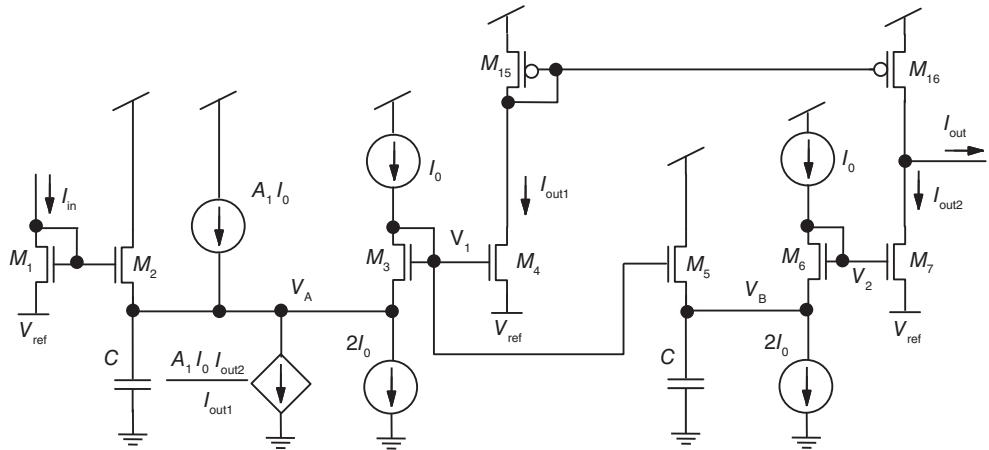


Figure 9.13 A second-order tau cell band-pass filter

with I_S as defined in Eq. (9.2) and V_{T0} the threshold voltage of the transistor. This shows that the drain current depends exponentially on the gate voltage. A spatial voltage distribution which decreases linearly with distance is easily created using a resistive polysilicon line; if there is a voltage difference between the two ends of the line, the voltage on the line will decrease linearly all along its length. It is therefore possible to create a filter cascade with an exponentially decreasing cut-off frequency by biasing the amplifiers of Figure 9.6 using MOS transistors whose gates are connected by equal lengths of the polysilicon line (Lyon and Mead 1988). As we can see in Eq. (9.32), however, the drain current also depends exponentially on the threshold voltage and small variations in V_{T0} will introduce large variations in the drain current. Because both the cut-off frequency and the quality factor of the filters are proportional to these drain currents, large parameter variations are generated by small V_{T0} variations. A root mean square (RMS) mismatch of 12% in the drain current of two identical transistors with equal gate and source voltages is not exceptional (Vittoz 1985), even when sufficient precautions are taken. We can circumvent this problem by using CMOS compatible lateral bipolar transistors (CLBTs) as bias transistors. A CLBT is obtained if the drain or source junction of a MOS transistor is forward-biased in order to inject minority carriers into the local substrate. If the gate voltage is negative enough (for an n-channel device), then no current can flow at the surface and the operation is purely bipolar (Arreguit 1989; Vittoz 1983). Figure 9.14 shows the major flows of current carriers in this mode of operation, with the source, drain, and well terminals renamed emitter E, collector C, and base B.

Since there is no p+ buried layer to prevent injection to the substrate, this lateral npn bipolar transistor is combined with a vertical npn. The emitter current I_E is thus split into a base current I_B , a lateral collector current I_C , and a substrate collector current I_{Sub} . Therefore, the common-base current gain $B = -I_C/I_E$ cannot be close to 1. However, due to the very small rate of recombination inside the well and to the high emitter efficiency, the common-emitter current gain $E = I_C/I_B$ can be large. Maximum values of E and B are obtained in concentric structures using a minimum size emitter surrounded by the collector and a minimum lateral

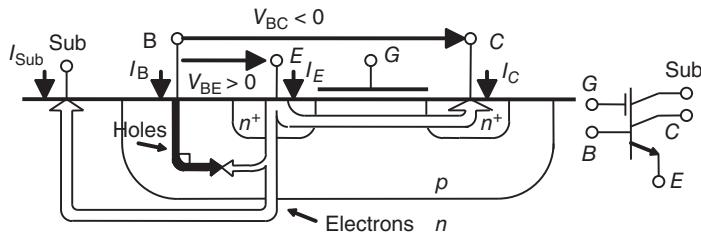


Figure 9.14 Bipolar operation of the MOS transistor: carrier flows and symbol. Adapted from van Schaik et al. (1996). Reproduced with permission of MIT Press

base width. For $V_{CE} = V_{BE} - V_{BC}$ larger than a few hundred millivolts, this transistor is in active mode and the collector current is given, as for a normal bipolar transistor, by

$$I_C = I_{Sb} e^{V_{BE}/U_T}, \quad (9.33)$$

where I_{Sb} is the specific current in bipolar mode, proportional to the cross section of the emitter-to-collector flow of carriers. Since I_C is independent of the MOS transistor threshold voltage V_{T0} , the main source of mismatch of distributed MOS current sources is suppressed, when CLBTs are used to create the current sources. A disadvantage of the CLBT is its low early voltage, that is, the device has a low output resistance. Therefore, it is preferable to use a cascode circuit as shown in Fig. 9.15a. This yields an output resistance several hundred times larger than that of the single CLBT; whereas the area penalty, in a layout as shown in Figure 9.15b, is acceptable (Arreguit 1989).

Another disadvantage of CLBTs, when biased using a resistive line, is their base current, which introduces an additional voltage drop on the resistive line. However, since the cut-off frequencies in the cochlea are controlled by the output current of the CLBTs and since these cut-off frequencies are relatively small (typically 20 kHz or less), the output current of the CLBTs will be small. If the common-emitter current gain E is much larger than 1, the base current of these CLBTs will be very small compared to the current flowing through the

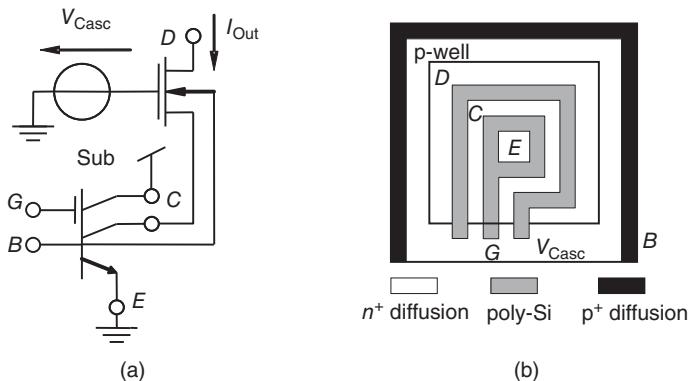


Figure 9.15 CLBT cascode circuit (a) and its layout (b). Adapted from van Schaik et al. (1996). Reproduced with permission of MIT Press

resistive line and the voltage error introduced by the small base currents will be negligible. Furthermore, since the cut-off frequencies of the cochlea will typically span two decades with an exponentially decreasing cut-off frequency from the beginning to the end, only the first few filters will have any noticeable influence on the current drawn from the resistive line.

9.5 The Inner Hair Cell Model

As shown in Section 9.2.5 and Figure 9.7, both the voltage-domain and the current-domain silicon cochleae typically have a current output signal for each section representing the velocity of the BM vibration. This current then serves as the input to an IHC circuit, modeling the transduction from vibration to a neural signal. There have been a number of IHC circuit models developed since the first, proposed in Lazzaro and Mead (1989). The more complex circuit models, such as the one presented in McEwan and van Schaik (2004), aim to reproduce the IHC behavior in great detail, including all the various time constants. Here we discuss the IHC model proposed in Chan et al. (2006) that models the two main properties of the IHC, namely, half-wave rectification and low-pass filtering. The circuit for this IHC model is shown in Figure 9.16.

In this circuit a current which represents the band-pass output of a cochlea section, I_c , is half-wave rectified by a current mirror. A DC offset can be added by using V_{off} to set the current I_{off} . This half wave-rectified current is, as a first approximation, given by

$$I_{HWR} = \max(0, I_c + I_{off}). \quad (9.34)$$

The output of the half-wave rectifier is passed through a first-order log-domain low-pass filter that is based on the tau cell of Figure 9.10, except that it uses pFETs rather than nFETs. The transfer function is nonetheless the same and is given in the Laplace domain by

$$\frac{I_{IHC}}{I_{HWR}} = \frac{G}{s\tau + 1} \quad (9.35)$$

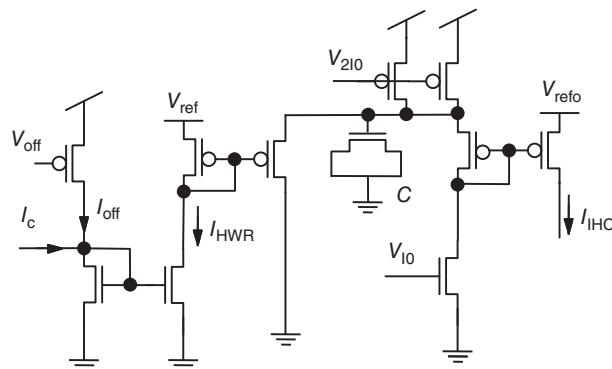


Figure 9.16 The inner hair cell circuit

where G is given by

$$G = e^{\frac{(V_{ref0} - V_{ref})}{U_T}}. \quad (9.36)$$

The time constant of the low-pass filter is given $\tau = \frac{C U_T}{I_0}$, where C is modeled by an MOS capacitor. The cut-off is set around 1 kHz as in the biological IHC, modeling the reduction in phase-locking observed on real auditory nerves at frequencies greater than 1kHz. The two control signals V_{ref} and V_{ref0} are slightly below V_{dd} to allow the two pFETs providing $2I_0$ to operate in saturation. Any voltage difference between V_{ref} and V_{ref0} will show up as a current gain depending exponentially on this difference as given in Eq. (9.36).

As shown in Chapter 4, the biological IHC exhibits adaptation to an ongoing stimulus. As a result it responds more strongly to the onset rather than the sustained part of a stimulus. This is facilitated by temporarily suppressing its response after the offset of stimulation. This adaptation can be directly modeled in a more complex IHC circuit, as in McEwan and van Schaik (2004), or it can be emulated by using the output of the IHC to stimulate a neuron with an adaptive threshold. In the second case it is actually the neuron that has a stronger response to the onset of a sound, rather than the IHC, but the end result is very similar. Silicon neurons are discussed in great detail in Chapter 7.

9.6 Discussion

In this chapter we have described circuits for the main building blocks of the silicon cochlea models described in Chapter 4. As discussed in Chapter 4, there are still many open issues in analog silicon cochlea design today; for example, whether the operating domain of these circuits should be current or voltage. The circuits in this chapter face the same design challenges usually of analog designs: noise, dynamic range, and mismatch to name but a few. While current-mode circuits are usually more compact than voltage-mode circuits and have a larger dynamic range, they are more susceptible to variations in threshold voltages. To reduce the mismatch in the fabricated devices, transistors have to be sized properly and calibration circuits are needed. These tradeoffs are taken by the designer in the various cochlea designs. While the design of silicon cochleas has come a long way since the first implementation by Lyon and Mead (1988), we still have a long way to go to match the remarkable performance of biological cochleas. Several research groups worldwide are actively researching ways to get closer to this goal.

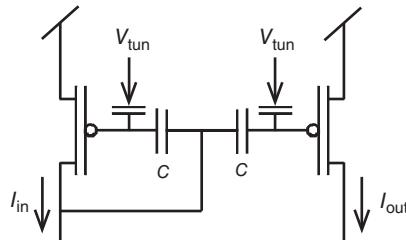
References

- Arreguit X. 1989. *Compatible Lateral Bipolar Transistors in CMOS Technology : Model and Applications*. PhD thesis Ecole Polytechnique Fédérale Lausanne, Switzerland.
- Chan V, Liu SC, and van Schaik A. 2006. AER EAR: a matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuits Syst. I: Special Issue on Smart Sensors* **54**(1), 48–59.
- Gilbert B. 1975. Translinear circuits: a proposed classification. *Electron Lett.* **11**, 14–16.
- Gilbert B. 1990. Current-mode circuits from a translinear viewpoint: a tutorial. In: *Analogue IC Design: The Current-Mode Approach* (eds Toumazou C, Lidgley FJ, and Haigh DG). Peter Peregrinus Ltd. pp. 11–93.
- Hamilton TJ, Jin C, van Schaik A, and Tapson J. 2008a. A 2-D silicon cochlea with an improved automatic quality factor control-loop. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1772–1775.

- Hamilton TJ, Jin C, van Schaik A, and Tapson J. 2008b. An active 2-D silicon cochlea. *IEEE Trans. Biomed. Circuits Syst.* **2**(1), 30–43.
- Hamilton TJ, Tapson J, Jin C, and van Schaik A. 2008c. Analogue VLSI implementations of two dimensional, nonlinear, active cochlea models. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 153–156.
- Lazzaro J and Mead C. 1989. Circuit models of sensory transduction in the cochlea. In: *Analog VLSI Implementations of Neural Networks* (eds Mead C and Ismail M). Kluwer Academic Publishers, pp. 85–101.
- Lyon RF and Mead CA. 1988. An analog electronic cochlea. *IEEE Trans. Acoust. Speech Signal Process.* **36**(7), 1119–1134.
- McEwan A and van Schaik A. 2004. An alternative analog VLSI implementation of the Meddis inner hair cell model. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 928–931.
- Mead CA. 1989. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.
- Sarpeshkar R, Lyon RF, and Mead CA. 1997. A low-power wide-linear-range transconductance amplifier. *Analog Integr. Circuits Signal Process.* **13**, 123–151.
- van Schaik A, Fragnière E and Vittoz E. 1996. Improved silicon cochlea using compatible lateral bipolar transistors. In: *Advances in Neural Information Processing Systems 11 (NIPS)* (eds. Touretzky DS, Mozer MC, and Hasselmo, ME). MIT Press, Cambridge, MA, pp. 671–677.
- van Schaik A and Jin C. 2003. The tau-cell: a new method for the implementation of arbitrary differential equations. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 569–572.
- Vittoz E. 1983. MOS transistors operated in the lateral bipolar mode and their application in CMOS technology. *IEEE J. Solid-State Circuits* **SC-24**, 273–279.
- Vittoz E. 1985. The design of high-performance analog circuits on digital CMOS chips. *IEEE J. Solid-State Circuits* **SC-20**, 657–665.
- Vittoz EA. 1994. Analog VLSI signal processing: why, where, and how? *J. VLSI Signal Process. Syst. Signal Image Video Technol.* **8**(1), 27–44.
- Watts L, Kerns D, Lyon R, and Mead C. 1992. Improved implementation of the silicon cochlea. *IEEE J. Solid-State Circuits* **27**(5), 692–700.

10

Programmable and Configurable Analog Neuromorphic ICs



A fundamental capability of any system, whether conventional or neuromorphic, is the ability to have long-term memories whether for program control, parameter storage, or general configurability. This chapter provides an overview of a programmable analog technology based on floating-gate circuits for reconfigurable platforms that can be used to implement such systems. It covers basic concepts of floating-gate devices, capacitor-based circuits, and charge modification mechanisms that underlie this configurable analog technology. It also discusses the extension of these techniques to program large arrays of floating-gate devices. The analog programmability afforded by this technology opens up possibilities to a wide range of programmable signal processing approaches (e.g., image processing) enabled through configurable analog platforms, such as large-scale field programmable analog arrays (FPAAs).

Parts of the text were taken from Hasler (2005). Reprinted with permission from IEEE.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

10.1 Introduction

Over the last decade, floating-gate circuit approaches have progressed from a few foundational academic results (Hasler et al. 1995; Shibata and Ohmi 1992) to a stable circuit and system technology with both academic and industrial applications. This programmable analog technology empowers analog signal processing approaches. An analog technology that is programmable can enable analog components to be seen as nearly as user-friendly as configurable digital options. This approach allows power-efficient computing for analog signal processing that is 1000–10,000 times more efficient than custom digital computation, making a range of portable applications not possible for over a decade a possibility today (Mead 1990).

The goal of this chapter is to develop general understanding of these programmable analog techniques. The next few sections detail the basic concepts of programmable analog technology. Section 10.2 discusses the basic concepts for floating-gate devices. Section 10.3 describes capacitor-based circuits, which are the basis of floating-gate circuit approaches. Section 10.4 describes the basic mechanisms for modifying the charge on a floating-gate device, and therefore making this analog technology programmable. Section 10.5 describes the techniques to extend these techniques to program an array of floating-gate devices, where each could be performing different computations.

In particular, many of these techniques are valuable, given the development and availability of large-scale field programmable analog arrays (FPAs) that enable applications for nonintegrated-circuit designers.

10.2 Floating-Gate Circuit Basics

Figure 10.1 shows the layout, cross section, and circuit symbol for the floating-gate p-channel FET (pFET) device (Hasler and Lande 2001). A floating gate is a polysilicon gate surrounded by silicon dioxide. Charge on the floating gate is stored permanently, providing a long-term memory, because it is completely surrounded by a high-quality insulator. The layout shows that the floating gate is a polysilicon layer that has no contacts to other layers. This floating gate can be the gate of a metal oxide semiconductor field effect transistor (MOSFET) and can be capacitively connected to other layers. In circuit terms, a floating gate occurs when there is no DC path to a fixed potential. No DC path implies only capacitive connections to the floating node, as seen in Figure 10.1.

The floating-gate voltage, determined by the charge stored on the floating gate, can modulate a channel between a source and drain, and therefore, can be used in computation. As a result, the floating-gate device can be viewed as a single transistor with one or several control gates where the designer controls the coupling into the surface potential. Floating-gate devices can compute a wide range of static and dynamic translinear functions by the particular choice of capacitive couplings into floating-gate devices (Minch et al. 2001).

10.3 Floating-Gate Circuits Enabling Capacitive Circuits

Floating-gate circuits provide IC designers with a practical, capacitor-based technology, since capacitors, rather than resistors, are a natural result of a MOS process. Programmable floating-gate circuit techniques, along with long-time retention characterization, have shown improved

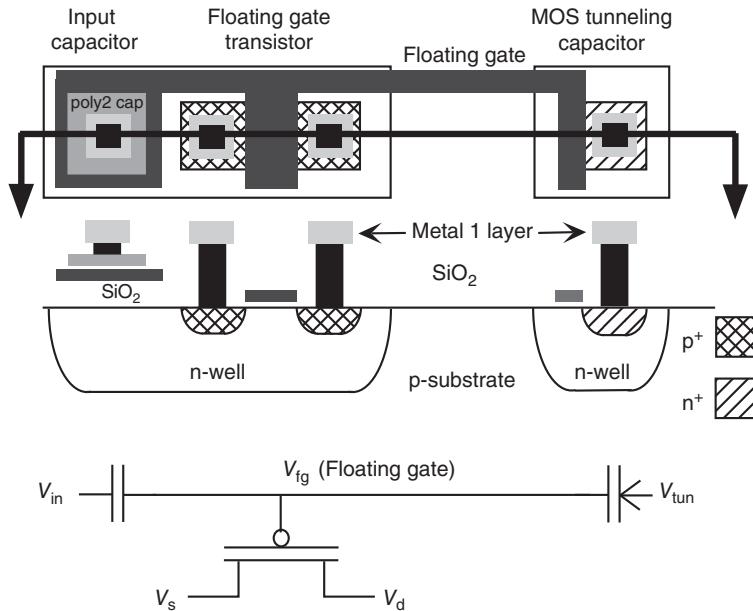


Figure 10.1 Layout, cross section, and circuit diagram of the floating-gate pFET in a standard double-poly, n-well process: The cross section corresponds to the horizontal line slicing through the layout view. The pFET transistor is the standard pFET transistor in the n-well process. The gate input capacitively couples to the floating gate by either a poly–poly capacitor, a diffused linear capacitor, or a MOS capacitor, as seen in the circuit diagram (not explicitly shown in the other two figures). Between V_{tun} and the floating gate is our symbol for a tunneling junction – a capacitor with an added arrow designating the direction of charge flow. © 2005 IEEE. Reprinted, with permission, from Hasler (2005)

performance for operational transconductance amplifiers (OTA) (Srinivasan et al. 2007), voltage references (Srinivasan et al. 2008), filters (Graham et al. 2007), and data converters (Ozalevli et al. 2008b) as well as a range of other analog circuit approaches. Figure 10.2 shows key basic capacitor circuit elements. Figure 10.2a shows the capacitive equivalent to a resistive voltage divider. The resulting expression is as expected from a capacitive divider, except that we have an additional voltage, V_{charge} , that is set by the charge (Q) at the output node, as $V_{charge} = Q/(C_1 + C_2)$. Figure 10.2b shows the capacitor circuit for feedback around an amplifier. As expected, the closed loop for this amplifier is $-C_1/C_2$; the output of this amplifier also has a voltage term due to the charge stored (Q) at the ‘−’ input terminal, where $V_{charge} = Q/C_2$.

Figure 10.2c shows a more realistic circuit model for the amplifier circuit in Figure 10.2b. This circuit description, which includes parasitic capacitances, is described as a single pole and zero system, assuming the amplifier sets a single pole (i.e., the amplifier has a frequency independent transconductance). In general, we can describe the amplifier as a transconductance amplifier, assuming the gain set by the capacitors is lower than the amplifier gain. Increasing C_w increases the input linear range; typically, C_w is larger than C_1 and C_2 , where C_w models the input capacitance of the amplifier, explicitly drawn capacitance, and parasitic capacitances. Increasing the function $C_w C_L / C_2$ proportionally increases the signal-to-noise ratio (SNR) (in

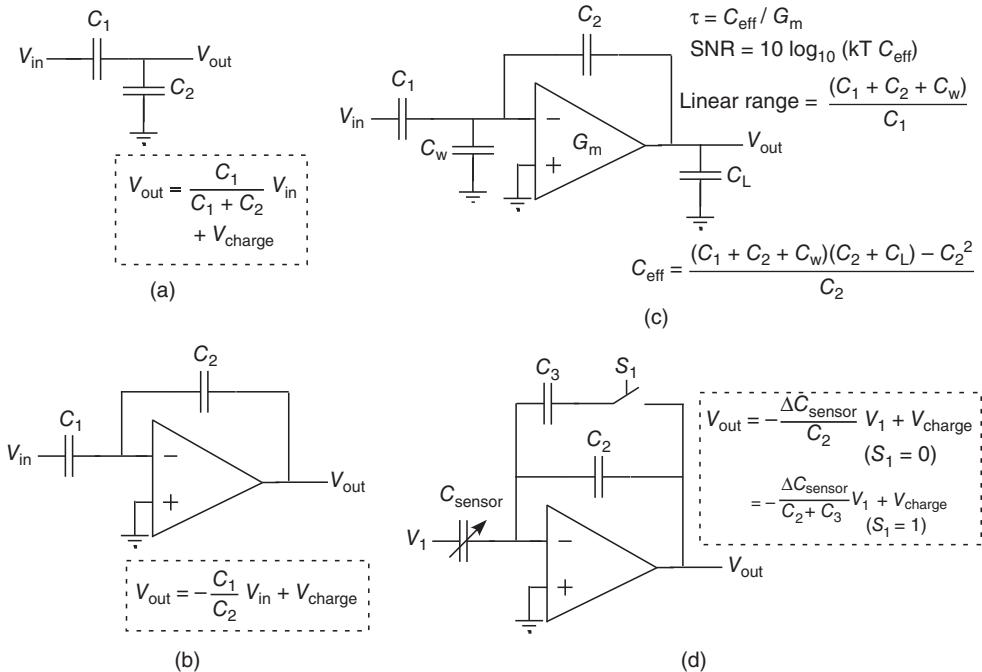


Figure 10.2 Basic capacitor circuits. (a) Capacitive divider circuit. (b) Capacitive feedback around an amplifier. The assumption is that the amplifier has MOS inputs; therefore inputs are practically only capacitive. (c) Assuming amplifier has a finite G_m , we identify the key parameters (bandwidth, signal-to-noise ratio, and input linear range) for this amplifier. (d) Circuit modification for direct measurement of capacitive sensors. © 2005 IEEE. Reprinted, with permission, from Hasler (2005)

signal power); therefore, unlike many filters where output noise and SNR are set by the load capacitance (kT/C thermal noise), this function allows for lower drawn capacitances for a given noise floor. When we improve the linear range of this amplifier, we simultaneously improve the SNR of the amplifier. These circuit approaches extend transconductance- C filter approaches to allow some parameters to be set by the ratio of capacitors (Graham et al. 2004), such as bandpass gain, the linear range, the noise level, and bandwidth. These approaches, coupled with the array programming techniques discussed in Section 10.4, result in an accurate low-power filter technology.

Figure 10.2d shows a slight extension of the other circuits toward measuring changes in capacitive sensors (i.e., microelectromechanical systems (MEMs) sensors). Analyzing this circuit with an ideal amplifier with gain A_v , we get

$$V_{out} = V_1 \frac{\Delta C_{sensor} + \frac{1}{A_v}(C_{sensor} + C_w)}{C_2},$$

$$\Delta V_{out} = V_1 \frac{\Delta C_{sensor}}{C_2}, \quad (10.1)$$

where C_w is the capacitance at the ‘-’ amplifier input, including the amplifier input capacitance. This circuit configuration attenuates the effect of sensor capacitance and C_w by the amplifier gain, an effect that is only a DC term, assuming V_1 remains constant. For example, for $C_{\text{sensor}} = 1 \text{ pF}$, maximum $\Delta C_{\text{sensor}} = 2 \text{ fF}$ (typical of some MEMs sensors), and $A_v = 1000$, we choose $C_2 = 20 \text{ fF}$ for a maximum V_{out} change of 1 V, resulting in an output offset voltage of 0.25 V. The constant part of C_{sensor} , as well as C_w , increases the linear range of the closed-loop circuit.

The circuit is still a first-order system (assuming a frequency independent amplifier with transconductance G_m) with its transfer function in the Laplace domain described by

$$\frac{V_{\text{out}}(s)}{\Delta C_{\text{sensor}}} = \frac{V_1}{C_2} \frac{1 - s(C_2/G_m)}{1 + s\tau}. \quad (10.2)$$

It has the same time constant (τ) as the amplifier in Figure 10.2c, and the zero, due to capacitive feedthrough, is typically at much higher frequency responses than the amplifier bandwidth. Typically, C_{sensor} and C_L (the output load capacitance, as in Figure 10.2c) are roughly equal in size (C_L might be larger), and are larger than C_2 . For the example above, with $C_L = C_{\text{sensor}} = 1 \text{ pF}$, the resulting bandwidth will be as shown in the following table:

Transconductance (G_m)	Bias current	Bandwidth
$1 (\text{k}\Omega)^{-1}$	$30 \mu\text{A}$	3 MHz
$10 (\text{k}\Omega)^{-1}$	$300 \mu\text{A}$	30 MHz

The resulting output noise (entire band) is

$$\hat{V}_{\text{out}} = \sqrt{qV_{\text{IC}}n \frac{(C_{\text{sensor}} + C_w)}{C_2 C_L}} \quad (10.3)$$

where n is the equivalent number of devices contributing to the amplifier noise, and V_{IC} is the ratio of the transconductance (G_m) to the differential pair transistor's bias current. For the example above, with a typical low-noise amplifier stage with input transistors operating with subthreshold currents result in 0.5 mV total noise, resulting in an SNR of roughly 66 dB between the maximum capacitor deflection and the minimum deflection. For our example circuit, the maximum capacitance change of 2 fF gives an output of 1 V, where a 1 aF change is at the 0 dB SNR level; by restricting the bandwidth of interest or making the amplifier bandwidth larger than the bandwidth of interest, the resulting sensitivity will increase. In practice, a bank of capacitors that can be switched into the circuit can be used to alter C_2 , and therefore the dynamic range and noise of these signals. Figure 10.2d shows the switching between gain levels; the switch is not at the charge storage node because an MOS switch on the ‘-’ terminal increases the leakage current at this node, decreasing hold time. These results have been experimentally verified through use of variable MEMs capacitor devices. In one particular system, a 100 aF capacitor change was observed resulting in 37.5 mV change for an amplifier with noise significantly less than 1 mV; therefore, 3 fF change resulted in a

1.13 V output swing and 3 aF change resulted in a 1 mV output swing (0 dB SNR point). The bandwidth of the amplifier was greater than 1 MHz.

10.4 Modifying Floating-Gate Charge

The floating-gate charge is modified by applying large voltages across a silicon–oxide capacitor to tunnel electrons through the oxide or by adding electrons using hot-electron injection. Although the physics of floating-gate circuits are discussed extensively elsewhere (Hasler et al. 1995, 1999; Kucic et al. 2001), they are briefly reviewed here.

10.4.1 Electron Tunneling

Charge is added to the floating gate by removing electrons using electron tunneling. Increasing the voltage across this tunneling capacitor, either by increasing the tunneling voltage (V_{tun}) or decreasing the floating-gate voltage, increases the effective electric field across the oxide, thereby increasing the probability of the electron tunneling through the barrier (Figure 10.3d). Starting from the classic model of electron tunneling, given as

$$I_{\text{tun}} = I_0 e^{(\mathcal{E}_0 t_{\text{ox}})/V_{\text{ox}}}, \quad (10.4)$$

where \mathcal{E}_0 is a fundamental parameter derived from a Wentzel–Kramers–Brillouin (WKB) solution of Schrödinger’s equation, t_{ox} is the thickness of the oxide dielectric, and V_{ox} is the voltage across the dielectric, we can derive an approximate model for the electron tunneling current around a given voltage across the oxide (tunneling voltage minus floating-gate voltage) as (Hasler et al. 1995, 1999)

$$I_{\text{tun}} = I_{\text{tun}0} e^{(V_{\text{tun}} - V_{\text{fg}})/V_x}, \quad (10.5)$$

where V_x is a tunneling device-dependent parameter that is a function of the bias voltage across the oxide.

10.4.2 pFET Hot-Electron Injection

pFET hot-electron injection is used to add electrons (remove charge) to the floating-gate. pFET hot-electron injection is used because it cannot be eliminated from a complementary metal oxide semiconductor (CMOS) process without adversely affecting basic transistor operation, and therefore will be available in all commercial CMOS processes. One might wonder how pFETs, where the current carriers are holes, inject hot electrons onto the floating gate. Figure 10.3e shows the band diagram of a pFET operating under bias conditions that are favorable for hot-electron injection. The hot-hole impact ionization creates electrons at the drain edge of the drain-to-channel depletion region, due to the high electric fields there. These electrons travel back into the channel region, gaining energy as they go. When their kinetic energy exceeds that of the silicon–silicon-dioxide barrier, they can be injected into the oxide and transported to the floating gate. To inject an electron onto a floating gate, the MOSFET must

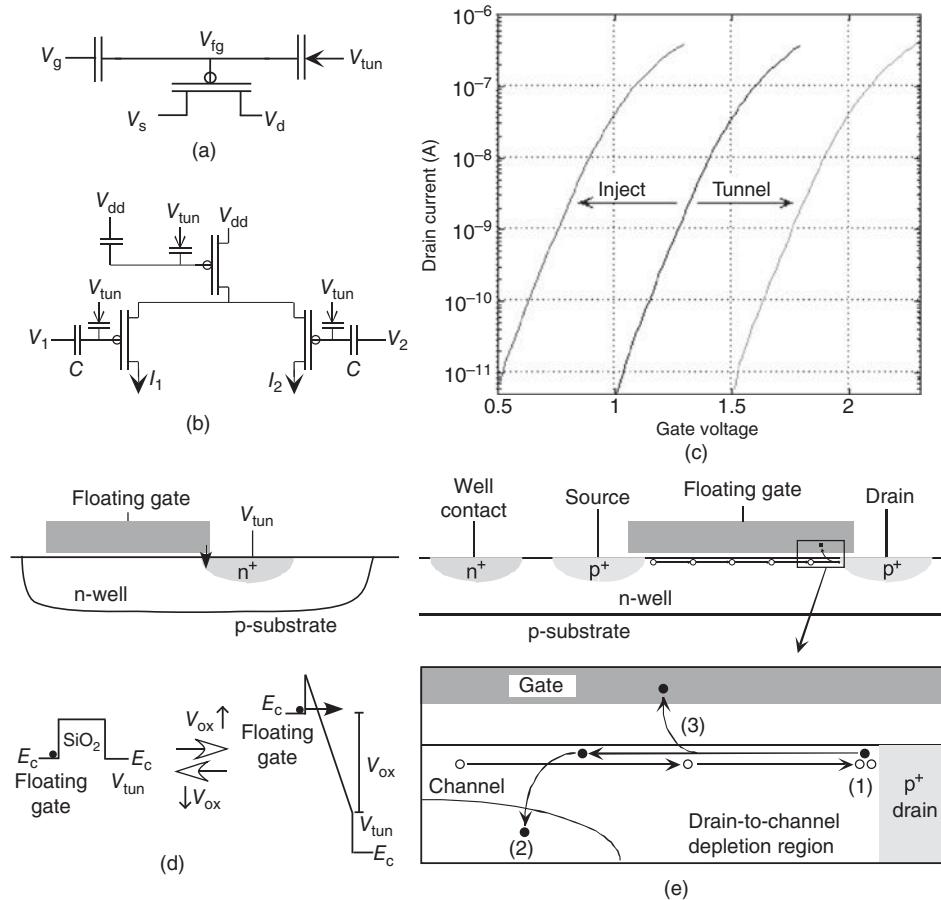


Figure 10.3 Approach to modifying floating-gate charge. (a) Basic circuit representation for a floating-gate device. (b) A programmable floating-gate differential pair. Both the input transistors as well as the current source transistors are programmed. Practically, it may be desirable to develop approaches to program the offset voltage for the amplifier, as well as the value for the current source. (c) Current-voltage curves from a programmed pFET transistor. We modify charge by a complimentary combination of electron tunneling (weaker pFET transistor) and hot-electron injection (stronger pFET transistor). (d) Basic picture of electron tunneling in $\text{Si}-\text{SiO}_2$ system (e) Basic picture of pFET hot-electron injection. Some holes moving through the channel gain sufficient energy to create an impact ionization event, generating electrons that increase in energy moving toward the channel. Some of these electrons will have sufficient energy to surmount the $\text{Si}-\text{SiO}_2$ barrier and arrive at the gate terminal. © 2005 IEEE. Reprinted, with permission, from Hasler (2005)

have a high electric field region ($>10 \text{ V}/\mu\text{m}$) to accelerate channel electrons to energies above the silicon–silicon-dioxide barrier, and in that region the oxide electric field must transport the electrons that surmount the barrier to the floating gate. As a result, the subthreshold MOSFET injection current is proportional to the source current, and is the exponential of a smooth function of the drain-to-channel potential (Φ_{dc}); the product of these two circuit variables

is the key aspect necessary to build outer-product learning rules. The first-principled model presented next is derived from basic physics that shows the resulting exponential functions derived in Duffy and Hasler (2003).

A simplified model for pFET injection that is useful for hand calculations relates the hot-electron injection current for a channel current (I_s) and drain-to-source (ΔV_{ds}) voltage as

$$I_{\text{inj}} = I_{\text{inj}0} \left(\frac{I_s}{I_{s0}} \right)^\alpha e^{-\Delta V_{ds}/V_{\text{inj}}}, \quad (10.6)$$

where $I_{\text{inj}0}$ is the injection current when the pFET is operating with a channel current reference (I_{s0}), where $I_s = I_{s0}$ at this reference current, and a drain-to-source voltage, V_{inj} is a device and bias dependent parameter, and α is $1 - \frac{U_T}{V_{\text{inj}}}$. Typical values for V_{inj} in a 0.5 μm CMOS process are 100–250 mV.

Choosing the appropriate model for simulation is critical for these devices. For example, when simulating a set of floating-gate devices that will be programmed, one typically does not need to implement the tunneling and injection currents, but rather make sure at the beginning of the simulation that the floating-gate voltages/bias currents are set correctly based upon the behavior of the particular programming scheme. In this mode of operation, one can set the floating-gate voltage through a very large resistor; for the total capacitance at a floating-gate node of 100 fF (a small device), a resistor of $10^{26} \Omega$ is consistent with the typical room temperature voltage drop of 4 μV over a 10-year period for a 10 nm oxide (Srinivasan et al. 2005). In some cases, transistor equivalent circuits can be used to simulate adaptive floating-gate elements, such as the capacitively coupled current conveyor (C^4) second-order section circuit and the synapse element (Graham et al. 2004; Srinivasan et al. 2005); these techniques tend to be useful circuits in their own right for applications requiring fast adaptation rates.

10.5 Accurate Programming of Programmable Analog Devices

The charge modification schemes, along with their detailed modeling, opens the door for accurate programming of a large number of floating-gate devices being utilized by a diverse set of circuits. Figure 10.4a shows the starting point for the story of automatically programming a large array of floating-gate elements. Figure 10.4a illustrates how programmable devices are accessed, which is defined here as **Prog** or program mode, and how computation is performed using these elements, which is defined as **Run** mode. Going from **Run** mode to **Prog** mode, means electrically reconfiguring all circuits such that each floating-gate device is configured into a two-dimensional mesh array of devices with the drain and gate lines moving in orthogonal directions. Individual elements are isolated (access to an individual gate and drain line) in a large matrix using peripheral control circuitry (Hasler and Lande 2001; Kucic et al. 2001). A standard technique like this is necessary when working with thousands and millions of floating-gate elements on a single die.

This programming scheme minimizes interaction between floating-gate devices in an array during the programming operation. Other elements are switched to a separate voltage to ensure that those devices will not inject. A device is programmed by increasing the output current using hot-electron injection, and erased by decreasing the output current using electron tunneling. Because of the poorer selectivity, tunneling is used primarily for erasing and for rough

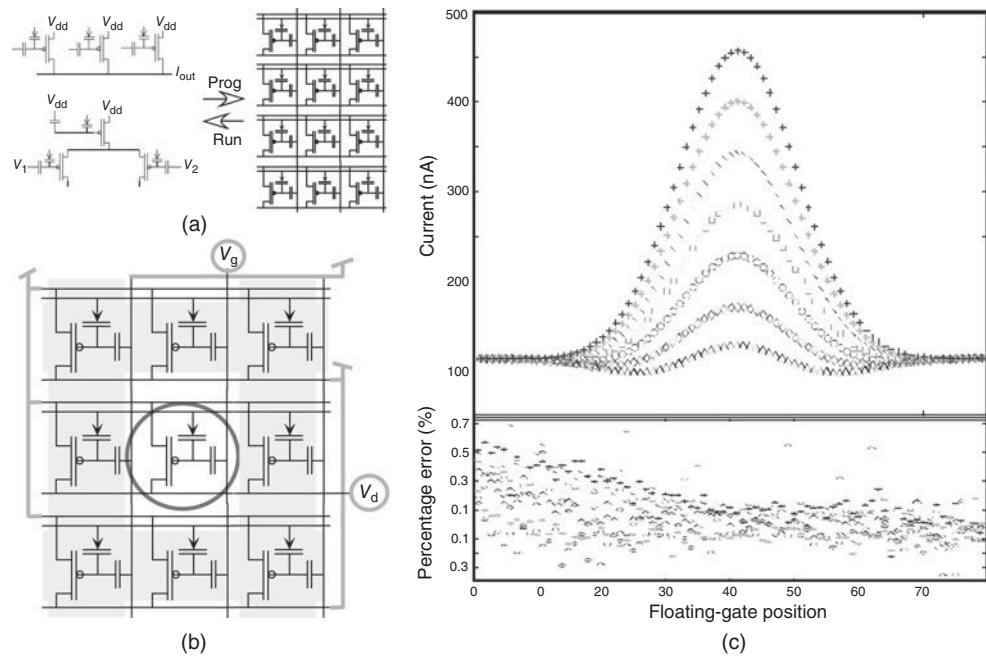


Figure 10.4 Programming of large number of floating-gate elements. (a) Infrastructure takes any number of floating-gate elements on a chip during **Run** mode, and reconfigures these devices into a regular array of floating-gate elements. (b) Hot electron injection requires both channel current (subthreshold) and high electric field; therefore in an array of devices a single element can be accessed using an ANDing scheme, both for measurement and for programming. (c) Experimental measurements for programming a sequence of functions of different amplitudes. The corresponding percentage error is plotted below the data; the experimental error (for this example bounded between 0.3% and 0.7%) does not correlate with the experimental waveform. © 2005 IEEE. Reprinted, with permission, from Hasler (2005)

programming steps. This scheme performs injection over a fixed time window (from 1 μ s to 10 s and longer) using drain-to-source voltage based on the actual and target currents. Most rapid programming techniques use pulse widths in the 10–100 μ s range, which potentially enable the programming of large arrays of floating-gate elements in mass production. Developing an efficient algorithm for pFET programming requires discussing the dependencies of the gate currents as a function of the drain-to-source voltage. This scheme also measures results at the circuit's operating condition for optimal tuning of the operating circuit (no compensation circuitry needed). Once programmed, the floating-gate devices retain their channel current in a nonvolatile manner. A custom programming board (PCB board) programs large floating-gate arrays around these standards (Kucic et al. 2001; Serrano et al. 2004), and on-going work is continuing to move all of these blocks on-chip using row-parallel programming techniques (Kucic 2004). These approaches have been used by over 40 researchers on hundreds of IC projects. The limiting factor for rapid programming of these devices is the speed and accuracy of the current measurements, rather than the hot-electron injection physics.

How accurately can these devices be programmed? In the end, the accuracy is limited by change in voltage (ΔV) due to one electron (q) moving off of the total capacitance (C_T) at the floating node, expressed as

$$\Delta V = q/C_T. \quad (10.7)$$

For C_T of 16 fF (a small device), the voltage change from one electron is 10 μ V. For a voltage source based on the floating-gate voltage for a 2 V swing, the accuracy is roughly 0.0005% or 17–18 bits. For a subthreshold current source, the accuracy is roughly 0.05% (11 bits) over the entire subthreshold current range (typically 6–8 orders of magnitude). The error decreases inversely proportional to an increasing C_T .

Therefore, the real question is how accurately can the programming algorithm achieve this theoretical limit. The first limitation is the accuracy of measuring the quantity to be programmed; for example, if we only have 8 bit accuracy to measure the output current we want to program, then we cannot expect to achieve significantly higher programming performance through that measurement. Second, is the limitation of the programming algorithm and associated circuitry, including parasitic elements; these approaches are designed to minimize the effect of parasitic elements by design. On the other hand, due to factors like capacitor mismatch, a fine-tuning programming step is often used to improve the effects due to these mismatches (Bandyopadhyay et al. 2005; Graham et al. 2004). Finally, C_T can set the thermal noise (kT/C noise) for the previous stage; a 16 fF capacitor will set a total noise for a single subthreshold device as 0.25 mV, an error which if not addressed in the resulting circuit, will be 25 times greater than the programming accuracy. Figure 10.4c shows measurements of programming accuracy from an array of floating-gate devices ($C_T \approx 100$ fF). Typical experimental results show that devices can be programmed within 0.5–0.1% accuracy over 3.5 decades of target currents, and devices can be programmed to at least as good as 1% accuracy over six decades of target currents, in CMOS processes ranging from 0.5 μ m to 0.25 μ m CMOS processes (Bandyopadhyay et al. 2006). More recently, the circuitry for programming arbitrary size floating-gate arrays has been fully integrated on chip throughout the pA to μ A range (Basu and Hasler 2011).

10.6 Scaling of Programmable Analog Approaches

The scaling down of the oxide thicknesses with CMOS processes makes one question the long-term potential of these analog approaches. From Eq. (10.4), we notice that for identical I_0 parameters, the tunneling leakage current is roughly constant for constant field scaling, that is, the power supply and oxide thickness scale are at the same rate, and therefore, these processes should all see minimal floating-gate charge loss at room temperature over a 10-year period. Unfortunately, these expressions are derived for a triangular barrier, and at low oxide voltages, voltages smaller than the equivalent 3.04 eV Si–SiO₂ barrier, we have a trapezoidal barrier, and the scaling no longer holds, an issue that becomes important for processes at 0.18 μ m and below. For typical 0.18 μ m processes, the SiO₂ oxide thickness is roughly 5 nm, which correspondingly is seen as the smallest oxide thickness that satisfies the typical 10-year lifetime requirements; most current electrically erasable programmable read only memory (EEPROM) processes use oxide thicknesses of 5 nm or larger.

The logical initial conclusion states that floating-gate circuits are not practical below 0.18 μm CMOS. Fortunately, there are multiple options for this technology to scale down to smaller processes. First, starting in the 0.35 μm and 0.25 μm CMOS processes, modern CMOS processes offer transistors with multiple oxide thicknesses. One application is developing circuits that can interface to larger external voltages (i.e., 5 V, 3.3 V, 2.5 V); these devices (both transistors and capacitors) will give at least one oxide layer at least as thick as 5 nm, and therefore allowing storage devices. These devices will be somewhat larger than the scaled down devices, but only in the active channel area, because the lithography improves the other transistor dimensions and parasitics as well. Because the overlap capacitance of the floating-gate field effect transistor (FET) effectively reduces its maximum gain, these devices do not need long channel lengths; designing for the minimum channel length or smaller is reasonable for these devices, even for precision analog circuits.

Further, the small oxide thicknesses in SiO_2 will be limited to roughly 1.3 nm before gate currents are too large for basic CMOS transistor operation. Therefore, the maximum improvement in capacitance will be a factor of three, and the maximum improvement in area will be a factor of nine. In practice, the area improvement will only be a factor of two less than it would be if all devices scale exactly with the process. Given that the capacitance typically sets the SNR for the device, this capacitance factor is less than a 1-bit loss of resolution compared to ideal scaling. Movement toward higher dielectric materials with low leakage in a given CMOS process will equally improve both standard digital CMOS as well as programmable analog CMOS. ('Leakage' here refers to transistor source-drain current when the transistor is off, not the substrate leakage; see Section 11.2.8.)

Second, small-oxide devices naturally enable adaptive floating-gate properties even within the typical process voltages. They can have an equilibrium between tunneling junctions and injection elements, as well as tunneling junction on the drain side, and directly obtain properties that have been demonstrated in the autozeroing floating-gate amplifier circuits (Hasler et al. 1996) and adaptive filter circuits (Kucic et al. 2001). The resulting gate leakage currents become a useful circuit property, not an effect to work around. Therefore, it can be concluded that not only will these programmable analog devices be available as CMOS processes scale to smaller dimensions, but they will be possible with a wider range of devices to choose from.

10.7 Low-Power Analog Signal Processing

The last three decades have seen the impact of rapid improvement in silicon IC processing and the accompanying digital processing resulting from this technology. This has resulted in a significant increase in processing and/or significant decrease in the power consumption. With the need for low-power portable and/or autonomous sensor systems, getting as much computation for a fixed power budget becomes more and more critical. Gene's law, named for Gene Frantz (Frantz 2000), postulates that the computational efficiency (operations versus power) for the specific case of digital signal processing microprocessors (Figure 10.5a) doubles roughly every 18 months due to advances using decreased feature size. Even with these remarkable improvements in digital computation, many digital signal processing algorithms are unreachable in real time within the power budget constraints required for many portable applications. Further, recent results (Marr et al. 2012) show that digital multiply and accumulate

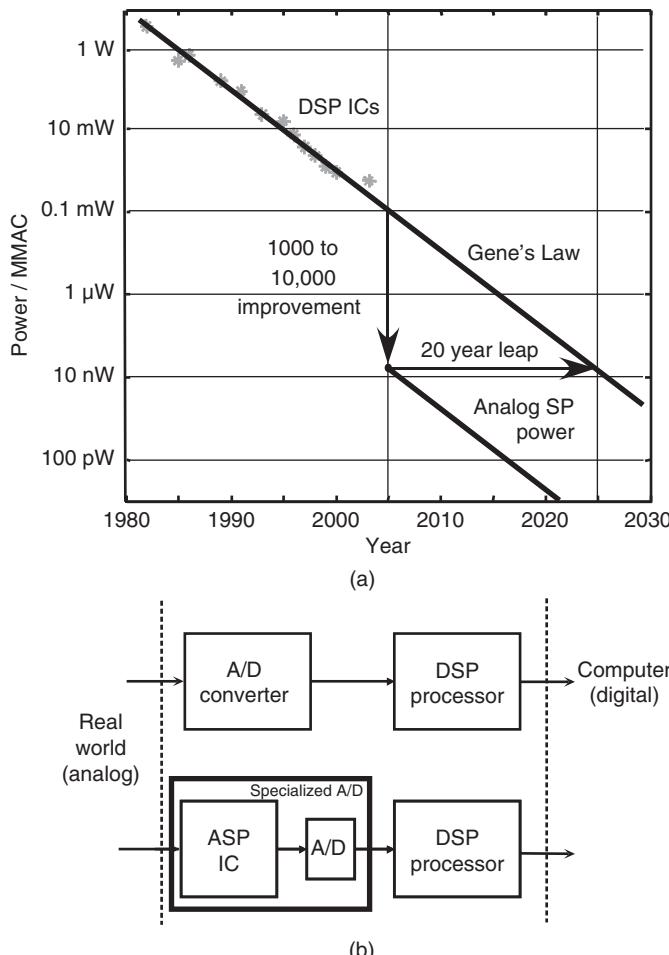


Figure 10.5 Motivation for using programmable analog technology. (a) Plot of computational efficiency (power consumption/millions of multiply-accumulate operations (MMAC)) for digital signal processing (DSP) microprocessors, the extrapolated fit to these data points (Gene's Law, Frantz 2000), and the resulting efficiency for a programmable analog system. The typical factor of 10,000 in efficiency improvement between analog and digital signal processing systems enables using low-power computational systems that might be available in 20 years. © 2005 IEEE. Reprinted, with permission, from Hall et al. (2005). (b) Illustration of the trade-offs in cooperative analog–digital signal processing. The question of where to put this boundary line between analog signal processing (ASP) and DSP strongly depends upon the particular requirements of an application

operations (MACs) are not improving computational efficiency with improved IC processes, further indicating the need for other approaches and improvements.

Mead (1990) hypothesized that analog computation could be a factor of 10,000 times more efficient than custom digital processing, for low to moderate (i.e., 6–12 bit SNR) resolution signals. Sarpeskar later quantitatively identified that the cost of analog systems will be lower

than the cost of digital systems for low to moderate SNR levels (Sarpeshkar 1997), SNR levels typical of most analog input (e.g., sensory) systems. Thus, the potential of analog signal processing has been seen for a while, but only recently become practical to implement.

This section presents a high-level overview of the range of available programmable analog signal processing technologies, where many of these technologies have proven Mead's original hypothesis. These approaches are enabled through a programmable analog technology in standard CMOS (Hasler and Lande 2001). These dense analog programmable and reconfigurable elements enable power efficient analog signal processing that is typically a factor of 1000–10,000 more efficient compared to custom digital approaches, as well as a precision analog design approach that scales with digital technology scaling. The result of this technology, shown graphically in Figure 10.5a, enables, in current technology, low-power computational systems that might be available through custom digital technology in 20 years, assuming digital performance scales at the same rate as over the past 20 years. This increase in efficiency can be used for reducing the power requirements of a given problem or addressing computational problems that are considered *intractable* by the current digital road map. The advance possible through programmable analog signal processing would be greater than the advances in DSP chips from the first marketed DSP chip to today's ICs.

The range of analog signal processing functions available results in many potential opportunities to incorporate these analog signal processing systems with digital signal processing systems for improved overall system performance. Figure 10.5b illustrates one trade-off between analog and digital signal processing. This model assumes the typical model of signals coming from real-world sensors, which are analog in nature, that need to be utilized by digital computers. One approach is to put an analog-to-digital converter (ADC) as close to the analog sensor signals as possible, to take advantage of the computational flexibility available in digital numeric processors. An alternate approach is to perform some of the computations using analog signal processing, requiring simpler ADC, and reducing the computational load on the digital processors that follow. This new approach is called cooperative analog-digital signal processing (CADSP) and promotes a new vision for the way signal processing systems are designed and implemented, promising substantially reduced power consumption in signal processing systems (Hasler and Anderson 2002).

10.8 Low-Power Comparisons to Digital Approaches: Analog Computing in Memory

Our starting point is a programmable analog transistor that allows for simultaneous nonvolatile storage (i.e., an analog weight), computation (i.e., compute a product between this stored weight and inputs), and analog programming that does not affect the computation, can adapt due to correlations of input signals, and is fabricated in standard CMOS processes; several circuits have been discussed in detail elsewhere (Hasler and Lande 2001). Accurate and efficient programming of these programmable transistors enables the analog signal processing approaches. For example, in a 0.5 μm CM OS process, programming a target current within 0.2% is achievable over a current range from 150 pA to 1.5 μA (Bandyopadhyay et al. 2005). Hundreds to thousands of elements can be programmed to this accuracy in a second. The resulting programmed currents show less than 0.1% change over 10 years at room temperature (300 K). Many circuit topologies work well over a wide temperature range.

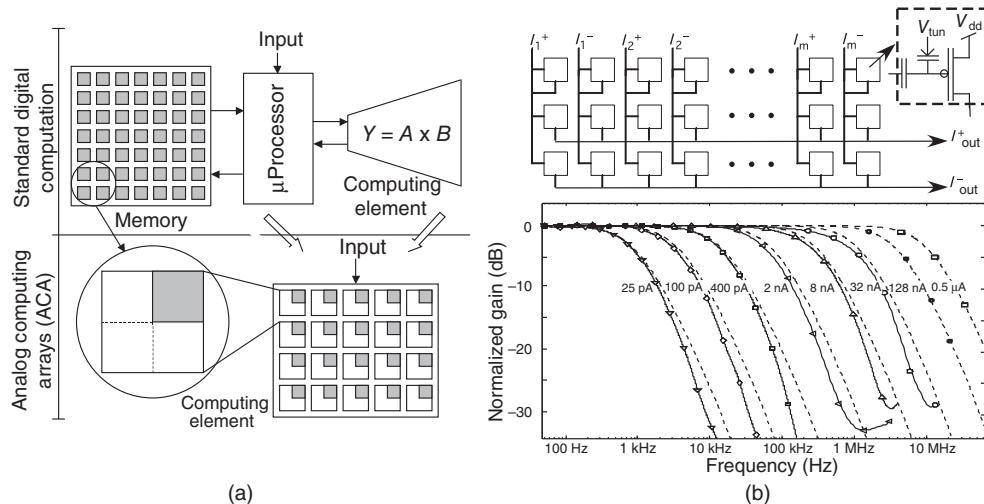


Figure 10.6 Computational efficiency for large-scale analog signal processing. (a) Comparison of the computing in memory approach for analog computing arrays with standard digital computation requiring a memory and processing units. In the complexity and power of accessing two columns of digital data for processing, we perform a full matrix computation on an incoming vector of data. (b) Illustration of the computational efficiency through a vector-matrix multiplication (VMM) computation. Using vector–vector multiplication (one programmable transistor per box), we compare experimental and simulation results throughout the region of subthreshold bias currents. From this experimental data, this analog VMM has a 4 millions of multiply-accumulate operations (MMAC)/μW computational efficiency (primarily due to the node capacitance), which compares favorably to the best DSP IC values between 4 to 10 MMAC/mW. Figure (b) © 2004 IEEE. Reprinted, with permission, from Chawla et al. (2004)

Figure 10.6 shows a comparison between programmable analog computation with traditional digital computation. The concept of analog computing arrays were introduced as parallel analog signal processing computations performed through programmable analog transistors, which appear similar to a few modified EEPROM cells, and therefore corresponds closely to EEPROM densities. Figure 10.6a shows a general block-diagram of the analog computing array, with comparison to traditional digital computation. Unlike digital memory, each cell acts as a multiplier that multiplies the analog input signal to that cell by the stored analog value. Performing the computation in the memory cells themselves avoids the throughput bottlenecks found in most signal processing systems. This computational approach can be extended to many other signal processing operations and algorithms in a straightforward manner. Therefore, a full parallel computation is possible with the same circuit complexity and power dissipation as the digital memory needed to store this array of digital coefficients at 4-bit accuracy. Further, in the complexity required to simply read out two columns of data to be sent to the digital processor, an entire matrix–vector multiplication as well as potentially more complex computations are performed. The trade-off is the possibly reduced precision of the computation and the requirement for floating- gate programming.

Figure 10.6b shows experimental data from a 128×32 vector matrix multiplier (VMM), built using these analog computing array approaches in $0.5 \mu\text{m}$ CMOS in 0.83 mm^2 area (Chawla

et al. 2004). The resulting analog computational analog efficiency, defined as the ratio of bandwidth and power dissipation (i.e., MHz/ μ W), was measured at 4 MMAC/ μ W; the power efficiency is tied to the node capacitance, and therefore values greater than 20 MMAC/ μ W can be achieved in this process (Chawla et al. 2004; Hasler and Dugger 2005). For comparison, the most power-efficient DSP processors (i.e., TI 54C series or DSP factory series) have power efficiencies of 1–8 MMAC/mW (best cases); therefore, the analog approach is 1000 times more computationally efficient than the resulting digital approach. Other analog signal processing techniques have further shown computational power efficiency improvements of between 300–1000-fold over digital approaches (Graham et al. 2007; Hasler et al. 2005; Ozalevli et al. 2008a; Peng et al. 2007). Further such approaches resulted in significant power efficiencies in commercial end products (Dugger et al. 2012). Recent results show greater improvement in power efficiency based on neural-inspired approaches enabling wordspotting computation through a dendritic-rich neuron network (George and Hasler 2011; Ramakrishnan et al. 2012).

10.9 Analog Programming at Digital Complexity: Large-Scale Field Programmable Analog Arrays

The last decades have seen the development of large-scale FPAA technology (Hall et al. 2002, 2004). FPAs have historically had few programmable elements and limited interconnect capabilities, primarily targeted as a replacement for a few op-amp ICs on a board (Anadigm 2003a, 2003b; Fas 1999; Lee and Gulak 1991; Quan et al. 1998). Similar to the evolution in configurable logic from programmable logic devices (PLD) to FPGAs, FPAA chips (Figure 10.7) are moving toward a large number of reconfigurable analog blocks, providing a reconfigurable platform that can be used to implement a number of different applications. The benefits of rapid prototyping for analog circuits would be significant in the design and testing of analog systems.

FPAs, like FPGAs, allow for rapid prototyping of hardware solutions. A typical fabric is composed of several computational analog blocks (CABs); currently the largest FPAs utilize 100 CABs with over 100,000 programmable analog parameters, where thousands of CABs are possible using state-of-the-art CMOS technologies. In the FPAA block, the *switching* device is an analog programmable element that can act as an ideal switch, variable resistor, current source, and/or configurable computational element in a single analog programmable memory element. The resulting chip can be programmed using a high-level digital control interface, typical of FPGA devices, and offers input/output (I/O) pins that connect to the array of CAB elements. Using these analog signal processing techniques for a system design, signal compression is not only essential for efficient transmission from the sensor, but also for efficient digital transmission between the ICs for low-power operation.

Several generations of FPAA devices have been innovated through the last few years including a family of large-scale FPAA devices of CABs and floating-gate routing useful for computation (Basu et al. 2010a, 2010b; Schlottmann et al. 2012a) as well as further architecture revisions enabling partial rapid reconfigurability (Schlottmann et al. 2012b) and integration with low-power configurable digital blocks (Wunderlich et al. 2013). These approaches also developed a USB-powered hardware platform (Koziol et al. 2010), simple enough to be used in class development (Hasler et al. 2011), as well as a wide range of software tools enabling high-level Simulink system design that compiles directly to working hardware (Schlottmann

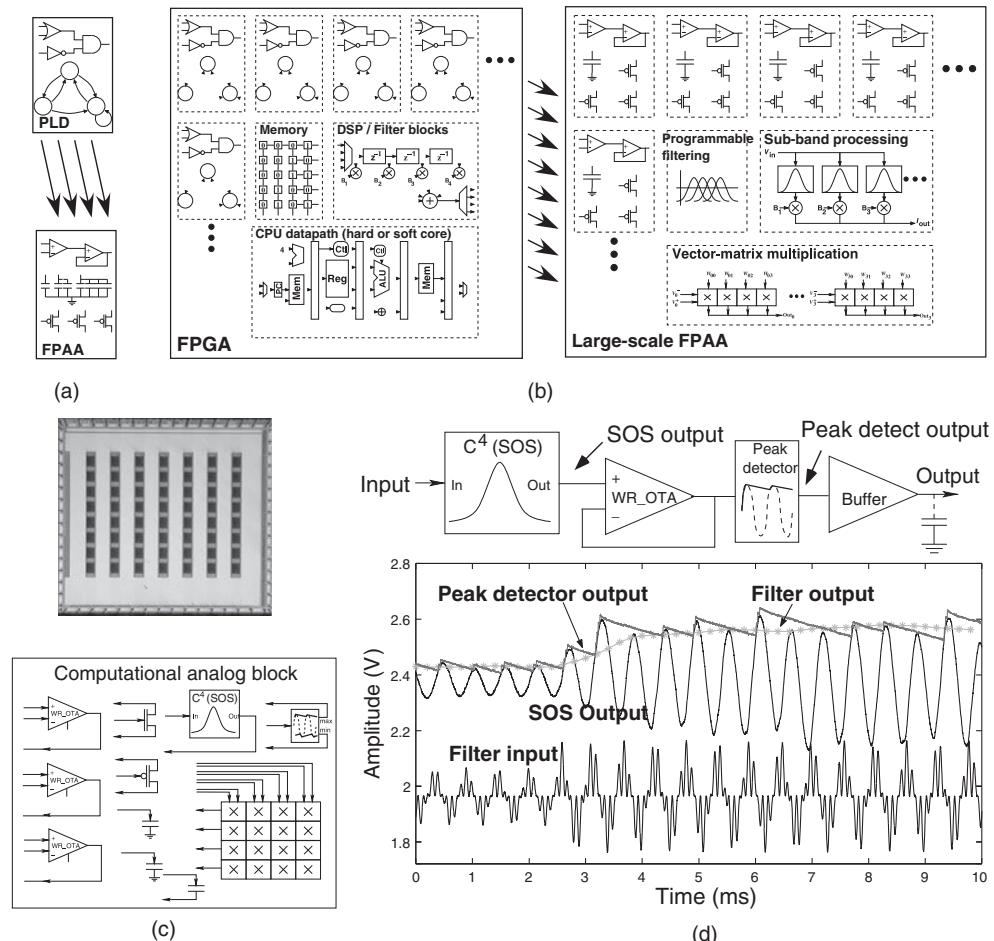


Figure 10.7 Illustration of large-scale field programmable analog arrays (FPAA). (a) Most previous FPAA ICs are closer to programmable logic devices (PLDs). (b) Large-scale FPAs have a computational complexity of analog arrays similar to field-programmable gate array (FPGAs), including multiple levels of routing. (c) Top figure shows a die photo of one large-scale FPAA composed of 64 computational analog blocks (CABs) and 150,000 analog programmable elements. The components in one example CAB are shown in the bottom figure of (c). It contains a 4×4 matrix multiplier, three wide-linear range OTAs (WR OTA), three fixed-value capacitors, a capacitively coupled current conveyor C^4 second-order section (SOS) filter, a peak detector, and two pFET transistors. (d) A typical subband system. The top figures show how the input signal is first bandpass filtered through a C^4 SOS filter before the SOS output is buffered by the WR OTA and then the magnitude of the subband is output from the peak detector. This sequence is analogous to taking a discrete Fourier transform of the signal. The experimental data are taken from an FPAA system. The input waveform is an amplitude modulated signal with 1.8 KHz and 10.0 KHz components. The output of the peak detector is shown with (dotted line connecting capacitor to output in top figure of (d)) and without an integrating capacitor added to the output stage. Figure (d) © 2005 IEEE. Reprinted, with permission, from Hall et al. (2005)

and Hasler 2012). FPAs have been built for a range of signal processing applications (Ramakrishnan and Hasler 2013; Schlottmann and Hasler 2011; Schlottmann et al. 2012b), as well as neural interfacing applications (Zbrzeski et al. 2010).

10.10 Applications of Complex Analog Signal Processing

This section looks at examples of analog signal processing applications where the IC system utilizes a few milliwatts of power. In particular, we will discuss analog transform imagers, CMOS imagers allowing for programmable analog signal processing on the image plane, and continuous-time analog adaptive filters and classifiers. Other interesting applications have been in enhancing speech in noise (Anderson et al. 2002; Yoo et al. 2002), analog front-end systems for speech recognition (Smith et al. 2002; Smith and Hasler 2002), and arbitrary waveform generators and modulators (Chawla et al. 2005). Applications have been proposed in beam-forming, adaptive equalization, radar and ultrasound imaging, software radio, and image recognition.

10.10.1 Analog Transform Imagers

Figure 10.8 shows one example of utilizing programmable analog techniques for an improved combination of CMOS imaging and resulting programmable signal processing with a relatively high fill factor (Hasler et al. 2003; Lee et al. 2005). The transform imager chip is capable of computing a range of matrix transforms on an incoming image. This approach allows for retina and higher-level bio-inspired computation in a programmable architecture that still possesses similar high fill-factor pixels to those of active pixel sensor (APS) imagers. This imager is capable of programmable matrix operations on the image. Each pixel is composed of a photodiode sensor element and a multiplier. The resulting data-flow architecture directly allows computation of spatial transforms, motion computations, and stereo computations. Figures 10.8b and 10.8c show a comparison of using this imager as a single chip solution with using a standard digital implementation for JPEG compression, critical for picture and video enabled handsets, resulting in orders of magnitude improvement in power dissipation, and therefore capable of enabling video streaming on handheld devices.

10.10.2 Adaptive Filters and Classifiers

By continuously programming analog transistors, due to continuous input signals, the transistor strength adapts due to input signal correlations (Hasler and Lande 2001). Adaptive devices enable the development of adaptive signal processing algorithms, like adaptive filters and neural networks, in a dense, low-power, analog signal processing architecture. Weight adaptation is implemented by continuously operating the programming mechanisms, such that the low-frequency circuit adapts to a steady state dependent upon a correlation of input signals and error signals. Hasler and Dugger (2005) reported on a least mean squares (LMS) node based upon an LMS synapse built from continuously adapting programmable transistors; Figure 10.9 summarizes their results. Most other outer-product learning algorithms (supervised or unsupervised) are straightforward extensions of an LMS algorithm. This approach enables

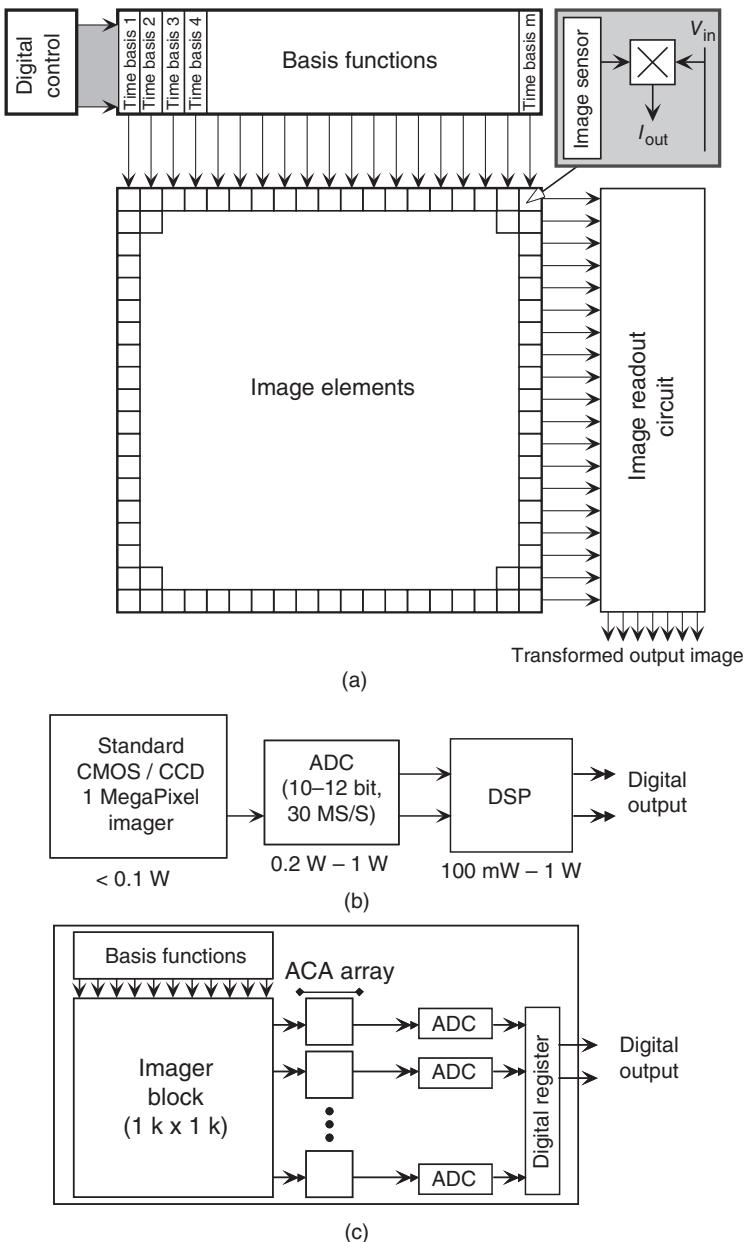


Figure 10.8 Analog technology applied to imaging signal processing applications. (a) Floorplan of the matrix transform imager. Each pixel processor multiplies the incoming input with the measured image sensor result, and outputs a current proportional to this result. The image output rate will be the same as the time to scan a given image. This approach allows arbitrary separable matrix image transforms that are programmable. © 2002 IEEE. Reprinted, with permission, from Hasler et al. (2002a). (b) In a standard joint photographic experts group (JPEG) implementation, the image acquired by a standard complementary metal oxide semiconductor/charge-coupled device (CMOS/CCD) imager is converted to digital values using an analog-digital converter (ADC) before the block discrete cosine transform (DCT) is computed on each image using a digital signal processor (DSP). Each of these blocks burn anywhere from less than 0.1 W to 1 W. (*Continued overleaf*)

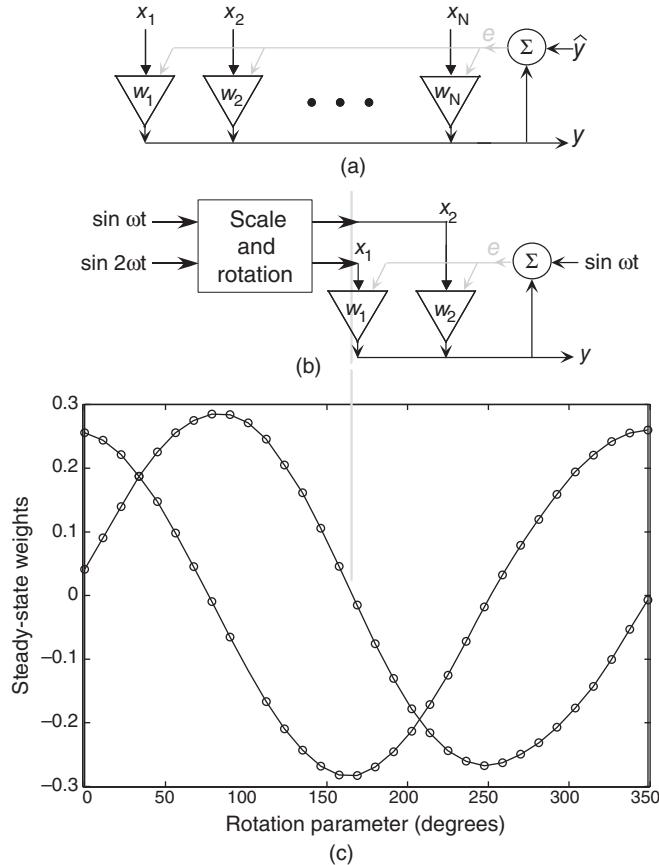


Figure 10.9 An adaptive floating-gate node. (a) Block diagram of a single adaptive floating-gate node. Either supervised algorithms or unsupervised one-layer networks can be built in this architecture. (b) Experimental setup for examining least mean squares (LMS) behavior in a two-input node. A scaling operation followed by application of a rotation matrix to an orthogonal signal-space basis of harmonically related sinusoids yields the system input signals; the fundamental sinusoid is chosen as the target. (c) Measured data show steady-state weight dependence on the parameter, θ , of the two-dimensional input mixing-matrix. As expected theoretically, a cosine curve is obtained for the first weight, and a sine curve for the second weight. Figures (a) and (c) © 2005 IEEE. Reprinted, with permission, from Hasler and Dugger (2005)



Figure 10.8 (Continued) (c) A comparison of a standard image processing system with the transform imager for JPEG computations. The JPEG compression is one representative example for the transform imager, where the matrices are programmed for block DCT transforms. The analog computing array (ACA) ADCs are used to read off the resulting transformed image, but these ADCs could be controlled with variable bit rate based on the information on each channel to result in significant power savings. The transform imager would require roughly 1 mW as a single-chip solution

large scale, on-chip learning networks by enabling a range of signal processing algorithms. Figure 10.9b shows the testing setup to characterize the weight adaptation behavior of a two-input LMS network over a range of input signal correlations; the experimental data (Figure 10.9c, 0.5 μm process) closely agrees with the ideal analytic expressions, proving the performance of this system. Using this adaptive floating-gate LMS node, arrays are being built on the order of 128×128 synapses in a $2 \text{ mm} \times 2 \text{ mm}$ area (0.35 μm process) that operate at under 1 mW of power at bandwidths over 1MHz; a custom digital processor or bank of DSP processors performing similar computations on signals of comparable bandwidth would consume 3–10 W. These techniques can be extended to other programmable and adaptive classifiers, like Vector quantization or Gaussian Mixtures (Hasler et al. 2002a; Peng et al. 2005).

10.11 Discussion

Floating-gate technology provides a compact device fabricated in standard CMOS processes, that simultaneously provides long-term (nonvolatile) storage, computation, automated system programmability in a common architecture independent of functionality, and adaptation in a single device. Such devices allow for digital routing as well as programmable computational elements, sometimes in the same structure. This approach has been used both in custom ICs and in reconfigurable architectures to build complex signal processing systems including programmable and adaptive filters, multipliers, amplification, matrix and array signal operations, and Fourier processing. The alternatives to floating-gate technology for dense storage in analog computing systems are a DAC per parameter or complex dynamic storage; when the number of parameters becomes significant (e.g., more than 30), these approaches have huge system level impacts (e.g., Schlottmann and Hasler 2012).

Floating-gate transistors are the foundation of most commercial nonvolatile memories, including EEPROM and flash memories. Current EEPROM devices already store 4 bits (i.e., 16 levels) in a single transistor occupying $100 \text{ nm} \times 100 \text{ nm}$ in area in a 32 nm process (Li et al. 2009; Marotta et al. 2010), with smaller technology nodes reaching production as of this writing. As a result, floating-gate circuit approaches can be expected to also scale to smaller process nodes, at similar array densities to EEPROM devices; current research efforts are actively working in such modern IC processes.

Floating-gate devices and circuits are used by multiple groups for multineuron neuromorphic implementations (Brink et al. 2013; Schemmel et al. 2004), including parameter storage, capacitive-based circuits, and adaptive learning networks; the use of these approaches results in the densest synaptic array ICs currently available (Brink et al. 2013).

One critical aspect of floating-gate circuits is the accurate and rapid programming of large numbers of floating-gate devices (Basu and Hasler 2011). The approach presented in this chapter enables, by construction, methods that minimize the effect of capacitive mismatch for programming accurate systems. A limiting factor in programming floating-gate devices is accurate current measurement over a wide range (i.e., pA to μA), which has been improved dramatically through the use of fully on-chip circuitry with all-digital interfaces for IC programming (Basu and Hasler 2011). Such infrastructure development has also served to facilitate the adoption of this technology by application specialists in the neuromorphic and signal processing communities.

Turning away from nonvolatile storage, Chapter 11 describes the digital configuration of analog currents and voltages, as a way of including a small number of configurable but volatile chip parameters using standard CMOS technology.

References

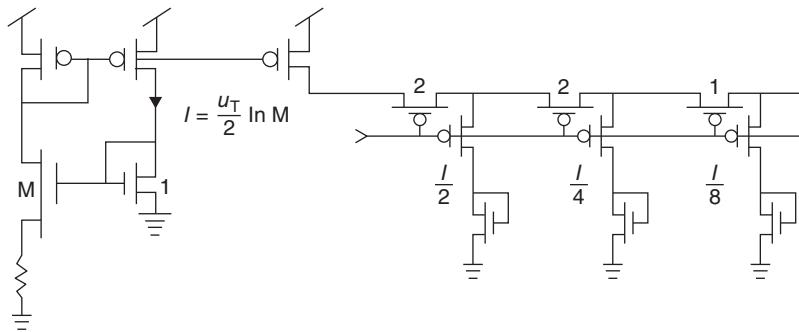
- Anadigm. 2003a. Anadigm Company Fact Sheet Anadigm. http://www.anadigm.com/Prs_15.asp/.
- Anadigm. 2003b. Anadigm FPAA Family Overview Anadigm. http://www.anadigm.com/Supp_05.asp/.
- Anderson D, Hasler P, Ellis R, Yoo H, Graham D, and Hans M. 2002. A low-power system for audio noise suppression: a cooperative analog-digital signal processing approach. *Proc. 2002 IEEE 10th Digital Signal Processing Workshop, and the 2nd Signal Processing Education Workshop*, pp. 327–332.
- Bandyopadhyay A, Serrano G, and Hasler P. 2005. Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **3**, pp. 2148–2151.
- Bandyopadhyay A, Serrano G, and Hasler P. 2006. Adaptive algorithm using hot-electron injection for programming analog computational memory elements within 0.2 percent of accuracy over 3.5 decades. *IEEE J. Solid-State Circuits* **41**(9), 2107–2114.
- Basu A and Hasler PE. 2011. A fully integrated architecture for fast and accurate programming of floating gates over six decades of current. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **19**(6), 953–959.
- Basu A, Brink S, Schlottmann C, Ramakrishnan S, Petre C, Koziol S, Baskaya F, Twigg CM and Hasler P. 2010a. A floating-gate based field programmable analog array. *IEEE J. Solid-State Circuits* **45**(9), 1781–1794.
- Basu A, Ramakrishnan S, Petre C, Koziol S, Brink S, and Hasler PE. 2010b. Neural dynamics in reconfigurable silicon. *IEEE Trans. Biomed. Circuits Syst.* **4**(5), 311–319.
- Brink S, Nease S, Hasler P, Ramakrishnan S, Wunderlich R, Basu A, and Degnan B. 2013. A learning-enabled neuron array IC based upon transistor channel models of biological phenomena. *IEEE Trans. Biomed. Circuits Syst.* **7**(1), 71–81.
- Chawla R, Bandyopadhyay A, Srinivasan V, and Hasler P. 2004. A 531 nW/MHz, 128×32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity. *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, pp. 651–654.
- Chawla R, Twigg CM, and Hasler P. 2005. An analog modulator/demodulator using a programmable arbitrary waveform generator. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 6106–6109.
- Duffy C and Hasler P. 2003. Modeling hot-electron injection in pFETs. *J. Comput. Electronics* **2**, 317–322.
- Dugger J, Smith PD, Kucic M, and Hasler P. 2012. An analog adaptive beamforming circuit for audio noise reduction. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, pp. 5293–5296.
- Fast Analog Solutions Ltd. 1999. TRAC020LH Datasheet: Totally Re-configurable Analog Circuit - TRAC® issue 2, Oldham, UK.
- Frantz G. 2000. Digital signal processor trends. *IEEE Micro* **20**(6), 52–59.
- George S and Hasler P. 2011. HMM classifier using biophysically based CMOS dendrites for wordspotting. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 281–284.
- Graham DW, Smith PD, Ellis R, Chawla R, and Hasler PE. 2004. A programmable bandpass array using floating-gate elements. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **1**, pp. 97–100.
- Graham DW, Hasler P, Chawla R, and Smith PD. 2007. A low-power, programmable bandpass filter section for higher-order filter applications. *IEEE Trans. Circuits Syst. I: Regular Papers* **54**(6), 1165–1176.
- Hall T, Hasler P, and Anderson DV. 2002. Field-programmable analog arrays: a floating-gate approach. In: *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream* (eds. Glesner M, Zipf P, and Renovell M). Lectures Notes in Computer Science, vol. 2438. Springer Berlin, Heidelberg, pp. 424–433.
- Hall T, Twigg C, Hasler P, and Anderson DV. 2004. Application performance of elements in a floating-gate FPAA. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **2**, pp. 589–592.
- Hall T, Twigg C, Gray JD, Hasler P, and Anderson DV. 2005. Large-scale field-programmable analog arrays for analog signal processing. *IEEE Trans. Circuits and Syst. I* **52**(11), 2298–2307.
- Hasler P. 2005. Floating-gate devices, circuits, and systems. *Proc. Fifth Int. Workshop on System-on-Chip for Real-Time Applications*, pp. 482–487.

- Hasler P and Anderson DV. 2002. Cooperative analog-digital signal processing. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)* **IV**, pp. 3972–3975.
- Hasler P and Dugger J. 2005. An analog floating-gate node for supervised learning. *IEEE Trans. Circuits Syst. I: Regular Papers* **52**(5), 835–845.
- Hasler P and Lande TS. 2001. Overview of floating-gate devices, circuits, and systems. *IEEE Trans. Circuits Syst. II* **48**(1), 1–3.
- Hasler P, Diorio C, Minch BA, and Mead CA. 1995. Single transistor learning synapses. In: *Advances in Neural Information Processing Systems 7 (NIPS)* (eds. Tesauro G, Touretzky D, and Leen T). MIT Press, Cambridge, MA. pp. 817–824.
- Hasler P, Minch BA, Diorio C, and Mead CA. 1996. An autozeroing amplifier using pFET hot-electron injection. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **III**, pp. 325–328.
- Hasler P, Minch BA, and Diorio C. 1999. Adaptive circuits using pFET floating-gate devices. *Proc. 20th Anniversary Conf. Adv. Res. VLSI*, pp. 215–229, Atlanta, GA.
- Hasler P, Smith P, Duffy C, Gordon C, Dugger J, and Anderson DV. 2002a. A floating-gate vector-quantizer. *Proc. 45th Int. Midwest Symp. Circuits Syst.* **1**, Tulsa, OK. pp. 196–199.
- Hasler P, Bandyopadhyay A, and Smith P. 2000b. A matrix transform imager allowing high fill factor. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **III**, pp. 337–340.
- Hasler P, Bandyopadhyay A, and Anderson DV. 2003. High-fill factor imagers for neuromorphic processing enabled by floating-gate circuits. *EURASIP J. Adv. Signal Process.* **2003**, 676–689.
- Hasler P, Smith PD, Graham D, Ellis R, and Anderson DV. 2005. Analog floating-gate, on-chip auditory sensing system interfaces. *IEEE Sensors J.* **5**(5), 1027–1034.
- Hasler P, Scholttmann C, and Koziol S. 2011. FPAAs chips and tools as the center of an design-based analog systems education. *Proc. IEEE Int. Conf. Microelectronic Syst. Education*, pp. 47–51.
- Koziol S, Scholttmann C, Basu A, Brink S, Petre C, Degnan B, Ramakrishnan S, Hasler P, and Balavoine A. 2010. Hardware and software infrastructure for a family of floating-gate based FPAAAs. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2794–2797.
- Kucic M. 2004. *Analog Computing Arrays*. PhD thesis Georgia Institute of Technology Atlanta, GA.
- Kucic M, Hasler P, Dugger J, and Anderson DV. 2001. Programmable and adaptive analog filters using arrays of floating-gate circuits. In: *Proceedings of 2001 Conference on Advanced Research in VLSI* (eds. Brunvand E and Myers C). IEEE. pp. 148–162.
- Lee J, Bandyopadhyay A, Baskaya IF, Robucci R, and Hasler P. 2005. Image processing system using a programmable transform imager. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)* **5**, pp. 101–104.
- Lee KFE and Gulak PG. 1991. A CMOS field-programmable analog array. *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, pp. 186–188.
- Li Y, Lee S, Fong Y, Pan F, Kuo TC, Park J, Samaddar T, Nguyen HT, Mui ML, Htoo K, Kamei T, Higashitani M, Yero E, Kwon G, Kliza P, Wan J, Kaneko T, Maejima H, Shiga H, Hamada M, Fujita N, Kanebako K, Tam E, Koh A, Lu I, Kuo CCH, Pham T, Huynh J, Nguyen Q, Chibvongodze H, Watanabe M, Oowada K, Shah G, Woo B, Gao R, Chan J, Lan J, Hong P, Peng L, Das D, Ghosh D, Kalluru V, Kulkarni S, Cernea RA, Huynh S, Pantelakis D, Wang CM, and Quader K. 2009. A 16 Gb 3-bit per cell (X3) NAND flash memory on 56 nm technology with 8 MB/s write rate. *IEEE J. Solid-State Circuits* **44**(1), 195–207.
- Marotta GG, Macerola A, D'Alessandro A, Torsi A, Cerafogli C, Lattaro C, Musilli C, Rivers D, Sirizotti E, Paolini F, Imondi G, Naso G, Santin G, Botticchio L, De Santis L, Pilolli L, Gallese ML, Incarnati M, Tiburzi M, Conenna P, Perugini S, Moschiano V, Di Francesco W, Goldman M, Haid C, Di Cicco D, Orlando D, Rori F, Rossini M, Vali T, Ghodsi R, and Roohparvar F. 2010. A 3 bit/cell 32 Gb NAND flash memory at 34 nm with 6 MB/s program throughput and with dynamic 2 b/cell blocks configuration mode for a program throughput increase up to 13 MB/s. *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, pp. 444–445.
- Marr B, Degnan B, Hasler P, and Anderson D. 2012. Scaling energy per operation via an asynchronous pipeline. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **21**(1), 147–151.
- Mead CA. 1990. Neuromorphic electronic systems. *Proc. IEEE* **78**(10), 1629–36.
- Minch BA, Hasler P and Diorio C. 2001. Multiple-input translinear element networks. *IEEE Trans. Circuits Syst. II* **48**(1), 20–28.
- Ozalevli E, Huang W, Hasler PE, and Anderson DV. 2008a. A reconfigurable mixed-signal VLSI implementation of distributed arithmetic used for finite impulse response filtering. *IEEE Trans. Circuits Syst. I: Regular Papers* **55**(2), 510–521.

- Ozalevli E, Lo HJ, and Hasler PE. 2008b. Binary-weighted digital-to-analog converter design using floating- gate voltage references. *IEEE Trans. Circuits Syst. I: Regular Papers* **55**(4), 990–998.
- Peng SY, Minch B, and Hasler P. 2005. Programmable floating-gate bump circuit with variable width. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **5**, pp. 4341–4344.
- Peng SY, Hasler P, and Anderson DV. 2007. An analog programmable multi-dimensional radial basis function based classifier. *IEEE Trans. Circuits Syst. I: Regular Papers* **54**(10), 2148–2158.
- Quan X, Embabi SHK, and Sanchez-Sinencio E. 1998. A current-mode based field programmable analog array architecture for signal processing applications. *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, Santa Clara, CA, pp. 277–280.
- Ramakrishnan S and Hasler J. 2013. A compact programmable analog classifier using a VMM + WTA network. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, pp. 2538–2542.
- Ramakrishnan S, Wunderlich R, and Hasler PE. 2012. Neuron array with plastic synapses and programmable dendrites. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 400–403.
- Sarpeshkar R 1997. *Efficient Precise Computation with Noisy Components: Extrapolating From an Electronic Cochlea to the Brain*. PhD thesis. California Institute of Technology, Pasadena, CA.
- Schemmel J, Meier K, and Mueller E. 2004. A new VLSI model of neural microcircuits including spike time dependent plasticity. *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, **3**, pp. 1711–1716.
- Schlottmann CR and Hasler P. 2012. FPAA empowering cooperative analog-digital signal processing. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, pp. 5301–5304.
- Schlottmann CR and Hasler P. 2011. A highly dense, low power, programmable analog vector-matrix multiplier: the FPAA implementation. *IEEE J. Emerg. Select. Top. Circuits Syst.* **1**(3), 403–41.
- Schlottmann CR, Abramson D, and Hasler P. 2012a. A MITE-based translinear FPAA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2**(1), 1–9.
- Schlottmann CR, Shapero S, Nease S, and Hasler P. 2012b. A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. *IEEE J. Solid-State Circuits* **47**(9), 2174–2184.
- Serrano G, Smith P, Lo HJ, Chawla R, Hall T, Twigg C, and Hasler P. 2004. Automatic rapid programming of large arrays of floating-gate elements. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **I**, pp. 373–376.
- Shibata T and Ohmi T. 1992. A functional MOS transistor featuring gate-level weighted sum and threshold operations. *IEEE Trans. Electr. Devices* **39**(6), 1444–1455.
- Smith PD and Hasler P. 2002. Analog speech recognition project *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)* **4**, pp. 3988–3991.
- Smith PD, Kucic M, Ellis R, Hasler P, and Anderson DV. 2002. Cepstrum frequency encoding in analog floating-gate circuitry. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **IV**, pp. 671–674.
- Srinivasan V, Dugger J, and Hasler P. 2005. An adaptive analog synapse circuit that implements the least-mean-square learning algorithm. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **5**, 4441–4444.
- Srinivasan V, Serrano GJ, Gray J, and Hasler P. 2007. A precision CMOS amplifier using floating-gate transistors for offset cancellation. *IEEE J. Solid-State Circuits* **42**(2), 280–291.
- Srinivasan V, Serrano GJ, Twigg CM, and Hasler P. 2008. A floating-gate-based programmable CMOS reference. *IEEE Trans. Circuits Syst. I: Regular Papers* **55**(11), 3448–3456.
- Wunderlich RB, Adil F, and Hasler P. 2013. Floating gate-based field programmable mixed-signal array. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **21**(8), 1496–1504.
- Yoo H, Anderson DV, and Hasler P. 2002. Continuous-time audio noise suppression and real-time implementation *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)* **IV**, pp. 3980–3983.
- Zbrzeski A, Hasler P, Kölbl F, Syed E, Lewis N, and Renaud S. 2010. A programmable bioamplifier on FPAA for in vivo neural recording. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 114–117.

11

Bias Generator Circuits



Neuromorphic chips often require a wide range of biasing currents which are independent of process and supply voltage, and which change with temperature appropriately to result in constant transconductance. These currents can span many decades, down to less than the transistor ‘off-current’. This chapter describes how to design wide-dynamic range configurable bias current references. The output of each current reference is a gate voltage which produces a desired current. Bias currents are generated by a bootstrapped-mirror ‘master bias’ current reference that generates a master current, which is successively divided by a digitally-controlled current splitter to generate the desired reference currents. Nonidealities such as power supply sensitivity, matching, stability, and headroom are also discussed. Open source design kits simplify the job of including these circuits on new designs.

11.1 Introduction

Analog and mixed-signal neuromorphic and bio-inspired chips such as the sensors described in Chapters 3 and 4 and some of the multichip systems described in Chapter 13 require a number of adjustable voltage and current references. Reference voltages are used, for example, as

inputs to differential amplifiers used in feedback transimpedance configuration, as thresholds for voltage comparators, and as inputs to tilted resistor ladders. Reference voltages typically do not need to be precise and on many designs it does not matter much if they vary somewhat with process, although they should not be sensitive to power supply noise. Current references are used everywhere, for example, for biasing amplifiers, for setting time constants of various circuits, and for powering loads for static logic.

Chips will often have large numbers of identical circuits (e.g., pixels, column amplifiers, or cells) that require identical biases. The required currents can extend over many decades. For instance, consider a chip with circuits that span timescales from nanoseconds to milliseconds which uses subthreshold $g_m - C$ filters with 1pF capacitors. The rise time T of simple $g_m - C$ circuits scales as C/g_m . The transconductance g_m of a transistor in subthreshold operation scales as I/U_T , where I is the bias current and U_T is the thermal voltage. Such a chip would require bias currents $I = CU_T/T$ from 10 μ A to 10 pA – a range of six decades. Actually the current range would be larger than this because the fast circuits would run super-threshold and require even higher currents. This huge range of bias current creates special requirements that are quite different than conventional industrial mixed-signal chips, which typically derive all bias currents from a few references spanning only a couple of decades in current.

Bias current references are often left out in experimental chips because designers assume that these ‘standard’ circuits can easily be added in later revisions when the chip’s design can be productized. But this is a poor strategy to achieve process, voltage and temperature (PVT) manufacturing tolerance. As a result, these chips need to be tuned individually for correct operation. These biases are often specified by directly setting bias transistor gate voltages using off-chip components. However, the required voltages depend on chip-to-chip variation in threshold voltage. If these voltages are generated by potentiometers or supply-referenced digital-to-analog converters (DACs), they are sensitive to power supply ripple. Also, the subthreshold currents depend in an exponential way on temperature, doubling every 6–8°C around room temperature. Potentiometers and voltage DACs consume significant static power. Each chip requires individual adjustment, which can be difficult and time consuming, especially if the parameter space has many dimensions. These adjusted parameters must be carried along with the chip throughout its assembly into a final system and the distribution of this system to end-users.

Some early designers generated scaled versions of the desired currents using off-chip resistors that supply current to on-chip scaling current mirrors (Gray et al. 2001). Although this scheme is straightforward, it requires a pin for each independent bias, it requires a regulated power supply, and it necessitates bulky on-chip current mirrors when a small on-chip current is required. For example, a feasible off-chip resistance of 1 M Ω requires an impractical current mirror with a ratio of 1 million.

Industry-standard current references are poorly suited to the biasing of most neuromorphic chips. The reason is that the references used in industry stem from a long tradition of analog circuits that run above threshold, and that only require a limited span of biasing currents. When these references are used on neuromorphic chips, they provide a small range of potential bias currents, which has the consequence that sometimes refabrication of the chip is required to adjust the current to the required value.

Several groups have developed circuits that replace these off-chip components with digitally configurable on-chip circuits. Some of the designs are open-sourced with layout and schematics (jAER 2007), and have been incorporated into chips from several groups, much as the shared

AER transceiver circuits from the Boahen group at Stanford. These ICs include vision and auditory sensors and multineuron chips.

This chapter starts with an extensive discussion of the core circuits, each followed by typical measurements of operating characteristics. The chapter concludes with a brief description of available design kits.

11.2 Bias Generator Circuits

The block diagram in Figure 11.1 shows three main components: the ‘master bias’ that generates the master current I_m , a ‘current splitter’ that subdivides the master current into scaled copies to form a set of smaller references, and a configurable buffer that copies the bias current to the places it is used. Each bias uses a copy of the master current and a current splitter. A selected fraction of the master current is copied from the splitter to form the individual bias current. This current is copied by a highly-configurable active mirror, resulting in a voltage that is connected to gates of transistors in the biased circuits. These circuits are described in the following sections, starting with the master bias.

11.2.1 Bootstrapped Current Mirror Master Bias Current Reference

The master current I_m in Figure 11.1 is generated by the familiar bootstrapped current reference in Figure 11.2 which is attributed to Widlar (1965, 1969) and is first reported in CMOS by Vittoz and Fellrath (1977) (see also textbooks such as Baker et al. 1998; Gray et al. 2001; Lee 2004; Razavi 2001). Transistors M_{n1} and M_{n2} have a gain ratio $(W_{n1}/L_{n1})/(W_{n2}/L_{n2}) = M$. Since the currents in the two branches are forced to be the same by the mirror $M_{p1}-M_{p2}$, the ratio in current density in the M_n transistors sets up a difference in their gate-source voltage, which is expressed across the load resistance R . The key feature of this circuit is that the current is determined only by the resistance R and ratio M and the temperature, such that the resulting transconductance is independent of temperature, as we will now discuss.

The master current I_m that flows in the loop is computed by equating the currents in the two branches. In subthreshold, this equality is expressed by

$$I_m = I_0 e^{\kappa V_n / U_T} = I_0 M e^{(\kappa V_n - I_m R) / U_T}, \quad (11.1)$$

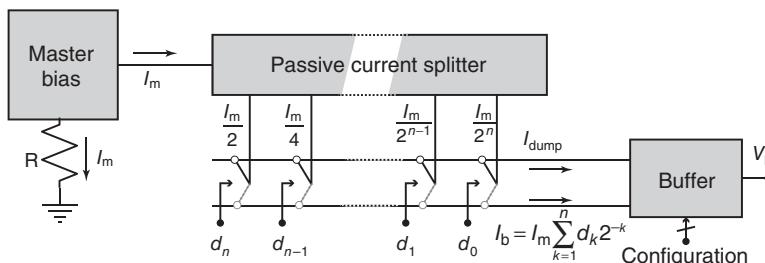


Figure 11.1 Bias generator architecture showing core circuits

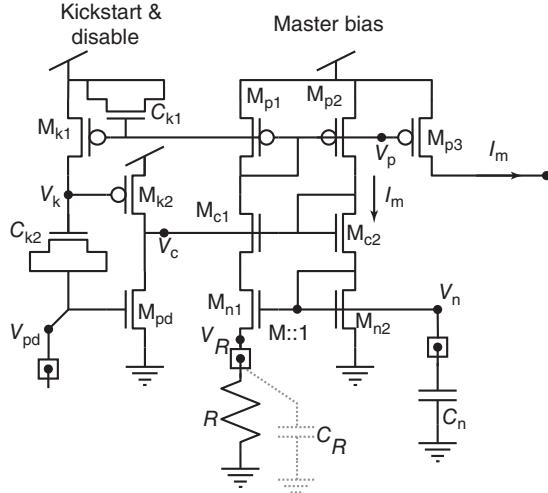


Figure 11.2 The master bias current reference, together with startup and disable circuit. I_m is the master current, and R is an external resistance that sets I_m . C_{k1} and C_{k2} are MOS capacitors of a few hundred fF. The squares represent bonding pads and recommended external connections for prototyping. Adapted from Delbrück and van Schaik (2005). With kind permission from Springer Science and Business Media

where I_0 is the transistor off-current and κ is the back-gate or body-effect coefficient (also known as $1/n$). Eliminating I_o and V_n from Eq. (11.1) results in the remarkably simple yet accurate formula Eq. (11.2):

$$I_m = \ln(M) \frac{U_T}{R}, \quad (11.2)$$

where $U_T = kT/q$ is the thermal voltage. The voltage V_R across the load resistor R does not depend on the resistance R in subthreshold and provides a direct measurement of temperature:

$$V_R = \ln(M) U_T. \quad (11.3)$$

An above-threshold analysis yields Eq. (11.4) that is not very accurate but still useful for estimating the required resistance:

$$I_m = \frac{2}{\beta_n R^2} \left(1 - \frac{1}{\sqrt{M}} \right)^2, \quad \beta = \mu_n C_{ox} \frac{W_{n2}}{L_{n2}}. \quad (11.4)$$

Here μ_n is the electron-effective mobility, and C_{ox} is the unit gate oxide capacitance. In strong inversion the current decreases with R^2 , while in weak inversion it decreases as R .

The actual I_m is approximately the sum of Eqs. (11.2) and (11.4). With ideal transistors I_m does not depend on supply voltage or threshold voltage, but is closely proportional to absolute temperature (PTAT) in subthreshold. In reality it is slightly affected by the supply voltage through drain conductance and also by mismatch of the threshold voltage between the

transistors in the current mirrors. This master bias circuit is often called the constant g_m circuit because the g_m of a transistor biased with current I_m is independent of temperature for both weak and strong inversion operation of the master bias. The transconductance of a transistor with W/L the same as M_{n2} biased with current I_m is given by

$$g_m = \frac{\kappa \ln(M)}{R} \text{ (in weak inversion)} \quad g_m = \frac{2(1 - 1/\sqrt{M})}{R} \text{ (in strong inversion).} \quad (11.5)$$

These g_m depend only on R , M , and κ (Nicolson and Phang 2004). Thus, g_m depends on temperature in either weak or strong inversion only on the temperature-dependence of R and κ . This dependence causes g_m to vary much less than would the exponential dependence of a voltage-biased transistor. This temperature independence holds only if the transistor is of the same type as M_{n1} and running in the same operating regime. Circuits that are biased from the smaller currents from the splitter outputs will have a slightly temperature-dependent g_m .

The ratio M is not critical as long as it is substantially larger than 1. Commonly used values range from 8 to 64. The main effect of M is to change the loop gain, which affects power supply rejection, startup, and stability.

11.2.2 Master Bias Power Supply Rejection Ratio (PSRR)

It is important that this master current not be affected by inevitable supply voltage variation or noise. To increase the power supply rejection ratio (PSRR) of the master bias current, the drain resistances of the transistors are increased by using long M_p 's and cascoding M_{n1} with M_{c1} and M_{c2} . The pFETs are not cascaded to preserve headroom. Razavi computes the power supply sensitivity of the master bias current as an exercise (Example 11.1 in Razavi (2001)). The result of this small-signal analysis is interesting and a bit surprising in that the sensitivity to power supply voltage vanishes if the M_{p2} mirror output transistor in Figure 11.2 has infinite drain resistance. In other words, if the p-mirror copies the current perfectly, the output resistance of the M_{n1} (or M_{c1}) transistor is irrelevant. Why is this plausible? If the p-mirror copies perfectly, it is impossible for the n-mirror to have unequal output current. Therefore, the original assumption that the current is equal in the two branches is satisfied and power supply variation has no effect. One might still think that finite drain conductance in M_{n1} increases the gain of the M_{n1} – M_{n2} mirror, but this is not the case, because increased drain conductance does not increase incremental mirror gain; it only increases output current, and incremental current gain is what determines the master bias current. Simulations of the master bias circuit that increase the length of the M_{n1} – M_{n2} mirror (which decreases the mirror's output conductance) slightly increase the master bias current.

For low voltage operation, where the cascode occupies too much headroom, making transistors M_{p1} and M_{p2} long and excluding the M_{c1} – M_{c2} cascode are good choices.

11.2.3 Stability of the Master Bias

There is a subtle potential instability in the master bias but it is easily avoided with proper understanding. Excessive C_R can cause large-signal limit-cycle oscillations. Since this node is often connected to an external resistor, it is easy to unintentionally create a large capacitance. A large M ratio can also create a large parasitic capacitance C_R . The circuit can be stabilized

by making the compensation capacitor C_n several times C_R . In practice, V_n is brought to a bonding pad, where an external capacitance ensures that the master bias can be stabilized. In the most recent design kits (jAER 2007), a large capacitor is included in the layout and it is not necessary to bring V_n to a pad. For more details of this possible instability, see Delbrück and van Schaik (2005).

11.2.4 Master Bias Startup and Power Control

A startup circuit is necessary for the master bias. Very small current (sometimes incorrectly called ‘zero current’) in both branches of the bootstrapped current mirror circuit can form a metastable operating point (Lee 2004), which can even be stable depending on parasitic currents; see Delbrück and van Schaik (2005) for an extensive discussion.

Whatever the cause of the metastable low current state, the startup simulation in Figure 11.3 shows that escape from an ‘off’ state can be slow, even when it is unstable, so a startup circuit is necessary to escape this parasitic operating point quickly when power is applied.

A number of startup mechanisms are currently in use (Ivanov and Filanovsky 2004; Razavi 2001; Rincon-Mora 2001). Here we describe a four-transistor startup circuit that transiently injects current into the current mirror loop on power-up and then shuts itself off. Unlike many other startup circuits, this mechanism is process-independent because it does not depend on threshold or supply voltage and does not require any special devices. It is used on a commercial computer peripheral product that has shipped hundreds of millions of units over more than ten years of commercial production. However, it is still a good idea to simulate startup behavior over the most pessimistic conditions, e.g., slow ramp-up of supply voltage, low threshold voltage and high temperature.

To understand the startup circuit, refer back to Figure 11.2. Transistors M_{k1} , M_{k2} , and M_{pd} , and MOS capacitors C_{k1} and C_{k2} enable the startup and power-control functionality. The loop is kick-started on power-up by the current flowing from M_{k2} , which is ‘on’ until V_k is charged

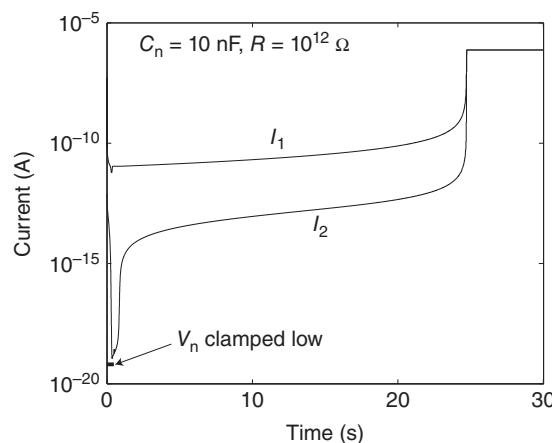


Figure 11.3 Simulation of slow restart when off state is intentionally produced. Adapted from Delbrück and van Schaik (2005). With kind permission from Springer Science and Business Media

to V_{dd} by M_{k1} , which then shuts off. C_{k2} holds V_k low on power-up (V_{pd} is at ground), while C_{k1} ensures that V_p is initially held near V_{dd} , holding M_{k1} ‘off’ so that the kick-start can occur. C_{k1} and C_{k2} must be large enough so that sufficient charge flows into the loop to get it going; a few hundred femtofarads is typical. C_{k1} and C_{k2} are MOS capacitors to avoid the necessity of a special capacitor layer, such as a second polysilicon layer. The polarity of the MOS capacitors is arranged so that they operate in inversion (C_{k2}) or accumulation (C_{k1}) while kick-start is active. (C_{k1} has another important role that is discussed later.) While the bias generator is operating, no current flows in this startup circuit. The charge injected by M_{k2} is a complex function of circuit parameters, but the essential point is that M_{k2} is not shut off until the master bias has current flowing in it.

If the master bias circuit ever falls into a metastable low-current state, there is no rapid automatic recovery. It is possible that such a circumstance could arise under, for instance, deep brown-out conditions with slow recovery of the supply voltage. Capacitor C_{k1} is important here because it tends to hold the gate-source voltage of M_{p1} – and hence its current – constant when V_{dd} changes. If C_{k1} is not included, a sudden drop in V_{dd} can transiently turn off M_{p1} , possibly leading to an unintended and extended shutdown of the master bias.

In some systems the ability to completely shut off all bias currents and then restart them is desirable, such as for a sensor chip that needs only periodic activation by an external periodic wake-up signal. This control enabled by the ‘power-down’ input V_{pd} , which is grounded for normal operation. Raising V_{pd} to V_{dd} turns off the master bias and the derived biases by pulling V_c to ground through M_{pd} and shutting off the current in the loop. Yanking V_{pd} back to ground yanks V_k low, through C_{k2} (M_{k1} is ‘off’), and the start-up circuit restarts the current as before. While V_{pd} is high, no current flows in M_{pd} , because V_k is at V_{dd} and M_{k2} is off. A conductive path to ground from V_n (say, through a leaky C_n) could require pumping V_{pd} for a few cycles at a sufficient rate to move the current mirror loop to a regime of positive feedback. But if, as usual, there is no DC path other than through M_{n2} , a single downward transition on V_{pd} is sufficient for restart. Other circuits such as the bias buffer in Section 11.2.10 are also adapted to the use of this disable signal by tying the bias to the appropriate rail in the disabled state.

11.2.5 Current Splitters: Obtaining a Digitally Controlled Fraction of the Master Current

The master bias described so far generates a reference current I_m . This master current is a single reference current generated in a single circuit block in a specific location of a chip. In order to use this current as a reference bias for the rest of the analog circuits in the chip, it needs to be first scaled to the specific bias currents needed in the chip, and second, distributed throughout the chip. In this section we concentrate on the first aspect: scaling it to needed values. More specifically, we will describe ways to down-scale to desired values in a digitally controlled manner. This is a good option specially for prototype circuits where the designer may want to explore alternative bias values than those used during the initial simulations and design, or where the biases need adjustment during operation. (Why not scale up the master bias current? Practical values of resistance dictate that the master current lies near the upper end of bias currents. Of course it is possible to scale up the master current by a small multiple by using a scaled current mirror, or more compactly, using a variation on the current splitter sometimes called a compound mirror, but we will not discuss these here. For details on the

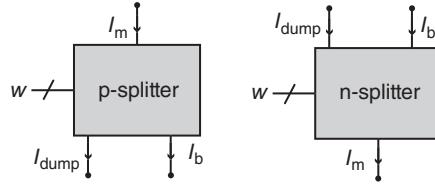


Figure 11.4 Splitter concept. The input master current I_m (either sourced on the left or sunk on the right) is converted into an output current I_b according to the digital word provided at control bus w . The remaining current is dumped to I_{dump} .

implementation of a compound-mirror splitter that multiplies the master current by factors of 8 and 64 see Yang et al. 2012, Figure 1.)

In general, the current splitter circuit can be represented by the block shown in Figure 11.4. In this block an input current I_m is converted into an output current I_b according to the digital word provided at control bus w . The fraction of input current I_m not driven to output I_b is dumped onto line I_{dump} . Depending on the circuit technique used for down-scaling the input reference current, this dumping output may or may not exist. In the rest of this section we describe several possible ways to provide a digitally-controlled down-scaled value of the reference current.

Current splitting is accomplished most robustly and compactly by using the principle of current division (Bult and Geelen 1992). We will compare passive and active current splitters and introduce the principle of current division in our discussion of passive splitters next.

Passive Current Splitting Ladder

In the passive splitter in Figure 11.5, the master current is copied to the current splitter by transistor M_{p3} . The splitter successively divides the copy of I_m in a way similar to an R-2R

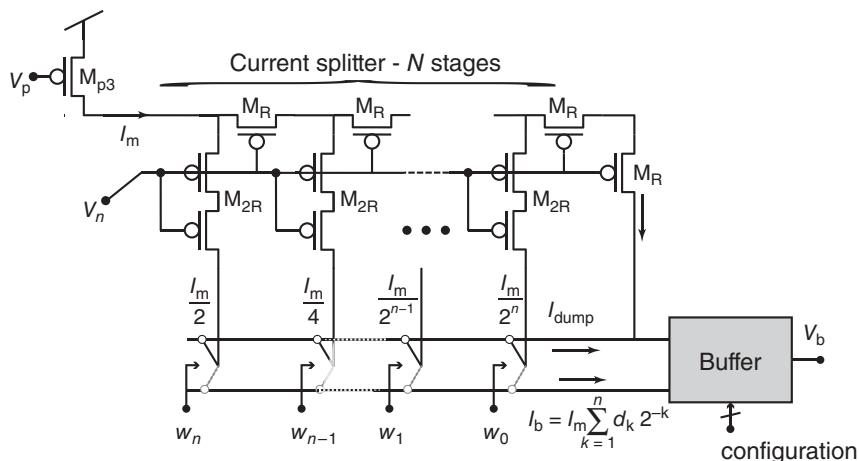


Figure 11.5 Passive current splitter. M_R and M_{2R} are identically sized unit transistors

resistor ladder, to form a geometrically spaced series of smaller currents. By configuring the switches using bits of w , the desired current branches of the splitter are connected to form I_b , while the rest of I_m goes to I_{dump} .

This splitter is called passive because we input a current and the same charge that we input is output at various taps. At each branch, half of the current is split off, and the rest continues to later stages. The last stage is sized to terminate the line as though it were infinitely long. By starting at the far end of the splitter, it is easy to see how the parallel and series combinations of the transistors result in an equal split of the currents at each branch. M_R and the two M_{2R} transistors (which each have the same W/L as the M_R transistor) form the R-2R network; the octave splitter is terminated with a single M_R transistor. The splitter has N stages, and the current flowing transversely out from the splitter at stage k is $I_m/2^k$. The last two currents are the same. The reference voltage V_n for the pFET gates in the splitter should be a low voltage to minimize splitter supply-voltage requirements, but it needs to be high enough to saturate the output circuit that sums up the selected currents. V_n here is the same master bias voltage V_n from Figure 11.2, because it conveniently scales correctly with master bias current.

The current division principle accurately splits currents over all operating ranges, from weak to strong inversion, dependent only on the effective device geometry. In this R-2R splitter, behavioral independence from the operating regime is most easily understood by following each transistor's operation back from the termination stage and observing that the transverse M_{2R} and lateral M_R transistors share the same source and gate voltage and that the transverse transistor is in saturation. It can be easily observed that combining series and parallel paths causes half the current to flow into each branch at each stage without any assumption about channel operating conditions. It is also easy to see that, looking from the input terminal, the entire splitter forms a 'compound transistor' that has an effective W/L equal to one of its M_R or M_{2R} unit transistors.

Active Current Splitting Ladder

The current splitting principle illustrated in Figure 11.5 passively divides an input current into branches. It is also possible to exploit the MOS ladder structure to actively generate weighted current sources. This is illustrated in Figure 11.6, where transistor M_0 is used to set the gate

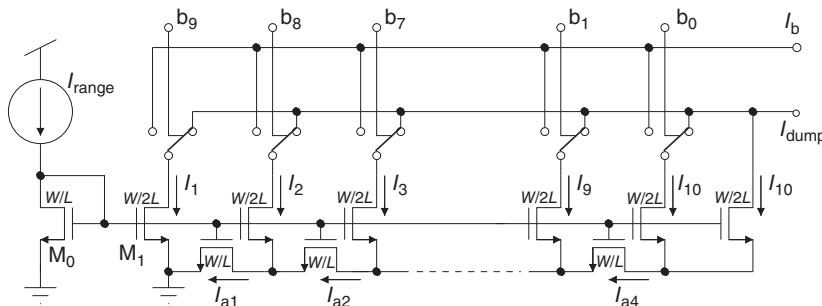


Figure 11.6 Active current splitter that exploits the current splitting principle for the generation of active current sources © 2003 IEEE. Reprinted, with permission, from Linares-Barranco et al. (2003)

voltage of the ladder structure. One can think of M_0 as being the input transistor of a current mirror (of size W/L), while the series-parallel compound of the other transistors behaves as an equivalent M_1 transistor also of size W/L . This M_1 transistor would be the output transistor of the current mirror. Since both M_0 and M_1 have identical size (W/L), they will drive the same current I_{range} . However, the total current through equivalent output transistor M_1 is physically branched into current components I_j which add up to I_{range} , where at each stage $I_j = 2I_{j+1}$. As in the passive current splitter case, one has to guarantee that the currents flow through each branch. Consequently, the drain terminals of all vertical transistors should be connected to voltage nodes with the same drain voltage V_D . In Figure 11.6 these drain terminals are connected to either node I_b or node I_{dump} , but both should be biased at the same voltage V_D to avoid differential drain conductance effects. The switches are set according to the digital values b_j . In the same way as for the passive splitter, the output current value available at node I_b can be any combination of unit current sources I_j . Therefore, the current mirror output transistor M_1 effectively operates as a digitally-controlled-size transistor whose size can be digitally set between 0 and W/L in steps of size $(W/L)/(2^n)$ where n is the number of control bits. Drain voltage V_D has to be large enough to guarantee that a W/L size transistor driving a current I_{range} would remain in saturation. Similarly to the passive current splitting case, this active current source also operates correctly under the weak, moderate, and strong inversion regimes. However, unlike for the passive splitter, if the smallest currents are smaller than the off current, then the source voltage must be shifted as described in Section 11.2.8.

Achieving Splitter Ratios other than Factors of Two

Figures 11.5 and 11.6 show R-2R splitters built from unit transistors that split by octaves, but other ratios are also possible by using M_R and M_{2R} with different aspect ratios. By changing the aspect ratios of the vertical and horizontal transistors, one can control the relative ratio between consecutive current branches. For example, Figure 11.7 illustrates how to ratio horizontal, vertical, and terminating transistors to achieve a ratio factor N among consecutive current branches.

However, we strongly recommend the use of unit transistors. A ratio $N = 8$, for instance, would require 7 parallel transverse and a series/parallel combination of 8/7 lateral unit transistors, making the layout large. In this case, it would be much simpler to build an octave splitter

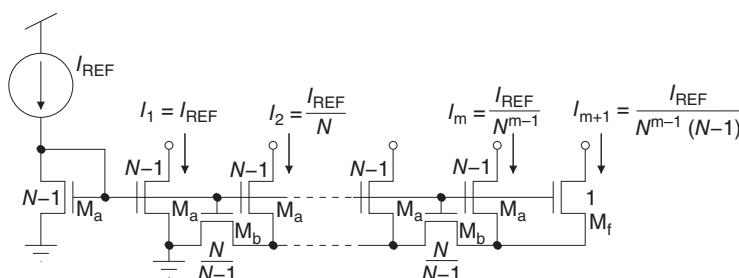


Figure 11.7 Generating arbitrary current ratios between consecutive branches. Adapted from Linares-Barranco et al. (2004). With kind permission from Springer Science and Business Media

and select only every third output. Subtle effects related to substrate bias and short/narrow channel effects can act differentially on transistors with differing aspect ratios. These effects cause nonideal splitter behavior, especially in deep subthreshold. These effects are not always modeled properly, so simulations may not show the correct measured behavior. See Yang et al. (2012) for a specific arrangement of a factor-of-eight splitter and measurements from this structure.

Comparing Active versus Passive Current Splitters

Active splitters have the advantage of lower headroom requirements, but the cost can be greater overall imprecision and smaller current range at the low end. Because the active splitter mirrors from a single input transistor to a number of copies, precision depends largely on the original copy, which depends on the matching of the input transistor. Since this transistor is sized to match the unit transistors used in the splitter, its area is constrained more strongly than the pFET copy of the master bias used in the passive splitter illustrated in Figure 11.5. This pFET can have arbitrarily large area, so its matching can be made quite precise. On the other hand, the passive splitter requires a headroom of more than one diode drop, compared with the single diode-drop of the active splitter. Both types of splitters have been successfully used in numerous designs.

11.2.6 Achieving Fine Monotonic Resolution of Bias Currents

MOS transistors in current splitter circuits are prone to inter-device mismatch. As a result the down-scaled digitally-controlled currents have random errors (from splitter to splitter) around the nominal desired value. For example, when using the $2W/L - W/L$ ratios shown in Figure 11.6 for an 8-bit active splitter with maximum nominal current of 500 pA, one would expect to attain 256 possible current bias values equally spaced between 0 and 500 pA. However, one could obtain the actual measured result shown in Figure 11.8a, where each of the 256 values deviates randomly from the desired nominal value. The maximum possible deviation is obtained for the two central values, because the ladder switches from current $\sum_{i=2}^8 I_i$ to I_1 (see Figure 11.6). In this particular case, the accumulation of random errors in the branches has resulted in a large gap. Note that increasing the number of branches per ladder

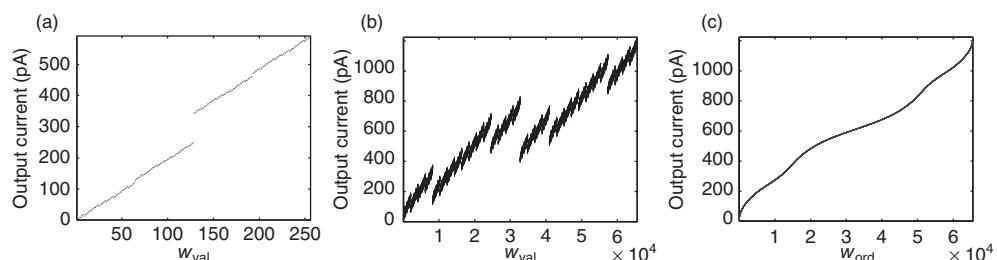


Figure 11.8 Motivation for fine splitters. © 2007 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2007)

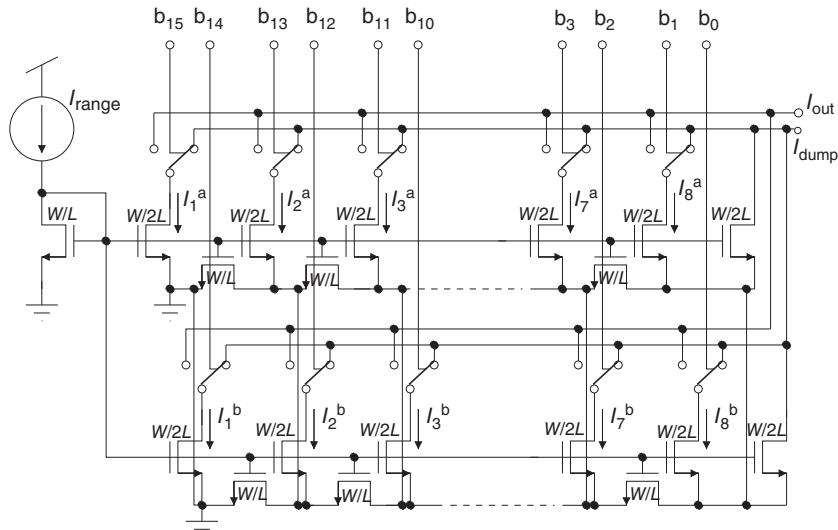


Figure 11.9 Duplicate ‘stochastic ladder’ circuit for fine biasing. © 2007 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2007)

does not solve this problem. This can be a severe problem if the user is trying to program a critical value within the gap. Or in another case, it could be that the situation is reversed, and the resulting currents are nonmonotonic in code value. The result would be that increasing code would actually decrease the current for some values. This could be a problem if the bias current is controlled as part of a feedback loop, since it might result in positive feedback instability.

To avoid this situation, one option is to use the ‘double ladder’ circuit shown in Figure 11.9 (Serrano-Gottaredona et al. 2007), which is also referred to as a ‘stochastic ladder’. This way, a lot of redundancy is introduced because now each ladder branch is replicated, each with its own random error, and any combination of branches is possible because now there are twice as many control bits. Figure 11.8b shows the result of combining two ladders, each like the one used for Figure 11.8a. As can be seen, the output current now goes up to twice the previous maximum and the digital code (x -axis) now reaches now 2^{16} . On the y -axis we can see that the values go up and down and that values overlap and the coverage is much finer. Figure 11.8c shows the same current output values as in Figure 11.8b, but re-ordered monotonically. This re-ordering is done in practice by first measuring all the values in Figure 11.8b (or at least those of the sixteen $I_i^{a,b}$ branches), and second storing a lookup table to map from original digital word \mathbf{w}_{val} to the ordered one \mathbf{w}_{ord} . The drawback of this technique is that each fabricated double ladder in each chip needs to be experimentally characterized and a lookup table of 2^n n -bit entries, where n is 16 in Figure 11.8, needs to be computed and stored, either in firmware or software. This calibration time could increase testing cost and the cost of the lookup memory could be prohibitive for production. Another implementation of range selection (Yang et al. 2012), open-sourced in the jAER project (jAER 2007), overlaps the coarse ranges and includes built-in current measurement circuits, potentially allowing filling in of overly-large steps and calibration during normal operation.

11.2.7 Using Coarse–Fine Range Selection

A generic digitally controlled bias cell should be designed in such a way that it can be used for providing a range of possible bias currents as wide as possible. The options described so far build a single ladder per bias cell with as many ladder branches as needed. For example, if our master bias provides a maximum current of $100 \mu\text{A}$, and we would need smallest bias currents around 1 pA , we would need an M-2M ladder with at least 27 bits. However, in this case, the smallest programmable current steps are in the order of the smallest current, thus making the tuning in the lower range extremely coarse and providing far too much unnecessary resolution at the highest biases. A solution for this is using coarse–fine range selection, which can also reduce the total number of required control bits (Bult and Geelen 1992; Serrano-Gotarredona et al. 2007); this is also the strategy employed in the latest open-source design kit (jAER 2007; Yang et al. 2012).

Figure 11.10 shows a possible realization of this concept. The reference current is first copied by a ladder structure with a ratio scheme of $N = 10$. This way, consecutive ladder branches currents are scaled by a factor of 10. A digital selection circuit chooses only one of the branches and provides current I_{range} . This current is then fed to a ladder with octave ratios. For example, we can have the range selection ladder cover from $100 \mu\text{A}$ down to 1 pA , which would require 8 output branches for the $N = 10$ ladder. To select one out of the 8 branches we only need four range selection bits. Then, for the $N = 2$ ladder we could use 8 bits to have 256 selection values for each range.

The diagram in Figure 11.10 also includes an additional current mirror for output sign inversion, which can be switched into the current path using bit w_{sign} . Also, the output current can be directed to its destination bias or to an external test pad for measuring the selected current, by configuring bit w_{test} . In this particular figure, the $N = 2$ ladder uses the stochastic

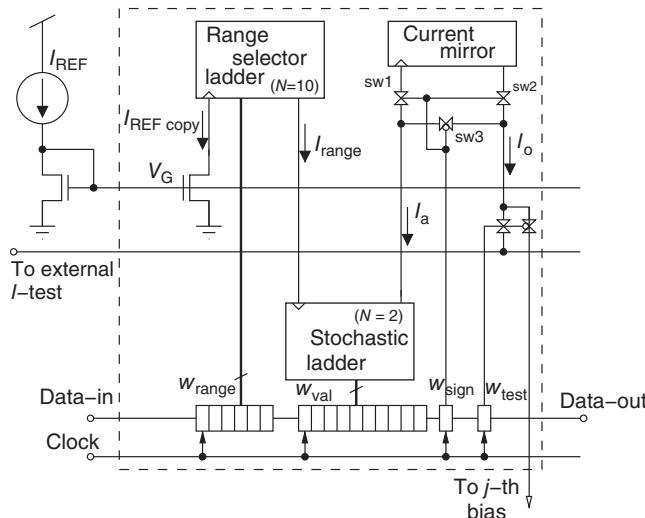


Figure 11.10 Example bias cell including range selection capability and output current sign selection.
© 2007 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2007)

ladder replication for fine current sweeping (Serrano-Gotarredona et al. 2007). The open-source implementation, aimed at ease of implementation and high degree of integration, is described in Yang et al. (2012). This design kit includes the configurable buffer described in Section 11.2.10.

11.2.8 Shifted-Source Biasing for Small Currents

As technology shrinks, so does the power supply voltage and the transistor threshold voltage. All else being equal, subthreshold leakage ‘off current’ I_0 grows exponentially with smaller threshold voltage. (We use the term ‘off-current’ to mean the drain-source current of a transistor that is off. Specifically we distinguish this current from the reverse-biased junction leakage current of sources and drains to the local substrate. The off current is the saturation current of a transistor with $V_{gs} = V_{sb} = 0$, i.e., the pre-exponential I_0 in the saturated subthreshold transistor drain current: $I_{ds} = I_0 \exp(\kappa/U_T)$, where κ is the back-gate coefficient and U_T is the thermal voltage. I_0 is typically several orders of magnitude larger than the substrate leakage current; e.g., 1 pA vs. 1 fA in a 180 nm process technology.) On the 2 μm technologies of the late 1980s, off currents were around 1 fA. In the mainstream 180 nm analog technologies of today, they hover around 10 pA. In the future many logic technologies will have even larger off currents of up to perhaps 1 nA. In analog designs a conventional current mirror can only copy currents down to a few times I_0 .

Linares-Barranco et al. (2003, 2004) presented the principle of shifted source (SS) biasing (Figure 11.11). The shifted source voltage V_{sn} of the current mirror is a few hundred millivolts from the power rail to allow the current mirror to operate with its gate voltage *below* its common source voltage. A level-shifting source follower (biased with current I_{bb}) tied from the current mirror drain input to the common gate voltage V_g holds the drain-source voltage of the input transistor in saturation even for I_1 that are as small as I_0 or even less. The common sources of both M_1 and M_2 are held at V_{sn} , which is typically 200 mV to 400 mV above ground. This arrangement allows V_g to drop below V_{sn} for small (sub-off) input currents. The current mirror is then capable of copying currents several decades smaller than I_0 as illustrated in the right part of Figure 11.11. To build a complete system, complementary pairs of such mirrors are required with supplies V_{sn} and V_{sp} .

The shifted source voltages V_{sn} and V_{sp} are quite close to the power rails and must be actively regulated there, particularly since they will also be used by any circuits biased by these small currents. The V_{sn} regulated voltage can be supplied from the low-dropout regulator circuit

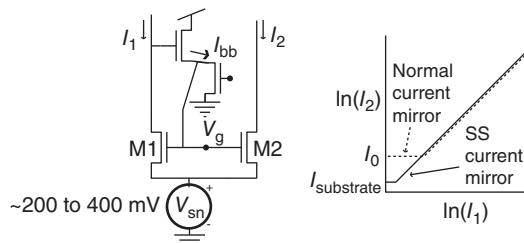


Figure 11.11 Principle of shifted-source (SS) current mirror. © 2010 IEEE. Reprinted, with permission, from Delbrück et al. (2010)

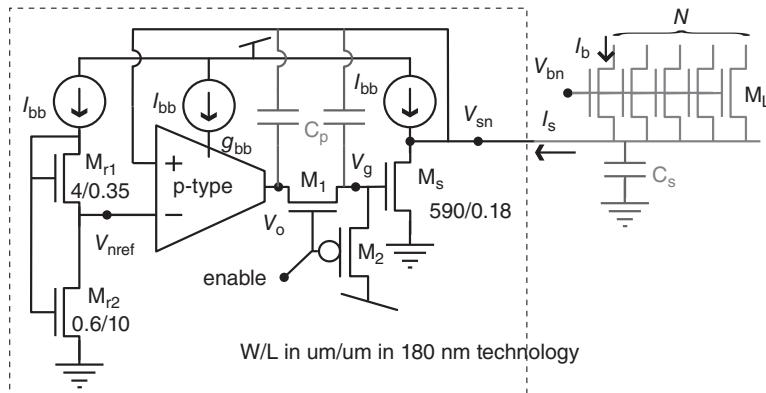


Figure 11.12 Shifted-source reference and regulator circuit. © 2010 IEEE. Reprinted, with permission, from Delbrück et al. (2010)

in Figure 11.12, with the complementary circuit for V_{sp} (Delbrück et al. 2010). The reference voltage V_{nref} is generated by the split-gate diode-connected pair M_{r1} and M_{r2} . M_{r1} is wide and short and M_{r2} is long and narrow. M_{r2} runs in triode mode, acting as a load resistance and thus allowing generation of a reference voltage of 200–400 mV. The buffer amplifier then is used to create a negative feedback loop that regulates the shifted source voltage to the reference voltage. It is important to use a wide-output-range amplifier for the buffer, because the output voltage, which is the gate voltage of the load transistor, may need to swing all the way from low to high voltages, depending on the I_s source current.

The programmable buffer bias current sources I_{bb} , which is typically about 1 μ A, sets V_{nref} and biases the p-type error amplifier. The wide pass transistor M_s sinks the current I_s supplied by the N external nFET sources M_L . The OTA drives V_g (when the regulator is enabled) in negative feedback to regulate V_{sn} to V_{nref} . If V_{sn} is too low then V_g is decreased, and vice versa. The I_{bb} current source onto V_{sn} holds V_{sn} up when I_s sourced externally vanishes. It also sets the minimum transconductance of M_s . Switches M_1 and M_2 allow disabling the SS regulator and tying V_{sn} to ground by disconnecting the OTA and tying V_g to V_{dd} . Parasitic capacitances C_p (especially across the drain-gate capacitance of M_s) can lead to instability since they provide a positive feedback path from V_o to V_{sn} . A large C_s/C_p capacitive divider ratio reduces the feedback gain to stabilize the regulator.

11.2.9 Buffering and Bypass Decoupling of Individual Biases

Once the current splitter provides its output current I_b , the next step is to distribute it to its destination point or points within the chip. The situation changes dramatically depending on whether it is a single destination bias point, or there are a large number of them (for example, a bias within the pixels of a million pixel array). In the general case where the bias is a small current or could be used to bias many cells, it is a good idea to actively buffer the bias voltage to de-couple it from external disturbance and enable special states such as being connected to the power rail. Recent design kits have included a general purpose bias buffer cell that provides these functionalities, as well as allowing post-layout choice of bias voltage polarity. This way,

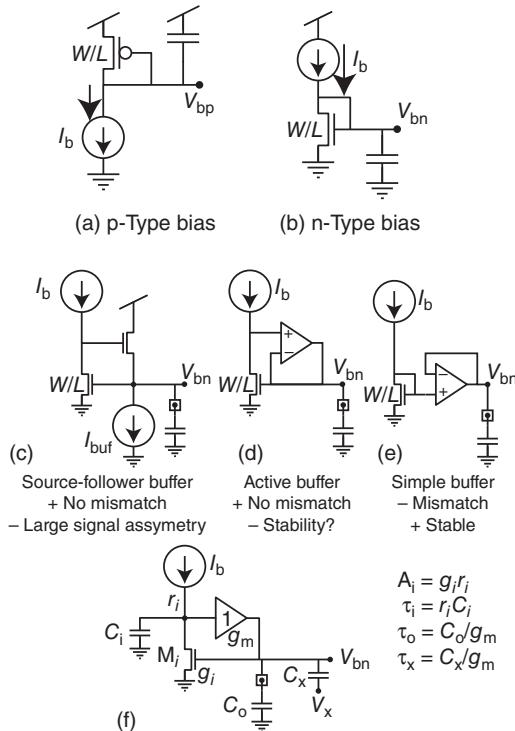


Figure 11.13 Bypassing and buffering. (a) and (b) show how to bypass n- and p-type biases. (c)–(e) compare ways to buffer bias voltages. (f) is the circuit used in the small signal stability analysis. Figures (c) to (f) adapted from Delbrück and van Schaik (2005). With kind permission from Springer Science and Business Media

designers can simply include the biasing layout and choose the voltage type and other options by setting configuration bits. Here we will first discuss some general decoupling and buffering considerations, illustrated in Figure 11.13, and then describe an implementation of a general purpose bias buffer (BB), shown in Figure 11.14.

A diode-connected transistor sinking current I_b and operating in subthreshold has a transconductance or source conductance of $g \approx I_b / U_T$ (neglecting the difference caused by κ). This means that the bias voltage for a small bias current will have a high impedance (e.g., $\approx 10^8 \Omega$ for $I_b = 100$ pA) and can easily be disturbed by other signals on the chip that are capacitively coupled to it, e.g., by crossing wires or by drain-gate parasitic capacitance. Figure 11.13a,b shows the simplest remedy which is to bypass the bias with a large capacitance to the appropriate power rail (V_{dd} for p-type and ground for n-type). Bypassing the bias has the additional benefit of greatly reducing the effect of power-supply ripple on the bias current. It is important to bypass to the appropriate power rail so that the bias voltage is stabilized relative to the appropriate transistor source voltage. The parasitic capacitance to the other rail will then have much less effect on the gate-source voltage.

If the chip will be exposed to light, care must be taken when bringing these generated bias voltages off chip because ESD protection structures in the bonding pads can produce significant

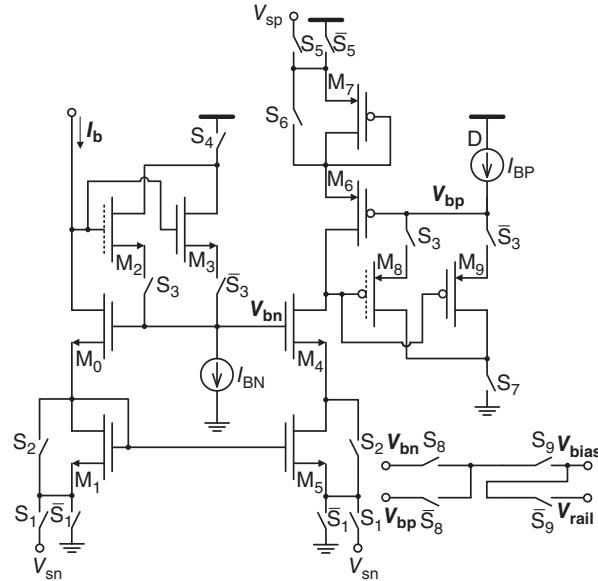


Figure 11.14 Configurable bias buffer circuit. The input to the buffer is the selected splitter output I_b and the output is the voltage V_{bias} . Configuration bits control whether the bias voltage is intended for n- or p-type transistor biasing, if the bias includes an extra diode-connected transistor for cascode biasing, whether the bias is tied weakly to the power rail, and whether a shifted supply voltage is used in connection with shifted-source low current biasing. Transistors with dashed gates are low threshold devices. © 2012 IEEE. Reprinted, with permission, from Yang et al. (2012)

currents under illumination (e.g., several nA under 1 klux). When the programmed bias current is small, these parasitic photocurrents in the bonding pads can significantly perturb the bias currents. In addition, parasitic conductance between package pins can significantly affect the generated biases when bias currents are in the sub-nA range, especially under humid conditions. Therefore it is a good idea to actively buffer this gate voltage so that its source impedance is independent of the bias current. Section 11.2.11 discusses how parasitic photocurrents affect current splitters. Here we discuss active buffering as shown in Figure 11.13c–f.

Active buffering of the gate voltage is simple but there are a few issues to watch out for and the principles should be understood to be used effectively. The total capacitance on the bias voltage is often large when a large number of identical circuits (e.g., pixels) are biased. This immediately helps bypass the bias voltage to reduce capacitive coupling. On the other hand, any disturbance of the voltage can last a long time because the large capacitance must be charged back to the static level. The circuits in Figure 11.13c and d and the bias buffer circuit in Figure 11.14 use an active current mirror with feedback that introduces no first-order mismatch because the gate voltage V_{bn} settles at the voltage required for the input transistor to sink the bias current I_b . This feedback gives them an advantage over the simple voltage buffer of the diode-connected transistor shown in Figure 11.13e, which introduces the offset voltage of the voltage buffer. However, the resonance frequencies of the active circuit must be considered because the amplifier may be driving a large capacitance, so the time constants at the input and output nodes of the amplifier can be comparable. If the resonance frequencies

are comparable to the disturbance frequencies, the buffer can amplify the disturbance rather than suppress it.

The condition that avoids resonance can be calculated (Delbrück and van Schaik 2005) by using the equivalent circuit in Figure 11.13f. The response to a sinusoidal disturbance V_x coupled through capacitance C_x is given by 11.6, where the parameters are shown in the figure:

$$\frac{V_{bn}}{V_x} \approx \frac{\tau_x s(\tau_i s + 1)}{(\tau_i s + 1)(\tau_o s + 1) + A_i}. \quad (11.6)$$

To achieve critical damping, the poles of Eq. (11.6) must lie on the negative real axis. In other words, the solutions to setting the denominator of the right hand side of Eq. (11.6) to zero ($(\tau_i s + 1)(\tau_o s + 1) + A_i = 0$) must be real and negative. To achieve this, Eq. (11.7) must apply to the buffer amplifier time constant:

$$\tau_o < \frac{\tau_i}{4A_i} \quad (11.7)$$

Here τ_o is the time constant of the output of the unity gain buffer amplifier, which drives (usually) a large capacitance. τ_i is the time constant of the amplifier input node, which consists of the small input capacitance C_i and the large input resistance $r_i \approx V_E/I_b$, where V_E is the Early voltage at the input node. A_i is the gain of the input node looking from the gate of M_i . If $\tau_o = \tau_i$, we have the condition of maximum resonance, and $Q = \sqrt{A_i}/2$, which will typically be about 10. When the circuit is properly biased according to Eq. (11.7), however, the equivalent time constant of the high pass filter shrinks to Eq. (11.8):

$$\tau = \frac{\tau_i}{4A_i} = \frac{C_i}{4g_i} = \frac{C_i U_T}{4kI_b} \approx \frac{C_i U_T}{I_b}. \quad (11.8)$$

Compared with the passive case of just diode-connecting the generated bias current, which has a high-pass time constant of $\tau = C_o U_T/I_b$, the actively buffered bias high-pass time constant is reduced by a factor C_o/C_i , which can be many orders of magnitude. To achieve this state, the error amplifier must be biased to be fast enough to fulfill Eq. (11.7). Because these parameters vary with use of the bias and bias current, the buffer current is also made programmable.

11.2.10 A General Purpose Bias Buffer Circuit

The active mirror in Figure 11.13c is used in the general purpose configurable bias buffer (BB) circuit shown in Figure 11.14 (Yang et al. 2012). The input to the BB is the splitter output current I_b and it generates the bias voltage V_{bias} . This complicated circuit has many configuration switches that enable various modes, but the principle of operation is simply based on a complementary pair of active mirrors that can be optionally configured to use shifted sources (Section 11.2.8) and whether an additional diode-connected transistor is inserted into the current path to generate a cascode bias. The input to the buffer circuit is the programmed fraction I_b of I_m , and the output is a buffered bias voltage V_{bias} . Internal buffer currents I_{BN} and I_{BP} are generated by a separate bias generator that is shared across all biases. In this buffer bias generator, I_{BN} and I_{BP} come from diode-connected transistors rather than the BB circuit. V_{rail} is the power supply voltage supplied in ‘disabled’ state.

When implementing this buffer, it is important to carefully simulate it to study headroom limitations. Transistors M_2 and M_8 are low threshold devices. These are automatically used for large bias currents to allow sufficient headroom. Switches S_3 are configured by logic (not shown) connected to the coarse bias bits so that the proper branch is used depending on the level of current.

11.2.11 Protecting Bias Splitter Currents from Parasitic Photocurrents

When building optical sensors, it is important to consider the effects of parasitic photocurrent when using small bias currents such as those using the shifted source scheme discussed in Section 11.2.8. Every active region can act like a photodiode, creating current to the local bulk (either substrate or well). These photocurrents can be comparable to bias currents for the smallest biases, which could affect circuit operation by changing bias current depending on the light shining on the chip.

For use on an optical sensor chip, current splitters built from pFETs are better to use, because these pFETs are built in an n-well implanted in a p-substrate. (We typically assume a p-type substrate because this has become by far the most common scenario.) Thus they can be protected from the effects of parasitic photocurrents by covering the n-well with metal. No minority carriers will then be generated in the n-well, so there will not be any parasitic photocurrent created in the pFETs. If nFET devices were used, their sources and drains would act as parasitic photodiodes which would collect diffusing minority carriers from exposed areas of silicon.

11.3 Overall Bias Generator Architecture Including External Controller

All these subcircuits interface to an external controller in a complete bias generator. The example discussed here, evolved from Delbrück et al. (2010), is shown in Figure 11.15. An external controller (typically a microcontroller, but it could be an integrated controller) loads configuration bits into shift registers that control a set of N independent bias currents, one of which is shown in the inset. (For an integrated controller, these bits could come directly from a register bank rather than from a shift register.) The master current I_m which is generated by a single bootstrapped mirror is shared over all N biases. In this implementation, the 32-bit configuration word for a bias is shifted into shift register stages (SR) and then latched (L). (The latches are necessary since otherwise the biases would change in an unpredictable manner as the bits were loaded.) The 32 bits are partitioned into 22 bits for the bias current I_b , 6 bits for the bias buffer current I_{bb} , and 4 bits of buffer configuration. Biases are daisy-chained (15 in the case of this implementation). Bits are loaded on the IN0 bit while clocking CLOCK; after all bits have been loaded, LATCH is toggled to activate new settings. In the original circuits like this one, designs were simplified by using an architecture where bits were serially loaded into a single global shift register. The most recent implementations use a coarse-fine range-selection strategy with addressable biases (Yang et al. 2012).

Figure 11.16 shows a partial layout of the bias generator layout in a 180 nm 4-metal 2-poly process. Each bias occupies an area of about 80% of a $200 \times 200 \mu\text{m}^2$ pad. To prevent parasitic photodiode action at low currents, the entire circuit is covered with image sensor black

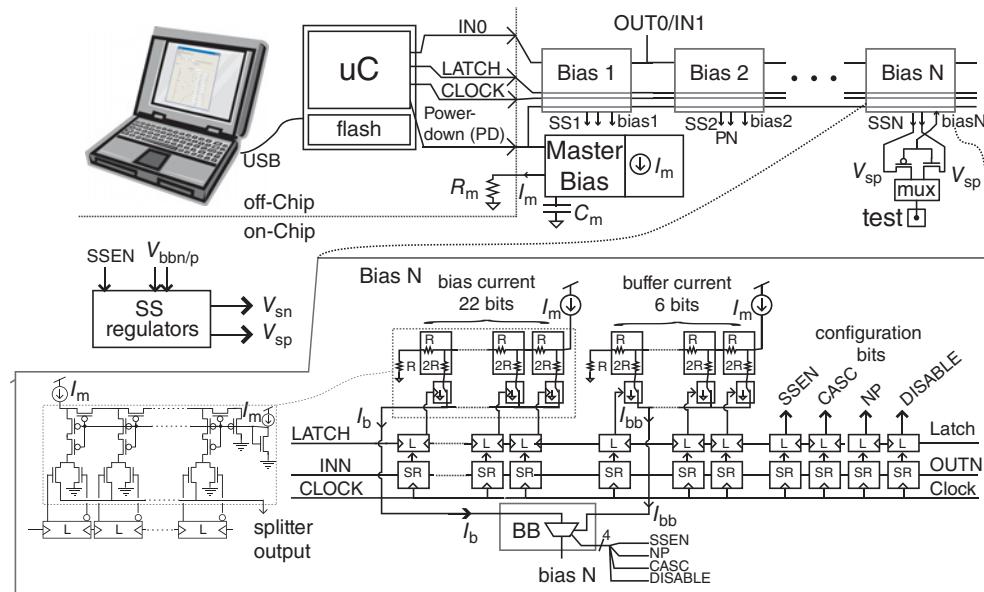


Figure 11.15 Example of overall bias generator architecture. © 2010 IEEE. Reprinted, with permission, from Delbrück et al. (2010)

shield and low current parts of the circuit are covered with metal; n-well guard rings around nFETs provide additional shielding from minority carriers that could diffuse laterally through the substrate.

11.4 Typical Characteristics

Discussion about measured characteristics from one of the fabricated bias generators will conclude this chapter. The intention is to show typical behavior and to bring out some of the observed nonidealities; one example was already shown in Figure 11.8 for current splitter

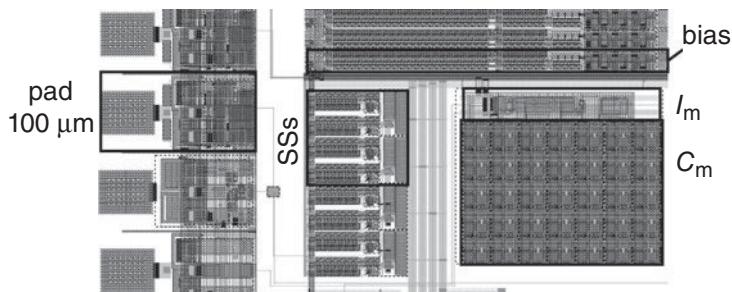


Figure 11.16 Part of the layout of the bias generator in Figure 11.15 in 180 nm technology. A single bias occupies an area comparable to a 100 μm -pitch pad

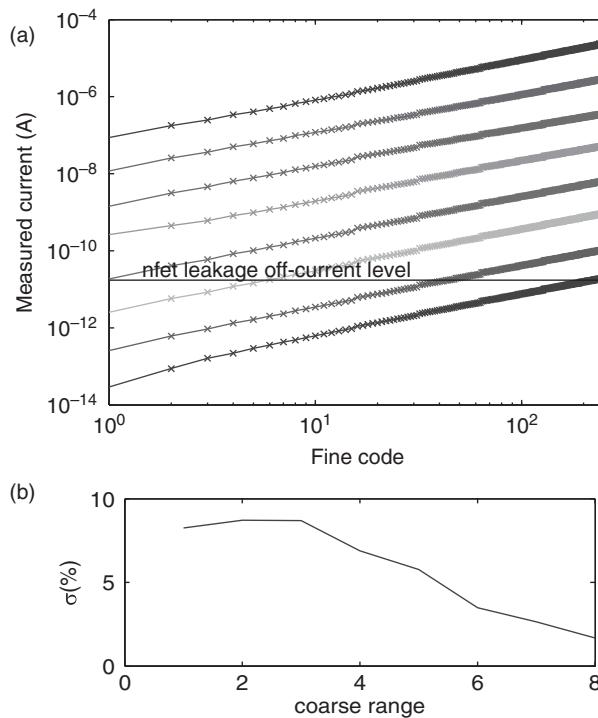


Figure 11.17 (a) Measured bias currents vs. fine bias code value over all coarse ranges. Shifted-source biasing was used for the lowest three coarse ranges. © 2012 IEEE. Reprinted, with permission, from Yang et al. (2012). (b) Measured coefficient of variation ($\sigma I / I$) for the fine code value 127 across 12 biases on one chip

variability. The second example shown in Figure 11.17 is from data taken from the coarse-fine design kit reported in Yang et al. (2012).

Figure 11.17a shows a measured bias current over all 8 coarse ranges. Currents can be generated over a range of more than 180 dB, extending from 25 μ A down to more than a factor of 100 smaller than the off current level. The overlapping ranges allow flexibility in choosing a desired bias current. Figure 11.17b shows matching of a mid-level bias current across 12 biases on one chip. The constant relative variability of about 10% (in the smaller current region) at any node of the current splitter is a consequence of the fact that series and parallel resistors combine to have the same total variability as any single element (Scandurra and Ciofi 2003). Variability is reduced for bias currents above threshold.

11.5 Design Kits

Several generations of bias generator design kits are open-sourced (jAER 2007). Open-sourced design kits allow developers to more easily add bias generators to their designs. In the latest generation kit, which targets a 180 nm process (Yang et al. 2012) and Cadence design tools,

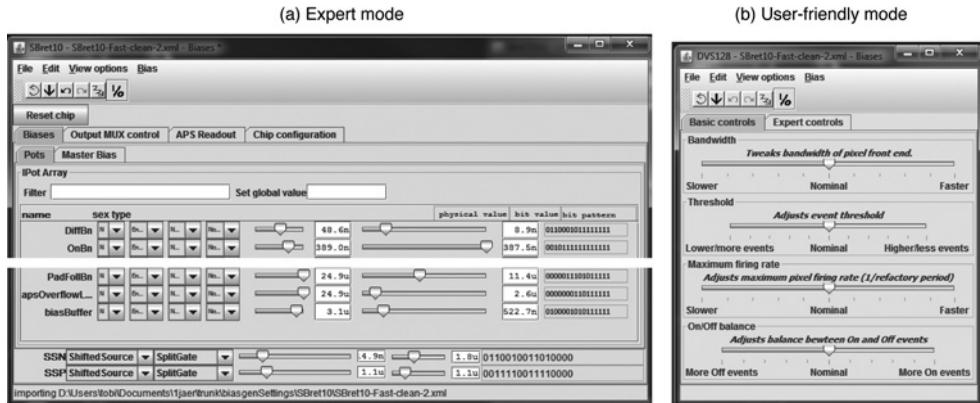


Figure 11.18 GUI control of biases. (a) Expert mode, showing selection of bias type, coarse range and fine code. (b) User friendly mode, where functional control of sensor characteristics is exposed and changes around the nominal value in (a) changes selected biases appropriately

biases are generated by coarse-fine strategy, can be individually addressed, can be measured with a built in calibration circuit, and are digitally configured with a variety of output options. The custom layout that is necessary for each bias is to tie the reset value of each bias to one of the two power rails. These kits include necessary firmware and host PC software necessary to build expert and user-friendly bias controls such as the ones illustrated in Figure 11.18.

11.6 Discussion

This chapter discussed the circuits and architectures for integrated bias current generators. These circuits can be thought of mostly as specialized current DACs. The next few years should bring still more evolution that adapts and improves these circuits and their system-level implementations towards still smaller process technologies with lower supply voltages and increased flexibility and functionalities. It is likely that calibration will become more important as the cost of digital circuits continues to decrease while transistor matching tends to worsen.

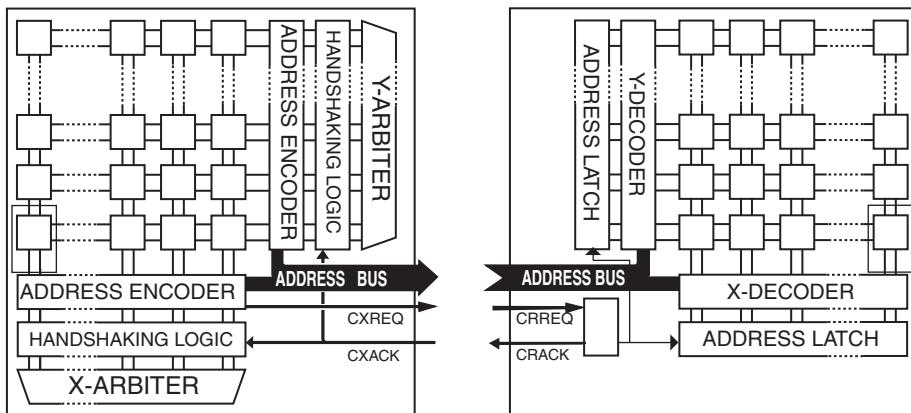
References

- Baker RJ, Li HW, and Boyce DE. 1998. *CMOS Circuit Design, Layout, and Simulation*. IEEE Press.
- Bult G and Geelen G. 1992. An inherently linear and compact MOST-only current division technique. *IEEE J. Solid-State Circuits* **27**(12), 1730–1735.
- Delbrück T and van Schaik A. 2005. Bias current generators with wide dynamic range. *Analog Integr. Circuits Signal Process.* **43**, 247–268.
- Delbrück T, Berner R, Lichtsteiner P, and Dualibe C. 2010. 32-bit configurable bias current generator with sub-off-current capability. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1647–1650.
- Gray PR, Hurst PJ, Lewis SH, and Meyer RG. 2001. *Analysis and Design of Analog Integrated Circuits*. 4th edn. Wiley.
- Ivanov VV and Filanovsky IM. 2004. *Operational Amplifier Speed and Accuracy Improvement: Analog Circuit Design with Structural Methodology*. Kluwer Academic Publishers.
- jAER. 2007. jAER Open Source Project, <http://jaerproject.org>.

- Lee TH. 2004. *The Design of CMOS Radio-Frequency Integrated Circuits*. 2nd edn. Cambridge University Press, Cambridge, UK.
- Linares-Barranco B, Serrano-Gotarredona T, and Serrano-Gotarredona R. 2003. Compact low-power calibration mini-DACs for neural massive arrays with programmable weights. *IEEE Trans. Neural Netw.* **14**(5), 1207–1216.
- Linares-Barranco B, Serrano-Gotarredona T, Serrano-Gotarredona R, and Serrano-Gotarredona C. 2004. Current mode techniques for sub-pico-ampere circuit design. *Analog Integr. Circuits Signal Process.* **38**, 103–119.
- Nicolson S and Phang K. 2004. Improvements in biasing and compensation of CMOS opamps. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 665–668.
- Razavi B. 2001. *Design of Analog CMOS Integrated Circuits*. 1 edn. McGraw-Hill, Inc., New York, NY, USA.
- Rincon-Mora G. 2001. *Voltage References: From Diodes to Precision High-Order Bandgap Circuits*. John Wiley & Sons.
- Scandurra G and Ciofi C. 2003. R- β R ladder networks for the design of high-accuracy static analog memories. *IEEE Trans. Circuits Syst. I: Fundamental Theory and Applications* **50**(5), 605–612.
- Serrano-Gotarredona R, Camunas-Mesa L, Serrano-Gotarredona T, Lenero-Bardallo JA, and Linares-Barranco B. 2007. The stochastic I-Pot: a circuit block for programming bias currents. *IEEE Trans. Circuits Syst. II: Express Briefs* **54**(9), 760–764.
- Vittoz E and Fellrath J. 1977. CMOS analog integrated circuits based on weak inversion operations. *IEEE J. Solid-State Circuits* **12**(3), 224–231.
- Widlar RJ. 1965. Some circuit design techniques for linear integrated circuits. *IEEE Trans. Circuit Theory* **12**(4), 586–590.
- Widlar RJ. 1969. Design techniques for monolithic operational amplifiers. *IEEE J. Solid-State Circuits* **4**(4), 184–191.
- Yang MH, Liu SC, Li CH, and Delbrück T. 2012. Addressable current reference array with 170 dB dynamic range. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3110–3113.

12

On-Chip AER Communication Circuits



This chapter covers the on-chip transistor circuits used for the communication fabric introduced in Chapter 2; in particular the asynchronous communication circuits for receiving and transmitting address events following the Address Event Representation (AER) protocol. AER circuits are needed for transmitters that emit address events, for example the sensors described in Chapters 3 and 4 and the multineuron chips in Chapter 7. The receivers decode the incoming address events and stimulate the corresponding synapses and neurons on multineuron circuits as described in Chapters 7 and 8. They can also be stimulated by outputs of commodity digital chips or artificial spike trains played from the computer via interfaces of the kind described in Chapter 13.

12.1 Introduction

The circuits for the early neuromorphic chips were first designed by Sivilotti (1991) and Mahowald (1994), students at the time in Carver Mead's laboratory at Caltech. These designs were influenced by the then on-going research in design of asynchronous circuits within the groups of Chuck Seitz and Alain Martin at Caltech. Designing asynchronous circuits was no easy matter at the time because design tools for asynchronous circuits were nonexistent. The Communicating Hardware Processes (CHP) design methodology introduced by Alain Martin, helped to facilitate the design of general asynchronous circuits (Martin 1986; Martin and Nyström 2006). This methodology involves the application of a series of program decompositions and transformations starting from a high-level specification of the intended system. As each step preserves the logic of the original program, the resulting circuit will be a correct logical implementation of the specifications. This methodology was used in Martin's laboratory to design various custom asynchronous processors including field-programmable gate arrays (FPGAs) and microcontrollers (Martin 1986; Martin et al. 1989, 2003).

Using this methodology, the on-chip AER circuits necessary on a transmitter (sender) for communicating address events asynchronously off-chip to a receiver have been constructed. Detailed information on how the AER transistor blocks are constructed following the CHP design methodology can be found in Boahen (2000, 2004a, 2004b, 2004c).

12.1.1 Communication Cycle

The block diagram in Figure 12.1 shows the organization of the various AER blocks within two-dimensionally organized transmitter and receiver chips. Signal flow between any two communicating blocks is carried out through the Request (R) and Acknowledge (A) signals. The initiation of a spike from a pixel in the transmitter starts a communication cycle first between the pixel and a block known as the row arbiter. The pixel does this by activating the row request RR line to the Y-arbitrer. Any active pixel on this row will also activate the same RR line. The Y-arbitrer block arbitrates among multiple active row requests and acknowledges only one of these requests by activating the corresponding RA row line. (Details of the arbitration

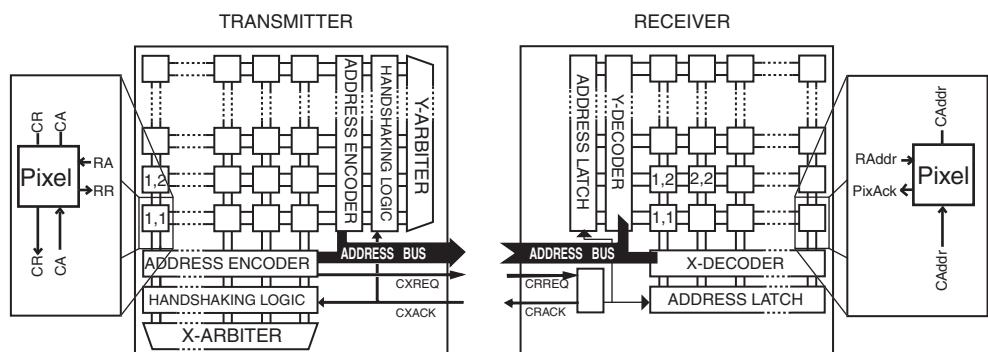


Figure 12.1 Block diagram of AER blocks on a 2D transmitter chip and 2D receiver chip. Details of signaling are described in the text. The signals CXREQ and CXACK can be connected to CRREQ and CRACK, if no translation of addresses from the transmitter are needed

process and circuits follow in Section 12.2.2.) The pixels on the row with an active RA signal then request in the column direction by activating their corresponding CR line to the X-arbitrer. The X-arbitrer then acknowledges one of these column requests through the corresponding CA line. The X- and Y-arbitrers also communicate with the X- and Y- encoders, which code the selected row and column addresses of the selected pixel. They also start a communication cycle with an on-chip handshaking logic block which handles the communication off-chip through CXREQ and CXACK. On receiving both the activated RA and CA lines, the pixel will then withdraw its request (i.e., it will no longer pull on these lines) from both arbiters. The Y-arbitrer does not start the next handshaking cycle with the remaining active row requests until the off-chip handshaking cycle is completed.

The off-chip communication cycle occurs between the transmitter and the receiver. The communication path in this case consists of both a data path (the digital address of the active pixel) and a control path (the request and acknowledge signals of the two devices). Following the single-sender, single-receiver signaling protocol in Figure 2.14, the data should be valid before the chip request R is activated. This sequence means that R can only be activated by the sender after a small delay from the onset of changing the address bits which encode the spike address to be transmitted. This protocol is also known as the *bundled data protocol* and is not truly delay insensitive as is required of standard asynchronous circuits. The delay-insensitive property is important in the circuit construction to ensure proper operation regardless of the final delay through the wires and transistors of asynchronous circuits in the fabricated device.

Most neuromorphic chips use the bundled data protocol because the circuits to generate the chip request signal are more compact than the delay-insensitive version. The delay in generating the chip request signal is usually done through a set of current-starved inverters and capacitors and this delay will vary after fabrication. The circuits have to be designed so that this delay variation is as small as possible for all process corners. The delay-insensitive version uses a dual-rail encoding scheme where each address bit is represented by two lines. One of the two lines is activated when transmitting a ‘0’ and the other line is activated when transmitting a ‘1’. A completion circuit then determines when one of the two lines for every address bit has been activated (Martin and Nyström 2006). The output of the completion circuit acts as the chip request signal. The trade-off between the bundled data protocol and the delay-insensitive version is that the dual-rail encoding scheme needs twice as many output pads and the additional completion circuits for generating the chip request signal.

The receiver on receiving an active CRREQ generates an active CRACK signal which is also used to latch the input digital address. The decoded X- and Y-addresses are then used to stimulate the corresponding pixel through activated row and column lines. When CRREQ is inactivated by the transmitter, CRACK is inactivated once the acknowledge signal of the pixel array (summed from all pixel acknowledges) is active, signaling that the pixel has received the decoded address. In some chips, the design is such that the pixel acknowledge is not used for the inactivation of the CRACK as a way of speeding up the communication cycle. This timing assumption is valid only if the on-chip decoding process is faster than the duration of the off-chip communication.

12.1.2 Speedup in Communication

The control flow of the request and acknowledge signals from a pixel on a transmitter to a pixel on a receiver is illustrated in Figure 12.2. In this diagram, the four-phase handshaking

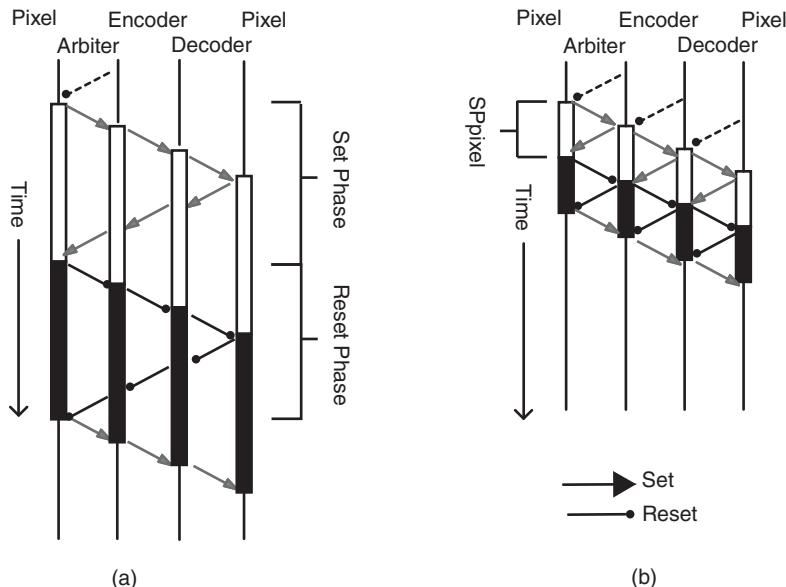


Figure 12.2 Control signal flow starting from a pixel through the arbiter and encoder on a transmitter to the decoder and a pixel on the receiver (left to right). The arrows show the activation of the requests between the various blocks (from left to right) and activation of the corresponding acknowledges (from right to left) in the Set phase and then first the inactivation of the request and the inactivation of the acknowledge signals in the Reset phase. (a) Completion of the *set* phase for the originating pixel before the handshaking is completed in the *reset* phase. (b) Pipelining reduces the overall handshaking time by allowing the signals to propagate forward in the set phase without waiting for the reset phase of the previous stage. SPpixel, the set phase of the pixel is shorter in duration than the set phase in (a). © 2000 IEEE. Reprinted, with permission, from Boahen (2000)

protocol shown in Figure 2.14 is assumed. When the Set phase of the pixel is initiated starting with the activation of the request from this pixel to the encoder and arbiter, this information is propagated all the way to a pixel on the receiver. The receiver pixel then activates its acknowledge signal, and this acknowledge information is transmitted all the way back to the pixel on the transmitter. When this cycle is completed, the Reset phase is executed starting from the inactivation of the request from the transmitter pixel all the way to the pixel on the receiver, whereupon the acknowledge signal from this pixel is now inactivated leading to the propagation of the inactivation of the acknowledge to the pixel on the transmitter. The communication time can be extremely long, since the request originating from a pixel on the transmitter has to be communicated all the way to a pixel on the receiver before the acknowledge from the encoder to the transmitter pixel can be activated. To speed up this communication cycle, pipelining is implemented where the Set phase is completed locally between two communicating processes without waiting for the activation of the request signal from a pixel on the sender to propagate all the way to the pixel on the receiver (Sutherland 1989). We return to this issue in Section 12.2.2.

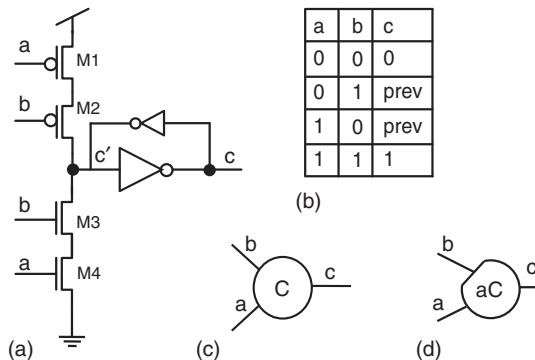


Figure 12.3 Muller C-element. (a) Transistor circuits. The feedback inverters hold the state c' when both inputs a and b do not have the same logic input state. (b) Truth table of this circuit means previous state. (c) Symbol for C-element gate. (d) Symbol for an asymmetric C-element gate in which the pFET M1 is not present in the transistor circuit shown in (a)

12.2 AER Transmitter Blocks

The three main types of handshaking blocks on a transmitter consist of the handshaking circuits within the pixel, the arbiter, and a chip-level handshaking logic block. The activation of signals in the handshaking protocol between two communicating processes is implemented through the Muller C-element gate shown in Figure 12.3a (Muller and Bartky 1959). The a and b inputs are driven by the request R and acknowledge A signals of a process. The output c does not change unless both a and b inputs are at 0 or 1. Because the output is not always driven, a staticizer consisting of two inverters connected in feedback holds the output state c' until a and b are at the same logic level. The truth table of this gate and its symbol is given in Figure 12.3b and 12.3c, respectively. If one of the pFETs or nFETs is removed, the circuit is known as an asymmetric C-element. In the case where the pFET M1 is removed as in (a), only the b input has both its up-going and down-going transitions checked and the corresponding symbol for this circuit is shown in (d). The asymmetric C-element is used in the fair arbiter circuit described in Section 12.2.2.

12.2.1 AER Circuits within a Pixel

The AER communication circuitry is frequently implemented through a neuron-like circuit, for example the Axon-Hillock circuit, but can be implemented by any circuit that generates digital pulses. In the Axon-Hillock circuit (see Figure 12.4), the input current I_{in} generated by the actual intended circuits for the function of the pixel, charges up a neuron-like circuit. The example in the figure assumes that I_{in} is generated by a synapse circuit. This current then charges up the membrane capacitance C_1 . When the membrane voltage V_m exceeds the threshold of the first inverter, the output of the second inverter, V_{sp} , becomes active and V_m is further increased by the positive feedback through C_1 and C_2 . V_{sp} drives M1 to pull down on nRR . When the RA line is activated from the row arbiter, transistors M2 and M3 activate

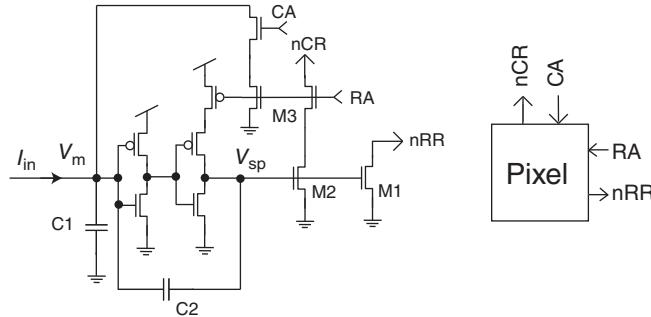


Figure 12.4 Example AER handshaking circuits within a neuron pixel. The neuron is modeled by the Axon-Hillock circuit. The handshaking signals generated by the circuit are the row request and acknowledge signals, nRR and RA, and the column request and acknowledge signals, nCR and CA, respectively. The prefix “n” on a signal name means that the signal is active low. Details of the circuit operation are described in the text

nCR. When the CA line is activated by the column arbiter, V_m is pulled low, thus resetting the neuron output V_{sp} , which in return stops pulling down nRR. This row line has a pullup (shown in Figure 12.11b) which returns nRR to V_{dd} when no pixel is requesting. This method is called a *wired OR* which replaces a conventional OR gate and reduces considerably the area required by replacing all the pull-up transistors with a single pFET. This pFET can be either connected to a fixed voltage (passive) or is actively driven by a peripheral handshaking circuit. The active-driven p-channel FET (pFET) scheme is described in Section 12.3.5.

12.2.2 Arbiter¹

The arbiter receives the row or column request lines from the pixel array through the arbiter interface block and arbitrates amongst the active lines (see Figure 12.5). The arbiter interface block handles the handshaking signals from the row (or column) requests and acknowledges and the ChipAck signal. It is discussed further in Section 12.2.3. For each dimension of row and column (assuming N rows or columns), the arbiter consists of a tree-like structure of $N - 1$ two-input arbiter cells, arranged in a tree with $b = \log_2(N)$ levels required to arbitrate between N pixels. Each two-input arbiter gate outputs a Req to the two-input arbiter at the next level, if any one of its two input Req signals is active. For example, the input request signals are labelled as r_{11} and r_{12} for the two-input gate A1 and the corresponding Ack lines are a_{11} and a_{12} . The output signal r_{21} from A1 goes to one of the two inputs of the next arbiter gate B1. This process continues to the top of the tree. The final selected output Req at the top of the tree then goes to an inverter, and the inverted output becomes the Ack signal that now propagates downward through the various gates until the corresponding RA line of an active RR line of a row of an array is activated by the arbiter interface block. For a 2D array, a second arbiter handles the CR and CA lines from the active pixels of the acknowledged row in the column dimension.

¹ Some of the text and figures in this section are adapted from Mitra (2008). Reproduced with permission of Srinjoy Mitra.

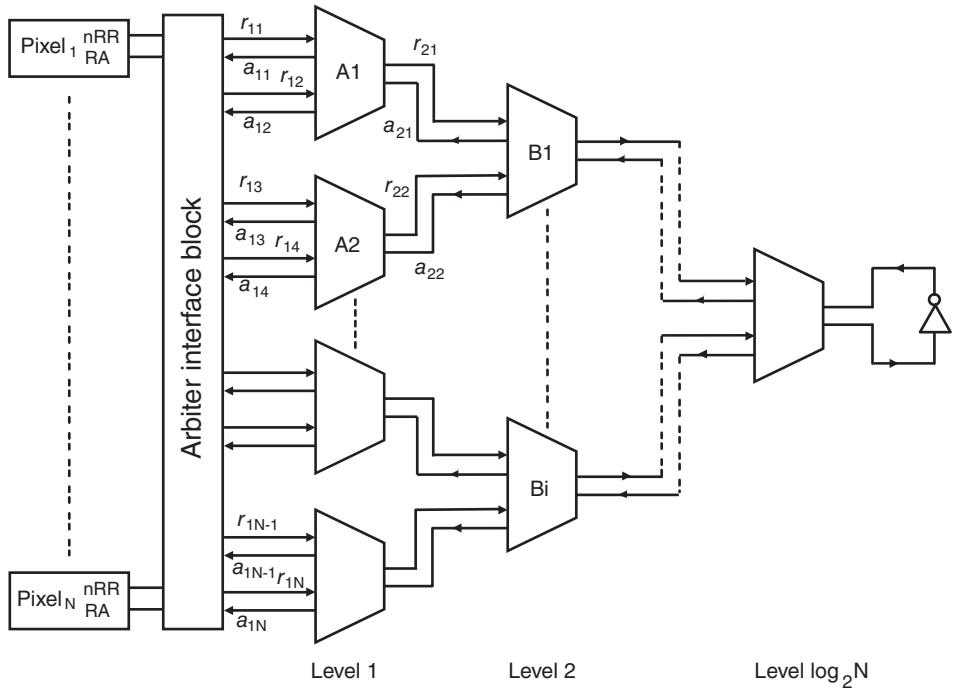


Figure 12.5 Communication interface between pixels of different rows and the arbiter tree of two-input arbiter gates. The circuits of the arbiter interface block are shown in Figure 12.11. The arbiter arbitrates amongst the nRR lines from the active rows in the array. In this figure, we assume that each row consists only of a single pixel

Arbiter circuits are designed to satisfy two criteria: low latencies and fairness. The first criteria ensures that incoming requests will be serviced as quickly as possible and the second ensures that requests are serviced in the order that they are received regardless of the position on the arbiter tree. The discussion of various arbiter architectures that trade-off between latencies and fairness are discussed in Chapter 2. We next discuss three different arbiters illustrating the implemented trade-offs in the circuits as they have evolved over time.

Original Arbiter

The original arbiter circuit used in the early neuromorphic chips is fair in its arbitration but slow in operation (Martin and Nyström 2006). The two-input arbiter gate is shown in Figure 12.6. Its core consists of an ME (Figure 12.6a). This element consists of a pair of cross-coupled NAND gates followed by inverters which are powered by the output of the opposite NAND gate (Mead and Conway 1980). When both inputs in1 and in2 are low corresponding to [0,0] in the truth table, the outputs o1 and o2 stay high. This condition is equivalent to both input requests to the arbiter gate being inactive. When one of the requests goes high, this condition corresponds to inputs [0,1] or [1,0] in the truth table and one of the ME outputs will

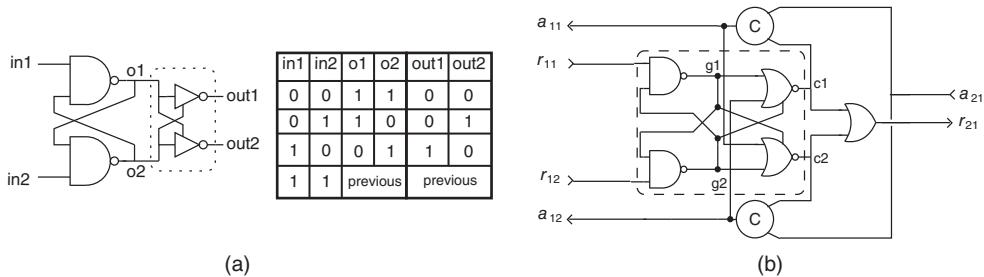


Figure 12.6 Implementation of a two-input arbiter gate. The core of the arbiter gate is the mutual-exclusive element (ME) block shown with its corresponding truth table in (a). The inverters in the ME are replaced by NOR gates in the input arbiter gate (b). Mitra (2008). Reproduced with permission of Srinjoy Mitra

go high signaling an active request to the next stage. When both input requests go high, the previous output state of the ME is retained.

The critical condition comes when both inputs in_1 and in_2 go high around the same time. The NAND outputs can temporarily go into a metastable state before the outputs settle into the eventual logic 0 and 1 case. The metastability comes from the inherent asymmetry of the physical implementation of the gates (Martin et al. 2003; Mitra 2008). The amount of time needed for the gate outputs to reach the final logic values depends on the time difference between both requests going high. A subsequent set of inverter gates which are powered by the outputs of the NAND gates ensure that the metastable outputs are not seen at the next stage. This metastable state can theoretically last for a long time but in practice, the circuit quickly stabilizes to the final logic values (Martin et al. 2003).

Two-Input Arbiter Gate

The operation of this gate, shown in Figure 12.6b, goes as follows: when one of the input requests becomes active ($r_{11} = 1$), c_1 goes high only if the acknowledge of the other input a_{12} is not active, that is, $a_{12} = 0$, thus ensuring that a second handshaking cycle is not initiated by this gate when it is in the middle of the handshaking cycle of its other input r_{12} . The output of the gate r_{21} goes high when either c_1 or c_2 is high. The corresponding input acknowledge goes high when the acknowledge from the next level of arbitration a_{21} goes high. The C-element gates ensure that both c_1 and a_{21} (and c_2 and a_{21}) go through the 4-phase handshaking cycle.

The ping-pong diagram of the timing of the handshaking signals between the first two levels of arbitration in the arbiter tree of Figure 12.5 is shown in Figure 12.7. The timing example illustrates the Set and Reset phases of the input request r_{11} after its activation and assumes that the arbiter tree only has two levels of arbitration (i.e., $N = 2$). If the input request r_{12} to gate A1 and r_{13} to the gate A2 become active during either the Set or Reset phase of r_{11} , then the requests will not be serviced till after the Reset phase of r_{11} . After the inactivation of the acknowledge signal a_{21} by gate B1, this gate is free to arbitrate again between its two active input requests r_{21} and r_{22} . If it chooses r_{22} , then the Set and Reset phases of the activation of r_{13} are as shown at the bottom of the ping-pong diagram.

The communication cycle of this arbiter is slow because the arbiter block at the first level has to wait till the handshaking cycle is completed in all the communicating processes of the

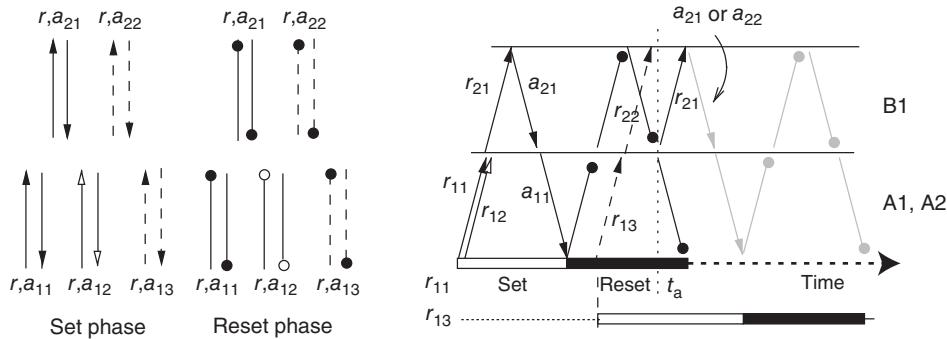


Figure 12.7 Timing of the handshaking signals between the first two levels of arbitration in Figure 12.5. The filled and unfilled black arrows show the activation of the handshaking signals of the arbiter gate A1 (r, a_{11} to r, a_{12} and r, a_{21}). The dashed arrows show the request and acknowledge signals for A2 (r, a_{13}). The inactivation of the handshaking signals in the reset phase are indicated by circles. The handshaking signals between the arbiter gates A1 and A2 to the arbiter gate B1 are depicted by (r, a_{21} to r, a_{22}). The labels A1, A2, and B1 refer to the gates in the arbiter tree shown in Figure 12.5. The Set and Reset phases for the activation of r_{11} and r_{13} are shown below the ping-pong diagram. Adapted from Mitra (2008) with permission of Srinjoy Mitra

arbiter tree before the acknowledge signal to this arbiter is activated. This design was used in the very early neuromorphic chips, including the retina of Mahowald (1994).

Unfair Arbiter

The original arbiter circuit was modified by Boahen (2000) to allow faster transmission of address events off-chip (see Figure 12.8). The core block is a modified version of the ME element and the circuit uses mostly standard logic blocks. When r_{11} is active, then the output NOR gate consisting of M1 to M4 will drive r_{21} high if the input to Mr is high, that is, the acknowledge a_{21} to this gate is inactive. Notice that a_1 and a_2 are still inactive until a_{21} becomes active. When a_{21} is active, then with the low state of g1, a_1 becomes active and together with the active r_{21} , a_{11} becomes active. In this design, even if r_{13} becomes active before r_{12} as shown in Figure 12.8b before the request from r_{11} enters the Reset phase, r_{21} remains active as long as both the inputs to M1 and M2 are not low. As long as the alternate input request to gate A1 becomes active during the Set phase of the other input request, the request of another row will not be serviced thus reducing the handshaking time of the pixel by not having to switch to a different row. One disadvantage of this arbiter design is that a row with high activity tends to hold on to the arbitration bus leading to the transmission of events from only part of the chip.

Fair Arbiter

Boahen proposed a fair arbiter circuit to replace the greedy arbiter circuit (Boahen 2004a). In this new scheme, a pixel that makes consecutive requests to the arbiter is not revisited until the rest of the tree has been serviced (see Figure 12.9). The new circuits also include a pipelining

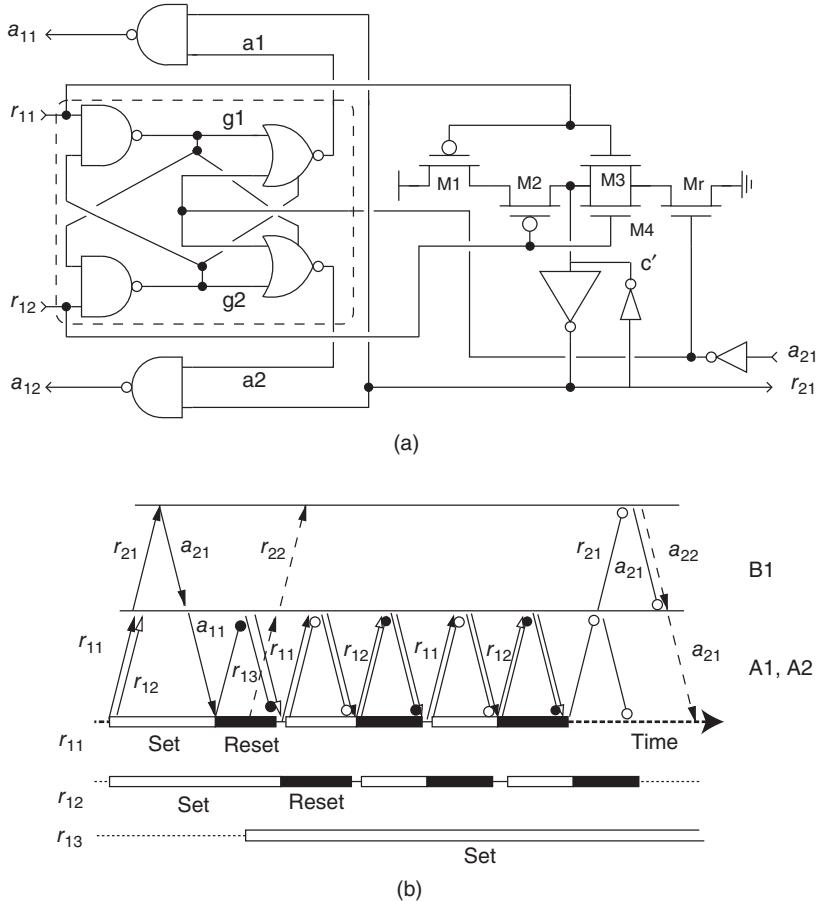


Figure 12.8 Unfair arbiter. (a) Circuit blocks for 2-input arbiter gate A1. (b) Timing diagram of a 2-level arbitration. The request r_{21} from A1 to the next stage stays active as long as r_{11} or r_{12} is active. The acknowledge a_{21} is inactivated only if neither r_{11} or r_{12} is active. The active request r_{13} to the A2 arbiter gate will not be serviced until both request inputs of gate A1 are inactivated. Adapted from Mitra (2008) with permission of Srinjoy Mitra

scheme, which allows new requests to propagate up the tree while old ones are still being serviced at the lower levels. These arbiter cells do not need to wait for requests from their parents to be cleared before resetting their acknowledge signals as seen in Figure 12.9b where $c1$ (output of the aC gate connected to r_{11}) ORed together with $c2$ acts as the request to the next level of arbitration. In the timing diagram, we see that after acknowledge a_{11} is inactivated and even after r_{11} becomes active again, the request r_{21} goes low only after r_{12} goes low. This request is not serviced again because $c1$ cannot be set high by the aC gate unless na_{21} is also high.

Figure 12.10 show the recorded AER spikes from two 32×32 multineuron chips with different arbiter types, an unfair arbiter and a fair arbiter. The spikes are in response to a common input current injected into all neurons. What is expected of the output is an identical

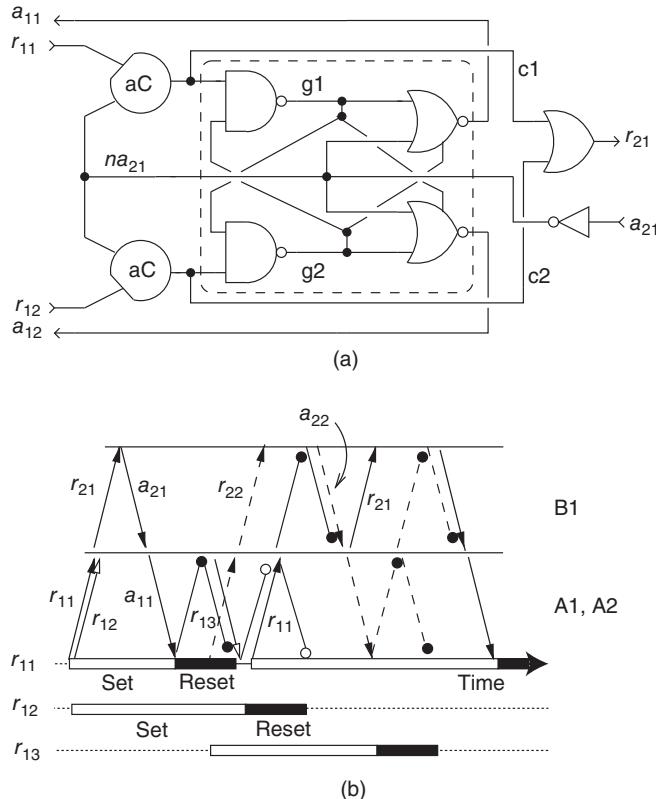


Figure 12.9 Fair arbiter. (a) Circuit blocks for 2-input arbiter gate A1. The output of the asymmetric C-element (aC) goes high only when both r_{11} and na_{21} are high but it goes low when r_{11} is low. (b) Timing diagram for the same 2-level arbitration cycle. Adapted from Mitra (2008) with permission of Srinjoy Mitra

recorded firing rate (except for mismatch) from all neurons but as we can see in the case of the unfair arbiter in (a), only spikes from half of the array were serviced by the arbiter. As the input current increases thus more spikes (and requests) are generated, the arbiter only serviced now a quarter of the array (b). In the case of the chip with a fair arbiter, we see a profile where all pixels show a more even distributed recorded rate demonstrating that the pixels are serviced roughly equally often, even with high pixel firing rates as shown in (c)-(d). The maximum spike rate of each pixel is limited by the refractory period of the event-generating circuit within the pixel.

12.2.3 Other AER Blocks

The AER arbiter interface block provides an interface between either the row or column Req and Ack lines from the 2D array, the arbiter, and the chip-level handshaking signal nCXAck. Figure 12.11 shows the interface circuits between the nRR and RA lines and the signals to the

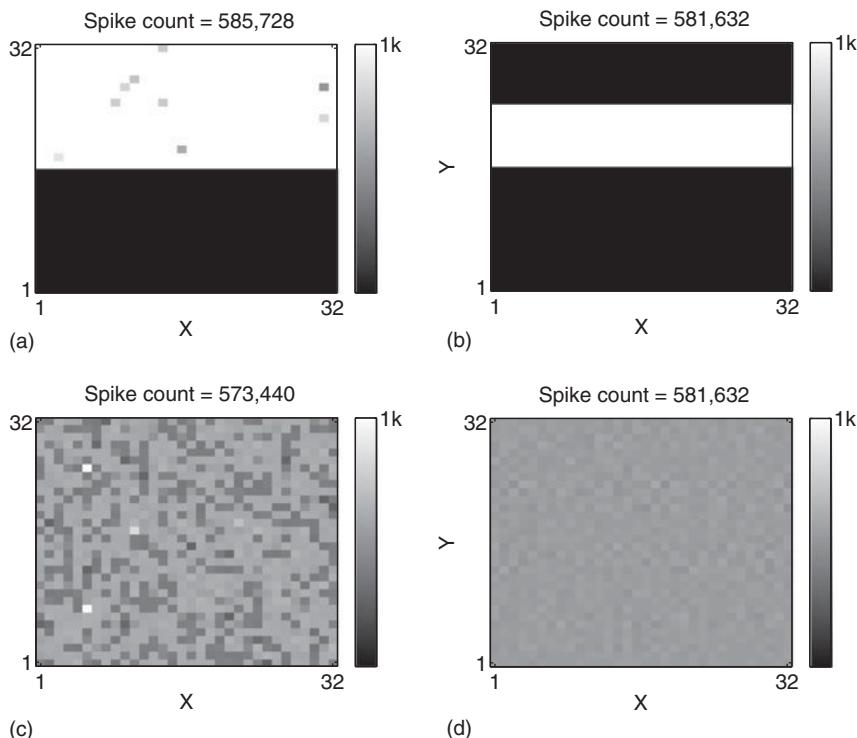


Figure 12.10 AER spikes recorded from two aVLSI chips using an unfair (a)–(b) and a fair arbiter (c)–(d), respectively. The figures show how the profile of recorded spikes resulting from a higher spike rate from all neurons receiving a common input current changes depending on the type of on-chip arbiter. Maximum spike rate is 1k events/second per pixel. Adapted from Mitra (2008) with permission of Srinjoy Mitra

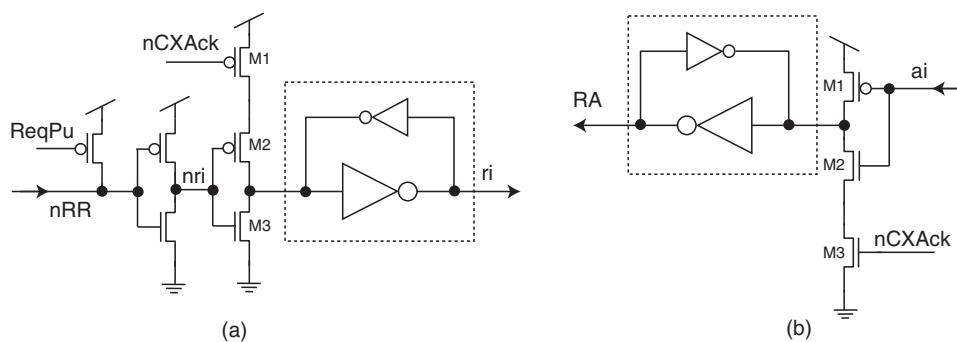


Figure 12.11 Arbiter interface circuit between the row request and the arbiter. The circuit in (a) shows how an active low nRR signal generates a request to the arbiter (ri). The pFET driven by $ReqPu$ returns nRR high when no pixels are requesting. The request to the arbiter is not taken away until $nCXAck$ becomes inactive. The circuit in (b) activates RA only if the arbiter returns an active acknowledge (ai) and $nCXAck$ is inactive. The staticizers (the feedback inverters within the dotted boxes) and transistors $M1$ to $M3$ in (a) and (b) form asymmetric C-elements

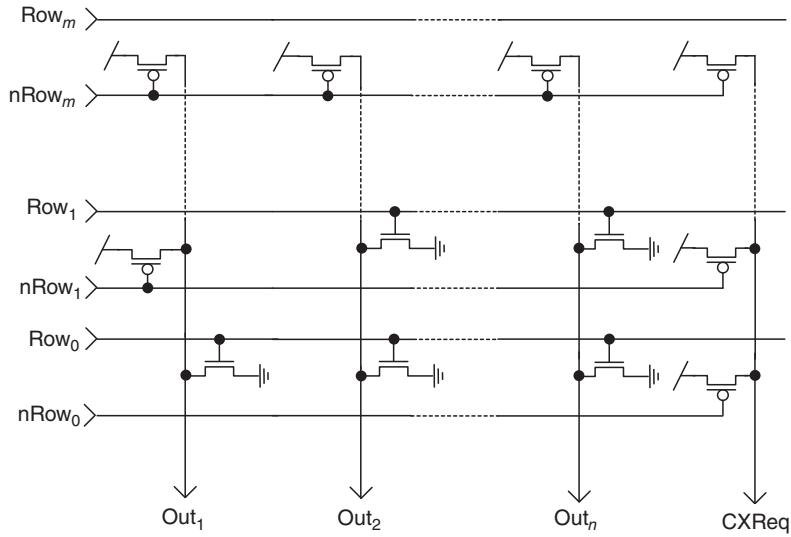


Figure 12.12 One-dimensional encoder circuit. An active high Row line pulls down the corresponding Out bit line through the connected n-channel FET (nFET). Any active Row line will drive CXReq high

arbiter. The interface circuits ensure that a new row can be selected by the arbiter only after the address of the sending pixel has been transmitted and recorded off-chip.

In addition, the row and column lines are encoded into a digital address. AER encoders were previously discussed in Section 2.2.1 of Chapter 2. The encoding circuit in Figure 12.12 takes m input lines and transforms the inputs into $n = \lceil \log_2(m) \rceil$ bits of output address. Every Row line or its complement drives the corresponding Out line through an nFET or pFET. The chip request CXREQ is driven high by any active Row for a 1D array and any active Col for a 2D array.

12.2.4 Combined Operation

The control signal flow between the various blocks in the transmitter is shown in Figure 12.13. The pixel makes a request to the arbiter through RR. Once it has received an active RA, it activates CR. Once CA is active, CXREQ is activated and the encoded pixel address is also ready for transmission. When the handshaking logic receives the corresponding active CXACK, both RA and CA are inactivated. The inactivation of CA leads to the inactivation of CXREQ and the inactivation of CXACK from the receiver. Only after the completion of the chip-level handshaking cycle can the interface handshaking block allow the next request and acknowledge cycle between the pixel and the arbiter to start again.

This timing follows the word parallel AER protocol described in Section 2.4.3 of Chapter 2. A faster readout scheme known as the word serial addressing scheme (Section 2.4.4 of Chapter 2) sends one row address and then column addresses of all active pixels on this row (Boahen 2004c).

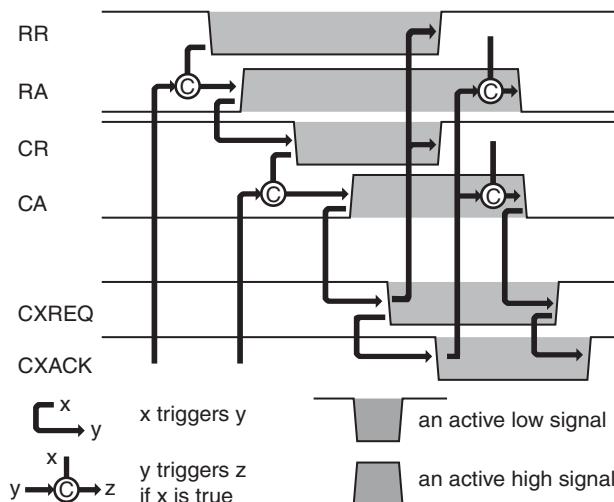


Figure 12.13 Timing of request and acknowledge signals for all blocks in the transmitter following the word-parallel signal protocol. © 2008. IEEE. Reprinted, with permission, from Lichtsteiner et al. (2008)

12.3 AER Receiver Blocks

The schematic of the various on-chip AER blocks for a receiver with a 2D array is shown in Figure 12.14. The control path depicts how the chip-level handshaking signals CXREQ and CXACK are handled and the data path depicts how the address bits are handled in conjunction with the handshaking signals. These different blocks are discussed in more detail in Section 12.3.1 to Section 12.3.3.

12.3.1 Chip-Level Handshaking Block

The chip-level handshaking block handles the communication between the external transmitter (sender) and the chip (receiver). The chip-level handshaking signals of the receiver consist of CRREQ, CRACK, and the digital pixel address as shown in Figure 12.1. In an array of multineuron chips, this digital address encodes a unique synapse and neuron address on chip.

The communication between the handshaking block and the decoder in the circuits presented here follows the 4-phase handshaking protocol. The CRACK signal is normally activated as quickly as possible after CRREQ is active. This CRACK signal is also used to latch the address bits. There are two ways in which the CRACK is inactivated once CRREQ becomes inactive. The first approach used to speed up the communication time in early neuromorphic chips and still in use in many current chips is to deactivate CRACK as soon as CRREQ is deactivated. This violation of the handshaking protocol (since there is no check whether the decoded address has reached a pixel) is done on the assumption that the communication between the remaining blocks on-chip will be faster than the communication between the sender and the receiver.

This violation should not affect the communication of the address to the targeted pixel on-chip but if the communication on-chip slows down, subsequent address events from the

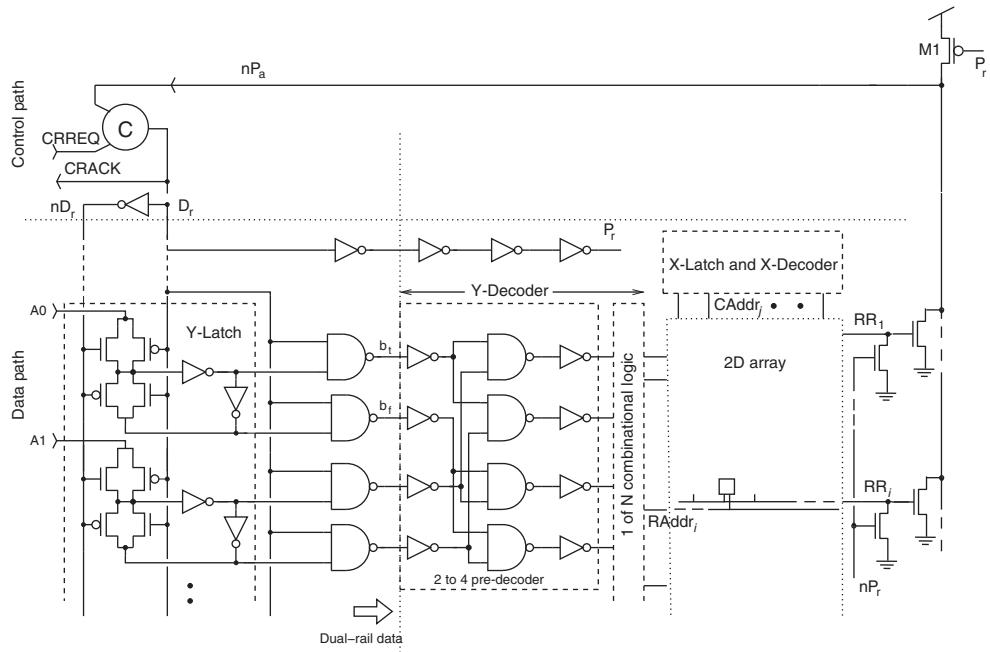


Figure 12.14 Schematic of the AER blocks within the receiver. When CRREQ is active and CRACK is inactive, then CRACK and D_r becomes active. The address bits are latched by D_r. The pre-decoder blocks decode the latched bits and the outputs of the pre-decoder blocks are then combined across the N outputs of the pre-decoder blocks using the 1 of N combinational logic to select one active row address. A similar operation is carried out in the X-latch and decoder. The selected pixel activates the row request line RR. The row request signals are then OR'd together. The active pull-up M1 is driven by P_r. This pullup is inactivated when CRACK is inactive. Once one of the row request lines is activated, nP_a is activated, leading to an inactivation of CRACK when CRREQ is inactive. Adapted from Mitra (2008) with permission of Srinjoy Mitra

transmitter will be affected. In that situation, the user has to determine whether the loss of spikes that come close together in time, for example, events that come in a burst even if the overall event rate is below the bus bandwidth, is a problem for the application. The second approach assures that the pixel did receive the spike by the use of additional circuits that generate a PixAck signal which is activated by the pixel which receives both active CA and RA signals. The PixAck circuits are described in Section 12.3.3. The PixAck signal from all pixels are then combined together with an inactive CRREQ to drive the inactivation of CRACK.

12.3.2 Decoder

The decoder converts the latched N bit address into a one hot-code of 2^N lines corresponding usually to the address of a pixel within an array. The address bits are first latched by using the CRACK signal and its complement. These bits then drive decoder circuits. The early decoder circuits used a long chain of AND transistors as shown in Figure 12.15 (Boahen

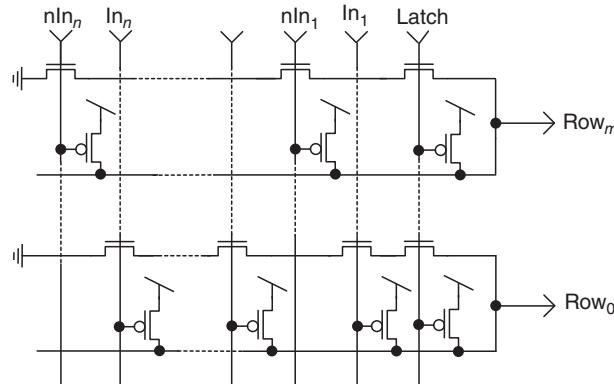


Figure 12.15 Decoder circuits which take n input In bit lines and output $m + 1$ Row lines. The Latch signal is generated from CRACK

2000; Mahowald 1994). The advantage of this decoding style is the simplicity of the design. However as arrays grow in size, this decoding circuit suffers from the increased capacitance that the address lines see and the wiring capacitance that each decoding gate has to drive. These capacitance issues are described in Mead and Conway (1980) and further discussed in Mitra (2008). By adopting a decoding style used for example in large memories, the capacitance issue can be minimized in large arrays. Newer pre-decoder circuits decode the N address bits through a combination of two to four pre-decoder blocks and three to eight pre-decoder blocks. The two to four pre-decoder block consists of a set of 2-input NAND gates (an example is shown in the Y-decoder block of Figure 12.14) and the three to eight pre-decoder block uses a set of 3-input NAND gates.

12.3.3 Handshaking Circuits in Receiver Pixel

The pixel includes circuits, which on activation of its CA and RA input signals generate an active nPixAck pulse. An example circuit is shown in Figure 12.16. The ANDing of CA and RA can be used as an input spike in the case of a synapse pixel. M1 biased by V_{pu} returns nPixAck to V_{dd} after CA and RA go away. M4 is part of a wired NOR along the row with a single pulldown nFET at the end of the row; RR is low in the inactive state and is pulled high when any pixel has received an input. Transistors M5 to M8 and C_{syn} form a current-mirror, diode-connected synapse. The transistor M5 biased by V_w determines the synapse current I_{out} that flows to the neuron membrane potential.

The circuit eliminates the problem whereby high-frequency switching of either RA or CA alone can lead to M2 turning on through the small parasitic capacitance, C_{p2} (Boahen 1998). If M5 were connected in series with M2 and M3, I_{out} can flow even though only one of the transistors M2 or M3 is turned on. This current can be much larger than the pA current levels typical of synaptic currents.

This circuit also eliminates the problem of capacitive turn-on, which happens when the CA line switches a lot and the voltage changes are transmitted to the nPixAck node through

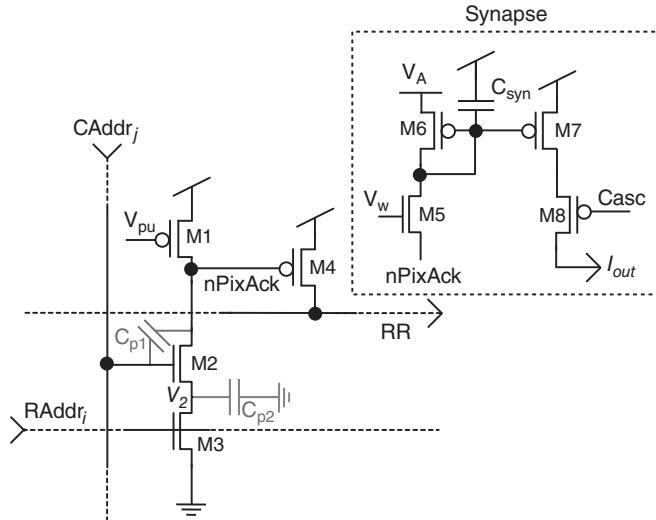


Figure 12.16 Handshaking circuits in a pixel with a synapse. The transistor M1 driven by V_{pu} together with transistors M2 and M3 produce a digital output swing on $nPixAck$ and ensure $nPixAck$ will not be charged through capacitive coupling. Transistors M5 to M8 form a current-mirror integrator synapse. The output current of the synapse I_{out} , flows only when the pixel is selected

the drain-gate overlap capacitance, C_{p1} , bringing its voltage to a few tenths of a volt below ground. Current can flow through transistor M5 even if V_w is at ground. In this design, the source voltage of M5 is brought to V_{dd} when the pixel is not selected.

12.3.4 Pulse Extender Circuits

Because of the requirement for the AER handshaking circuits to have a short latency (<10 ns), the use of the pulse from the Figure 12.16 as the input pulse is too short for a synapse circuit to generate a finite rise and fall time in the order of milliseconds and also allowing for subthreshold currents.

12.3.5 Receiver Array Peripheral Handshaking Circuits

Receiver array peripheral handshaking circuits ensure that a pixel receives its decoded address before CRACK is inactivated at the chip level. The row request RR lines each have an active pulldown circuit that is driven by nP_r , a signal which is generated by the chip-level C-element in Figure 12.14 and is a delayed version of the CRACK signal. The row RR signals drive another wired-OR global line which has an active pullup driven by P_r . When CRACK is activated, the active pulldowns of the row lines and the active pullups of the global array acknowledge line are disabled. P_r goes high, thus preventing it from pulling up the array acknowledge signal nP_a . Any decoded pixel will pull its corresponding RR line high and the global line nP_a low.

This transition then leads to the inactivation of the CRACK once the CRREQ signal is also inactivated.

12.4 Discussion

The design of asynchronous circuits is not easy because of the lack of accessible and comprehensive design tools. The asynchronous tools assume logic signals without any assumption of rise and fall times in the physical circuits. On the physical device, signals can transition over a time of several tens to hundred nanoseconds depending on the wiring and parasitic capacitances. In such cases, the asynchronous design can fail at the silicon level if the transitioning order of signals is not preserved. Additional inverters are needed to sharpen up the edges of the signals which see large capacitances but the slow changing inputs to the inverters will also lead to higher power consumption.

As designs scale to larger arrays, the row and column line capacitances can lead to significant rise times and fall times for the logic signals and in addition a difference between the rise time of a global signal seen on one side of the array versus the other side of the array. To circumvent such problems, circuits have to be designed for example, in such a way as to ensure that pixels on both edges of the array have seen the level changes in the row and column request and acknowledge lines before the next sequence of handshaking signaling is carried out as shown in Figure 12.17.

Another area of improvement is to move from the bundled-data protocol which is delay-sensitive to an AER link which is delay-insensitive. The first step is to go to a dual-rail encoding scheme for the signals as in normal asynchronous circuits. In this scheme, each signal uses two lines, a valid signal of 1 leads to one line going high and a valid signal of 0 leads to the other line going high. An invalid signal leads to both lines being low and an incomplete signal is indicated by both lines being high. In the communication of the address bits between a transmitter and receiver, the ChipReq signal can be removed and the global address bits can be combined together to create an equivalent of a ChipReq (Martin and Nyström 2006). In this way, there is no delay which needs to be specifically introduced in the generation of the ChipReq.

A second scheme is to use a one-hot code for encoding the addresses off-chip. A one-hot code uses 2^b lines in encoding b bits. For example, the four combinations of two bits are encoded by four lines. Every combination is then indicated by the activation of only one of the four lines. The address bus is then divided into groups of four where one out of four bits

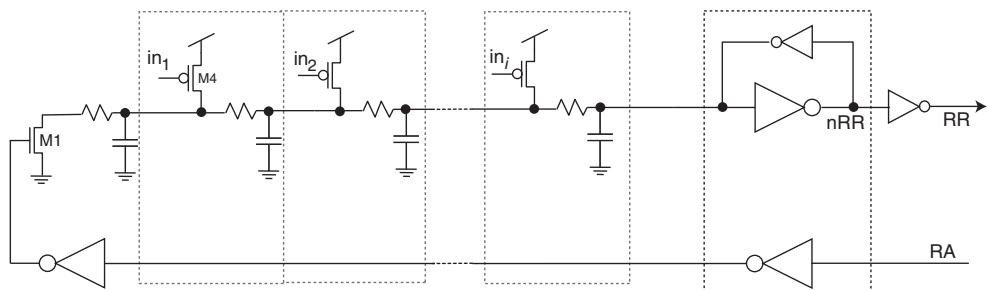


Figure 12.17 Wired OR for the row requests with an active pulldown. In the receiver, the in signals driving the pFETs come from the local nPixAcks. Because of the resistance and capacitance of the row lines, the use of an active pulldown M1 at the far end of the line is needed to guarantee correct behavior

is activated in each group. The validity and neutrality of the different groups can be checked by a binary tree of m 4-input OR gates (if only groups of one in four are used). This scheme was first introduced in the design of Neurogrid (Lin and Boahen 2009). The authors suggested that they could also save in the number of power pads which are normally interleaved between each address bit pad by using the one-hot code scheme as the address space scales up.

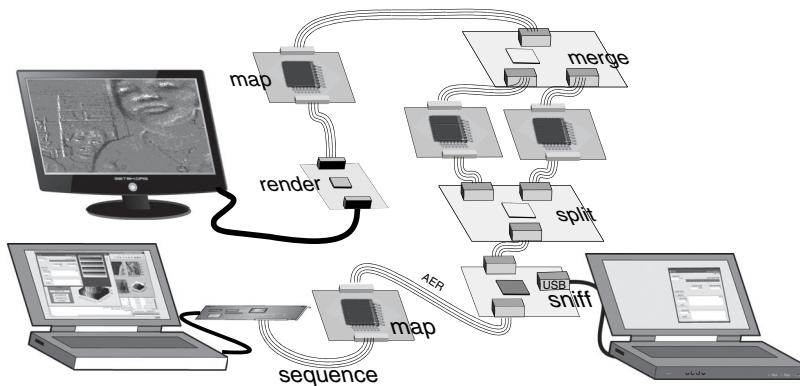
Finally, the AER bandwidth of the chip can affect system specifications and continual improvement is needed in both the signalling and the circuit domains. Even though the cycle time for a read-out has decreased from 2 μs (computed for a 64×64 array in a 2 μm process) (Mahowald 1992) to a range of 30–400 ns (computed for a 104×96 array in a 1.2 μm process) (Boahen 1999), this bandwidth needs to be increased as arrays scale up to more than a million pixels. The AER circuits to implement a word-serial readout as described in Boahen (2004a, 2004b, 2004c) (and briefly summarized in Section 2.4.4 of Chapter 2) are examples of improvements in AER readout to increase bandwidth.

References

- Boahen KA. 1998. Communicating neuronal ensembles between neuromorphic chips. In: *Neuromorphic Systems Engineering* (ed. Lande TS). The International Series in Engineering and Computer Science, vol. 447. Springer. pp. 229–259.
- Boahen KA. 1999. A throughput-on-demand address-event transmitter for neuromorphic chips. *Proc. 20th Anniversary Conf. Adv. Res. VLSI*, Atlanta, GA, pp. 72–86.
- Boahen KA. 2000. Point-to-point connectivity between neuromorphic chips using address-events. *IEEE Trans. Circuits Syst. II* **47**(5), 416–434.
- Boahen KA. 2004a. A burst-mode word-serial address-event link—I: transmitter design. *IEEE Trans. Circuits Syst. I: Reg. Papers* **51**(7), 1269–1280.
- Boahen KA. 2004b. A burst-mode word-serial address-event link—II: receiver design. *IEEE Trans. Circuits Syst. I: Reg. Papers* **51**(7), 1281–1291.
- Boahen KA. 2004c. A burst-mode word-serial address-event link—III: analysis and test results. *IEEE Trans. Circuits Syst. I: Reg. Papers* **51**(7), 1292–1300.
- Lichtsteiner P, Posch C and Delbrück T. 2008. A 128×128 120dB 15us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* **43**(2), 566–576.
- Lin J and Boahen K. 2009. A delay-insensitive address-event link. *Proc. 15th IEEE Symp. Asynchronous Circuits Syst. (ASYNC)*, pp. 55–62.
- Mahowald M. 1992. *VLSI Analogs of Neural Visual Processing: A Synthesis of Form and Function*. PhD thesis. California Institute of Technology, Pasadena, CA.
- Mahowald M. 1994. *An Analog VLSI System for Stereoscopic Vision*. Kluwer Academic, Boston.
- Martin AJ. 1986. Compiling communicating processes into delay-insensitive VLSI circuits. *Distrib. Comput.* **1**, 226–234.
- Martin AJ and Nyström M. 2006. Asynchronous techniques for system-on-chip design. *Proc. IEEE* **94**(6), 1089–1120.
- Martin AJ, Burns SM, Lee TK, Borkovic D, and Hazewindus PJ. 1989. The design of an asynchronous microprocessor. *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference*, pp. 351–373.
- Martin AJ, Nyström M, and Wong C. 2003. Three generations of asynchronous microprocessors. *IEEE Des. Test Comput.* **20**(6), 9–17.
- Mead CA and Conway L. 1980. *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA.
- Mitra S 2008 *Learning to Classify Complex Patterns Using a VLSI Network of Spiking Neurons*. PhD thesis. ETH Zurich.
- Muller DE and Bartky WS. 1959. A theory of asynchronous circuits. *Proc. Int. Symp. Theory of Switching, Part 1*, pp. 204–243.
- Sivilotti M. 1991. *Wiring Considerations in Analog VLSI Systems with Application to Field-Programmable Networks*. PhD thesis. California Institute of Technology, Pasadena, CA.
- Sutherland IE. 1989. Micropipelines. *Communications of the ACM* **32**(6), 720–738.

13

Hardware Infrastructure



To make use of custom address event (AE) based neuromorphic chips built using the circuits described in the previous chapters, they need to be embedded into a larger hardware infrastructure. This chapter describes some of the considerations which must be borne in mind when designing, building, and operating the printed circuit boards which form this infrastructure for relatively small-scale, chiefly experimental systems. Examples are given taken from several projects. The necessary accompanying software is described in Chapter 14. The present chapter also includes a section reviewing the use of field programmable gate arrays (FPGAs) in neuromorphic systems. Hardware infrastructure for larger systems containing more than a handful of neuromorphic chips is discussed separately in Chapter 16.

13.1 Introduction

As we have already seen in Chapter 2, in order to construct larger AER systems, a certain amount of hardware infrastructure ‘glue’ is necessary between the active computational elements.

In fact even for the simplest systems, at least some means of capturing AE data is required. General purpose commercially available data acquisition boards are usually ill-suited to perform the asynchronous handshaking expected by an AER sender, and do not record the times at which individual events are received. For the purpose of capturing data from a simple experiment with perhaps one AER sender device and one AER receiver device, or for debugging purposes on parts of larger systems, a logic analyzer can easily be used to record the AE stream on a parallel bus. Logic analyzers, however, are expensive, have limited memory depth, and usually operate in a batched mode. This means that while they are acquiring data, this data are not available for further processing. Only when an acquisition is complete, or between acquisitions can the data be read into a computer. Thus logic analyzers are not ideally suited for the continuous processing and analysis of AE streams, and specialized AER receiver devices, often known as *monitors*, will be required to interface to conventional digital hardware.

If an AER system does not contain a sensory input device such as a retina chip or cochlea chip, it will typically require an external source of AEs for stimulation. Even in systems which do contain sensory input devices, it is often useful or necessary to be able to provide AE stream input either as a replacement for or in addition to input from those sensory devices, for instance for debugging, or in order to provide repeatable input for the purposes of an experiment. For this stimulation of AE systems, good off-the-shelf solutions are difficult to find. Most general purpose commercially available digital output boards are again ill-suited because neither are they designed to perform the asynchronous handshaking expected by an AER receiver nor are they necessarily designed to preserve the timing between output words. Thus, specialized AER sender devices are required to allow conventional digital hardware to produce AE output. These devices are often known as *sequencers* because they generate a sequence of AEs.

Section 2.4 referred to idealized address-event receiver blocks within which source addresses (the addresses of source neurons) must be converted to multiple destination addresses (the addresses of individual target synapses), thus performing a fan-out function. Even if no fan-out is required, it is usual that some address translation will be required between sending and receiving devices, since unless sender and receiver have been conceived of together as a pair, it is unlikely that their address spaces will match one to one. In practice, chip-level devices that conform to this idealized model of performing address translation and/or fan-out internally have only relatively recently been constructed (Lin et al. 2006), and are still not the norm. Without these, there is a need for separate AER devices that act as receivers on one bus and senders on another performing any desired address translation and fan-out in between. Such devices are often called *mappers*. If a mapper has multiple possible output buses, it may also be called a *router*. Being able to perform programmable address translation between a sender and receiver also provides the very important feature of reconfigurability. Not all source to destination connections must be decided at the time of manufacture of the devices. General purpose devices can be constructed and the precise connectivity between their elements can be determined and modified, after they have been built into systems, and learning in which new connections are made and others eliminated at run-time is facilitated.

Further operations on AE streams which can be supported by interface logic include *splitting*, in which events on one incoming bus are copied to more than one outgoing bus, and the inverse, *merging*, in which events are taken from multiple incoming buses and copied to one outgoing bus. These functions can also be implemented by mappers having more than one input or output bus, respectively. (Merging can alternatively be seen as an essential internal part of any AER device which must receive AEs from more than one input bus—there must be some form of arbitration between the incoming buses, unless events are to be dropped in the case of events arriving on multiple buses at the same time.)

Thus we have three major classes of interface logic functionality used between AER modules: *monitors*, *sequencers*, and *mappers* (these terms were introduced in Dante et al. 2005). Combinations of these are of course also possible and are frequently encountered.

13.1.1 Monitoring AER Events¹

An AER monitoring tool should be able to capture long sequences of AEs, interfering as little as possible with the system under test. As events arrive asynchronously and at a wide range of possible rates, depending on system activity, they need to be buffered, typically in dedicated First-In First-Out (FIFO) memory before being further processed or recorded in long-term memory. The FIFO decouples the reception of the incoming AEs from the operations of the host reading those events over some bus, for example, PCI or USB, the use of which may be shared with other peripherals. In the ideal case, the FIFO will be read often enough that it never fills up, or overruns, such that events are lost. A mechanism is usually provided to alert the user if and when such an overrun does occur. Key to avoiding overruns is getting the data out fast. Early monitors, for example, the on Rome PCI–AER board described in Dante et al. (2005), used polled and/or interrupt driven I/O to transfer data from the board, but could not do bus mastering or direct memory access (DMA, a feature that allows peripherals to transfer data to and from memory without involving the processor). This was the main factor limiting its performance to approximately 1 Meps. In some cases this may be acceptable, but many AE chips can produce much higher spike rates. To increase performance, techniques such as DMA and bus mastering are required such that the host CPU is not involved in merely copying AE words and time-stamps out of the monitor FIFO. Otherwise, where high bandwidth monitoring is required, a stream of events and high resolution time-stamps can be captured into RAM local to the monitoring device for later, off-line processing, as was done in the CAVIAR USB–AER board (Gomez-Rodriguez et al. 2006).

Monitor functionality is usually implemented in an FPGA.

Data-Width, Channel Addressing, and Framing Errors

Monitors must obviously be constructed to capture the full width of the AE words in use, but it makes little sense to make a monitor that captures, for example, exactly 21-bit AE words because a particular sender happens to produce 21-bit words. Monitored data will sooner or later be stored in a conventional computer RAM system having a data width of 2^n 8-bit bytes.

¹ Parts of the text in this section are © 2006 IEEE. Reprinted, with permission, from Paz-Vicente et al. (2006).

Thus in our 21-bit example, the data is going to be stored in at least 32-bit words, so it makes sense to construct a more general monitor which can capture up to 32-bit wide AE words. However, any bits which are not driven by the sender must be set to some particular value, typically zero, such that in our 21-bit example, the whole 32-bit word is always consistently guaranteed to be the same for the same 21-bit value.

Some monitors allow the simultaneous monitoring of multiple devices on multiple channels and allow these devices to be distinguished by placing a different value for each channel in otherwise unused high-order address-bits. For example, the Rome PCI-AER board (Dante et al. 2005) could be configured to monitor one 16-bit sender, two 15-bit senders, or four 14-bit senders.

Whenever AE words are to be transmitted over an interface which is less wide than the words in question, one must guard against so-called *framing errors* whereby the distinction between the inter- and intra-word boundaries may be lost leading to mistaking, for instance the high-order part of one word with the low-order part of the next. This is particularly true across interfaces where the data flow may be interrupted and resumed by the reader. An example of this problem and its resolution occurs in the AEX board (Fasnacht et al. 2008) in which 32-bit AE words are transmitted over serial buses using 16-bit Serializer–Deserializer (SerDes) chips. In order to detect the 32-bit word boundary, it is defined that the two 16-bit words must be sent back-to-back with no IDLE characters in between. Once an IDLE character is seen, the receiver knows where the 32-bit word boundary is. This allows 32-bit words also to be sent back-to-back, once the receiver has seen a single IDLE character, thus the full bandwidth available can still be used for AE data.

Inter-Spike Intervals and Time-Stamps

Since neurons code information not only in their identity but also in the frequency or timing of their spikes, pixels that transmit their addresses over an AER bus also code information in the frequency or timing of appearance of those addresses on the bus. Therefore, the inter-spike intervals (ISIs) are critical for this communication mechanism and monitor needs to preserve the information conveyed in the ISIs of the AER which would otherwise be lost as soon as the incoming addresses are buffered. This is done by recording in the buffer some form of time-stamp, that is, the value of a timer, with the address of each event, as soon as it arrives. Time-stamps can be relative to the previous event arrival time, in which case they directly represent an ISI, or they can be so-called absolute, in which case they are actually relative to some earlier timer reset time, typically the time when the monitor hardware was powered up, or at the beginning of an experiment. Note that in the case of relative time-stamps, the ‘ISI’ to which they refer is an interval between two spikes which are not necessarily, indeed usually not, from the same neuron, but simply between two spikes from any two neurons which report their spikes onto the same bus. This may lead to difficulties later in processing the events, since if any events and their associated relative time-stamps are disregarded, the spike timing of the remaining events can no longer be faithfully reconstructed. Therefore the use of so-called absolute time-stamps, where the time recorded alongside the event address is unambiguous, is more usual. A source of ambiguity remains, however, in that we may not know exactly when the time-stamp timer was reset in relation to other clocks in the system, for example, a host PCs system clock, the timer of another monitor, wall-clock time, or the time of a stimulus

onset in an experiment. Achieving correspondence between clocks is not trivial even if we can command a reset of the time-stamp timer. Even if we can do this, given the vagaries of operating system schedulers and PC hardware, we cannot be certain when a PC-initiated timer reset actually takes place. Ideally a time-stamp timer should be not only resettable, but also readable. This does not solve the problem completely, since we will still not know the exact time at which the timer is read, but at least then multiple readings of the timer can be taken and correlated against the PC system clock without having to reset the timer.

Synchronization between Monitors

When multiple AER streams are to be recorded simultaneously by multiple monitors, it can be very difficult if not impossible to ascertain after the fact what the precise timing offsets between the recorded streams might have been. In order that multiple monitors run with synchronized timers, there may be provision made in the monitor hardware to establish direct synchronization links between two or more monitors. One monitor is then designated the master and the remainder are slaves. The time-stamp timers of the slaves can then be clocked by the master. When the master timer is reset, all the slave timers are also reset and thus have the same absolute time-stamp as the master.

This approach to the synchronization of several monitors was implemented in the USBAER-mini2 (Berner et al. 2007; JAER Hardware Reference n.d.; Serrano-Gotarredona et al. 2005), see Section 13.2.3.

Time-base

The time-base used, that is, the period at which the time-stamp timer ‘ticks,’ also requires consideration. A time-base of 1 ms may be considered adequate given the time-scale of a biological action potential, but this depends on the application, and 1 μ s or less might be preferred. Some monitor hardware offers the user a choice of time-base, for example, the Rome PCI-AER board (Dante 2004; Dante et al. 2005).

Wrap-around

Obviously, the faster the clock, the more bits are needed to store the time-stamps before they wrap-around from the maximum storable value back to zero. For example, with a time-base of 1 μ s and 16-bit time-stamps, wrap-around from 65 535 to zero would happen every 65.536 ms leading to the identical time-stamp value representing multiple points in time on a period of 65.536 ms. There are several possible approaches for dealing with this wrap-around. In some systems and circumstances, for instance when data are simply being captured over short time periods for immediate display, it may be possible to simply do nothing. Alternatively, increasing the number of bits used for the time-stamp clearly reduces the frequency of wrap-around. A 32-bit time-stamp used with a time-base of 1 μ s wraps around only roughly every hour and 11.5 minutes, but comes at the expense of having to store and transmit twice as many bits. Running with a slower time-base is of course also possible, but results in a loss of timing resolution for the individual events.

Another approach that has been used is generally to store 16-bit time-stamps, but to reserve a bit pattern to indicate that, exceptionally, a 32-bit time-stamp follows.

The USBAERmini2 described by Berner et al. (2007) takes the approach of retaining 16-bit time-stamps, but places a special *time-stamp-wrapped* event in the stream of data read by the host whenever the time-stamp time wraps around. The host can then reconstruct 32-bit

time-stamps from the 16-bit time-stamps present in the data together with the number of time-stamp-wrapped events that it has seen.

Heartbeats

The device described by Merolla et al. (2005) does not use time-stamps but instead uses a special ‘heartbeat’ address sent at regular intervals. This scheme uses less bandwidth but requires interpolation to reconstruct time-stamps.

Early Packets

One of the main challenges with monitoring is coping with high data rates, but low data rates can also be problematic. Consider for instance a monitor connected to a silicon retina of the kind described in Chapter 3 which only produces events when something in the observed scene changes. If there is very little change in the scene, then very few events are emitted and the monitor FIFO fills up only very slowly. An application performing visualization of the data or using it for real-time control would normally have no visibility of the data accumulating in the FIFO until the amount of data exceeds some threshold that causes it to be read by the host. To avoid this problem, a monitor can implement an ‘early packet’ feature, as was done by Berner et al. (2007), which ensures a certain maximum interval (e.g., 10 ms) between the times at which the data are available to the host. An additional early packet timer is used to force the FIFO to be read even if it is not full whenever the early packet interval has elapsed with no intervening reads.

Sniffers vs. Interposed Monitors

Suppose we have an AER transmitter chip connected to an AER receiver chip, and we want to monitor the communication between the two. In principle, there are two possibilities as shown in Figure 13.1: connecting an AER *sniffer* element to the bus (Figure 13.1a) or interposing a new AER element between the transmitter and the receiver (Figure 13.1b).

A sniffer element will capture the addresses of the events on an AER bus without being involved in the handshaking taking part on the bus, except insofar as for each request that appears on the AER bus, it stores the corresponding address, together with a time-stamp, in memory. (This is how a logic analyzer may be used to monitor an AER bus). The problems

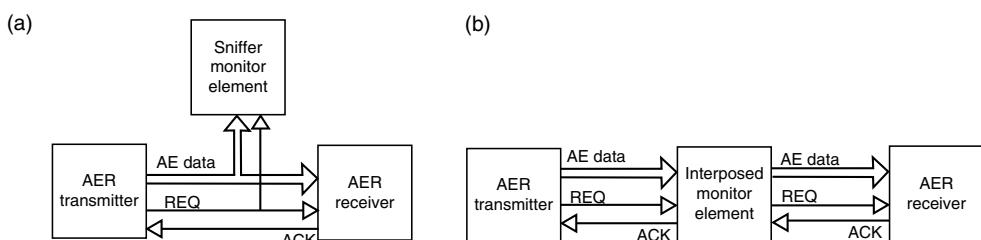


Figure 13.1 A monitor implemented as a sniffer (a) as opposed to an interposed normal AER sender/receiver element (b)

with this approach are that the speed of the bus protocol lines (e.g., 15 ns per event) could be faster than the maximum speed supported by the sniffer, causing events to be lost, or that the throughput of the AER bus might be so high that the contents of the buffers on the interface cannot be transferred to the computer's main memory in time. This also implies that events are lost.

The other possibility is to interpose a new AER element between the two chips. In this case the transmitter sends the event to the AER element and the AER element sends the same event to the receiver chip. The problem now is that the new AER element will always introduce extra delay, and may also block the transmitter if it is not able to keep up with its throughput therefore causing ISIs not to be conserved. But the behavior will be the same as if we had connected the transmitter to a slower receiver.

AER to Frame Conversion

Although much interesting processing can be carried out directly on address event streams (see Chapter 15) it is still sometimes desirable to convert AER data into a frame-based representation, that is one in which 'pixels' in regularly generated frames reflect the frequency of occurrence of particular AEs, analogous to a 2D histogram. Some monitors implement AER to frame conversion in hardware (Paz-Vicente et al. 2006). This is a relatively simple task as it basically only requires counting the events received for each pixel address over the frame period. Under certain circumstances, with high activity, this technique may reduce the bandwidth required to represent the data. Of course, experimenters will still sometimes want to know the precise timing of each event, thus both alternatives should be preserved.

13.1.2 Sequencing AER Events

By sequencing AER events, we mean 'playing' a stream of AEs out onto an AER bus. A sequencer therefore allows events from a host PC to be sent out to the attached AER receiver. These events may for example represent a pre-computed, buffered stimulus pattern, or they might come from a previously monitored and stored AE stream, but they might also be the result of a real-time computation. In order for the individual AEs to be 'played' out onto an AER bus at the correct moments, the AE data delivered to the sequencer needs to include some form of timing information, typically in the presence of words defining the ISIs between the address words. These addresses and ISIs are then buffered in a FIFO with the sequencer simply waiting the indicated length of time, when it reads an ISI word and outputting the AE, when it reads an AE word. (Note that as described for monitoring in Section 13.1.1 these 'ISIs' are not true ISIs in the biological sense because the interval to which they refer is an interval between the transmission of any two addresses which represent spikes on the same bus, and not usually between two identical addresses representing the spikes of the same neuron.)

While monitors usually work with absolute time-stamps, sequencers often use relative 'ISI' values (as described in Section 13.1.1 under the heading of Inter-Spike Intervals and Time- Stamps) to determine the timing of the address-events they emit. In this case, the output of a monitor is not suitable to be the direct input to a sequencer. Given that relative ISIs are used, there is no concern with the time-stamp wrapping round, but it may not be possible to represent a required delay within the number of bits provided to represent the ISI. Therefore a

sequencer operating with relative ISIs should be prepared to accept and honor delays specified by a consecutive sequence of multiple ISI delay words.

Choice of time-base is also of relevance for sequencing, and the same arguments apply as for monitoring (c.f. Section 13.1.1). Sequencer functionality, like monitor functionality, is also usually implemented using an FPGA. And as with monitors, it makes sense to construct sequencers that are able to output AE words which have a width of 2^n 8-bit bytes.

Discounted Delay

If the acknowledge from the receiver is delayed, the delay can be subtracted from the time to wait before transmitting the next event. If the result of the subtraction is negative, there is no wait before transmitting the next event. With this treatment, described in Paz-Vicente et al. (2006) as a ‘discounted’ delay, the delay between events is no longer strictly relative to the earlier one, and a limited delay in receiving an ACK for one event will not propagate to cause a delay of all successive events.

First In, First Out

Analogous to monitoring, but operating in the opposite direction, a FIFO is necessary to decouple the buffer-wise operations of the host writing the events into the sequencer from the asynchronous carefully timed transmission of these events onto the bus. In the ideal case, the FIFO will be written to often enough that it never empties, or underruns. If it does so, the interval between the last spike to be emitted before the underrun and the next spike will be much longer than intended, with potentially bad consequences on the neuromorphic computation being performed by the receiving system. A mechanism is usually provided to alert the user if and when such an underrun does occur. To avoid underruns, data must always be ready to write to the FIFO whenever it signals, for example with a half-empty interrupt, that more data should be supplied.

Frame to AER Conversion

Sometimes it is desirable to produce AER output from a sequencer based on a frame-based image representation; the conversion from frames to AER can be delegated to the sequencer hardware. Particularly if the same frame is to be ‘presented’ to the receiver attached to the sequencer for some time, this may off-load the host CPU and the bus to the sequencer, since the frame need only be handed off to the sequencer once.

Implementing AER to frame conversion is a relatively simple task. Producing AER from a frame representation is not quite so trivial and several conversion methods have been proposed (Linares-Barranco et al. 2006). The number of events for a specific pixel should depend on its associated gray level and those events should be evenly distributed in time. The normalized mean time from the theoretical pixel timing to the pixel timing resulting from the various methods is an important comparison criterion. In Linares-Barranco et al. (2006) it is shown that, in most circumstances, the behavior of the various methods is similar and, thus, hardware implementation complexity is an important selection criterion. From the hardware implementation point of view, random, exhaustive, and uniform methods are especially attractive. Frame to AER conversion was implemented in, for example, the CAVIAR USB-AER board (Paz et al. 2005).

13.1.3 Mapping AER Events

Mapping performs address translation between AE source and destination address spaces, or the fan-out of single input events from AE sources analogous to axons to multiple destination addresses analogous to synapses. To perform the former function, a mapper operates in a one-to-one mode. The latter is a one-to-many mode. In either case the architecture of a mapper remains the same. There is usually a FIFO, an FPGA, and a block of RAM which holds a look-up table. Incoming AEs are first buffered in the FIFO such that they are not lost while the mapper is processing preceding events. When the FPGA reads an AE from the FIFO, it uses the AE as an address in the look-up table and reads from that address. If operating in a one-to-one mode, the data read on this first look-up operation determines the new outgoing AE. In one-to-many mode, there are two possibilities, fixed-length destination address lists and variable length destination address lists, see Figure 13.2.

Fixed vs. Variable Length Destination Lists

In the case of fixed length destination lists, for each source address there is a certain fixed amount of mapper memory available for the associated destination addresses, F_{\max} words. Of course it is still possible to arrange to map source addresses to fewer than F_{\max} words, but F_{\max} is the maximum fan-out of the mapper. If F_{\max} is small, this may severely limit the flexibility of the mapper in comparison with mappers using variable length address lists

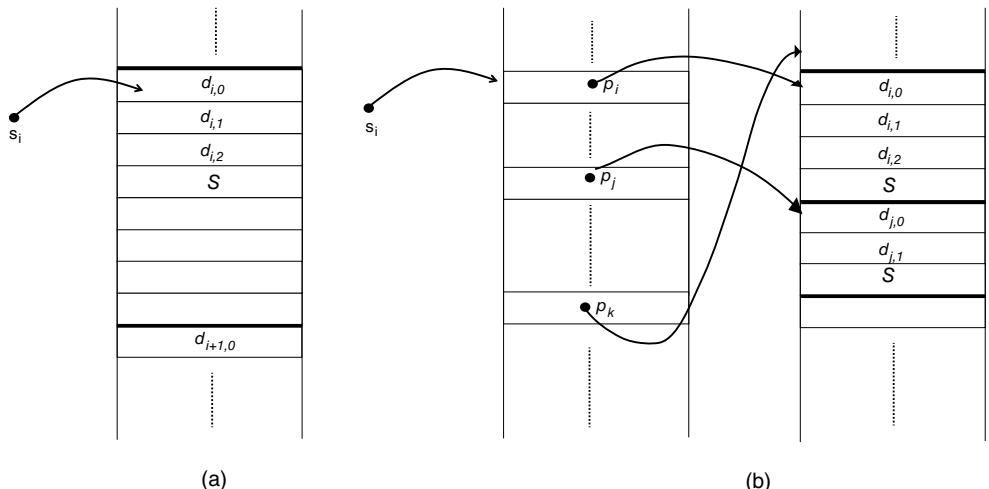


Figure 13.2 Fixed-length destination list mapping (a) versus variable length destination list mapping (b). In both cases the source address s_i is treated as a pointer into a look-up table (LUT). In (a) the destination addresses $d_{i,0}, d_{i,1}, d_{i,2}$ are found directly as a result of the first look-up. Each list of destination addresses is stored in a fixed length block, the length of which determines the maximum fan-out F_{\max} (here $F_{\max} = 8$). In (b), an extra level of indirection (pointer lookup) is needed through the pointer p_i to find the actual destination addresses. There need not be gaps between the blocks of destination addresses, and the blocks may be arbitrarily long and stored in any order as required for memory management purposes. In both (a) and (b), S is a sentinel value used to mark the end of a block (not required in the case of (a) in blocks in which all F_{\max} entries are used)

and may also waste considerable memory if the fan-out required for most source addresses is less than F_{\max} . However, fixed length destination list mappers are easier to construct and to program than variable length destination address list mappers: the address in the mapper memory of the block of destination addresses corresponding to a given source address can be trivially calculated from the source address, for example by a simple left shift if F_{\max} is chosen to be a power of two. Examples of fixed length destination list mappers can be found in Paz et al. (2005) and Fasnacht et al. (2008).

Variable length destination list mappers have no per-source address limit on fan-out other than that imposed by the total memory available. They are therefore more flexible than mappers using fixed length destination lists and can use the available memory more efficiently since destinations lists of various lengths can be packed contiguously in memory. However, this makes them a little more complex to construct and program, particularly if dynamic rewriting of the destination lists is required. A two-stage lookup is now required as a simple calculation of the address of the destination list is no longer possible. The first lookup of the source address now yields not a destination address directly, but rather a pointer back into the mapper memory to where the destination list can be found. Following these pointers requires more clock cycles than are required in fixed length destination list mappers. Furthermore, if it is required that the destination lists can be dynamically changed, for example, individual destination addresses added (analogous to the formation of new synapses) to facilitate some form of learning, while the mapper is in operation and thus without rewriting the whole mapper memory, some relatively sophisticated memory management routines are required to move the address lists around in the memory to make room for new entries. Variable length destination list mapping can be found in the SCX project (Deiss et al. 1999) and Rome PCI-AER board (Dante et al. 2005).

Sentinel Values or List Lengths

With variable length destination lists (and with fixed maximum length destination lists when the number of destinations n is less than the maximum) a so-called sentinel value S is required to terminate the list, which then takes the form d_1, d_2, \dots, d_n, S . The value S must be chosen so that it is not a member of the set of valid destination address. Equivalently formulated, the value chosen for S can never be used as a destination address. Commonly, a binary all ones or all zeros word is chosen meaning that, for example in a 16-bit system the address 65 535 or 0, respectively, cannot be used. Alternatively, the length of the list can be stored explicitly at its head, thus, n, d_1, d_2, \dots, d_n , in which case no sentinel value is need to be reserved.

Writing to the Mapping Table

Obviously there must be provision to write to the memory forming a mapper lookup table to set up the mapping table in the first place, and possibly to rewrite entries dynamically while the mapper is running. If the memory is attached to an FPGA that performs the mapping, this FPGA must also support writing to the memory from a host, for example, over a USB connection. An alternative would be the use of dual-ported RAM (DPRAM) where the host writes on one port and the mapping FPGA reads on the other, but DPRAM is relatively scarce and expensive and usually less easy to interface to than the more commonly used static RAM (SRAM). SRAMs have very fast access times and high bandwidth, and are typically used for cache memory.

If the mapping table needs to be written to on the fly while the mapper is running, care must be taken to ensure that the device (e.g., the FPGA) doing the mapping only ever sees consistent mapping tables, particularly in the case where a sentinel value is used to mark the end of the list of destination addresses, since a missed sentinel will typically cause the device to go on emitting whatever the LUT memory contains until it happens to encounter another sentinel value.

Algorithmic Mapping

Up to now we have discussed only LUT-based mappers. It is also possible to perform so-called algorithmic mapping in which the destination addresses emitted are generated by applying some algorithm to the source address. This algorithm can be as simple as adding an offset to the source address. More interestingly, one might form projective fields (Lehky and Sejnowski 1988) by specifying for instance that the destination addresses corresponding to source address s are given by $d_i = f(s) + i - \lceil \frac{w}{2} \rceil \forall i \in \{0, 1, \dots, w - 1\}$, where $f(s)$ is calculated by the mapper for each s as it arrives and w is the width of the projective field. A kind of hybrid between table look-up and algorithmic mapping to produce projective fields was proposed (Whatley 1997), though not finally implemented, for the SCX project (Deiss et al. 1999), in which the mapping was performed by a DSP rather than an FPGA.

Provision of Mapping Memory

Mapping, in contrast to monitoring and sequencing, can be carried out by a stand-alone device, that is in the absence of a host, once the mapping tables have been set up. To facilitate true stand-alone operation, the CAVIAR USB-AER board permitted its initial configuration, including mapping tables, to be loaded from nonvolatile storage in the form of an MMC/SD card as used in digital cameras (Paz-Vicente et al. 2006).

At the other extreme, in order to store mapping tables up to 2 GB in size in affordable, fast memory, Fasnacht and Indiveri (2011) took the approach of using a PC to provide the memory. Affordable FPGA systems do not have large amounts of RAM with high bandwidth and low access latency. On the other hand, in PC hardware, gigabytes of fast DRAM are readily available at prices which are orders of magnitude lower than memory with equivalent specifications on FPGA development platforms. Therefore, PC hardware and FPGAs are combined using the PCI interface. The actual mapper device is an FPGA on a PCI board plugged into a PC motherboard with 4 GB of RAM, of which 2 GB are reserved to store the mapping table, see Figure 13.6.

Repetition Counts, Probabilistic Mapping and Delays

The look-up table in an LUT-based mapper need not only hold destination addresses. It can also encode repetition counts if it is desired to emit some addresses multiple times for the same source address, transmission probabilities (Fasnacht et al. 2008; Paz-Vicente et al. 2008) to mimic synaptic release probabilities, and transmission delays (Linares-Barranco et al. 2009a) to mimic axonal conduction delays. Any or all of these properties can be incorporated by extending the words stored in the LUT, so that only some of the bits represent the destination address with further bitfields being added for the property or properties in question.

Transmission probabilities are implemented by drawing a number from a pseudo-random number generator (PRNG) for each destination address to be processed. If the number drawn

is greater than the probability value for the current destination address, then the address is discarded. Otherwise, the destination address is output in the usual way.

Transmission delays are implemented similarly to the way a sequencer handles inter-spike intervals (see Section 13.1.2). However, if different events require different delays, some events which result from a mapper look-up operation will need to be transmitted *before* events from previous mapper look-up operations. Thus the mapper output can no longer be handled as a simple FIFO queue, but must be kept sorted in order of desired transmission time.

13.2 Hardware Infrastructure Boards for Small Systems

13.2.1 Silicon Cortex

The first attempt to build a board that could handle multiple senders and receivers was the ‘Silicon Cortex’ (SCX) board (Deiss et al. 1999). This board implemented an AE communication infrastructure that could be used for interchip communication in simple neuromorphic systems. The SCX framework was designed to be a flexible prototyping system, providing re-programmable connectivity among on the order of 10^4 computational nodes spread across multiple chips on a single board, or more across multiple boards. The SCX was implemented as a VME-bus card, designed and fabricated by Applied Neurodynamics of Encinitas, California, USA. Each SCX board could support up to six chips or other AE sources, and multiple boards could be linked together in arbitrary topologies to form larger systems.

The SCX was devised to test and refine several fundamental problems encountered in building systems of analog chips that use the AE representation: coordinating the activity of multiple sender/receiver chips on a common bus; providing a method of building a distributed network of local buses sufficient to build an indefinitely large system; providing a software-programmable facility for translating AEs that enable the user to configure arbitrary connections between neurons; providing extensive digital interfacing opportunities; and providing ‘life-support’ for custom analog chips by maintaining volatile analog parameters or programming analog nonvolatile storage.

This board was an ambitious project at the time. There were two sockets to accommodate custom neuron chips and a daughterboard connector. Daughterboards could contain up to four elements that need to talk on the local address-event bus (LAEB), for example, four additional custom neuron chips, or an interface to peripheral sensory devices, such as retinas.

Communication among all of the chips in the system took place on three address-event buses (AEBs). The LAEB used for intra-board communication used a variant on the usual AER protocol as described in Section 2.4.2. Details of the LAEB protocol are described in Deiss (1994). The domain AEBs (DAEBs) used for inter-board communication used a fast synchronous broadcast protocol.

Communication between the chips on the SCX board (and any daughterboard) took place via the LAEB. The occurrence of an event on any chip was arbitrated within that chip, and led to a request from the chip to the bus arbiter. The bus arbiter determined which chip would have control of the LAEB at each cycle, and that chip broadcast an AE onto the bus. These events could be read by all chips attached to the bus. In particular, the bus was monitored by a DSP chip, which could route the AEs to a number of different destinations. For example, the DSP chip could use a look-up table to translate a source AE into many destination AEs.

Because of the bulkiness of the SCX solution and restrictions on the directly supported chip-pinouts, designers of multichip systems resorted to simpler independent boards for their needs.

13.2.2 Centralized Communication

The Rome PCI-AER board designed by Vittorio Dante of the Istituto Superiore di Sanità in Rome (Dante et al. 2005), attempts to provide a more flexible partitioning of functionality than the SCX board by separating the neuromorphic chips from the hardware infrastructure in a centralized hardware system. It consists of a 33 MHz, 32-bit, 5 V PCI bus add-in card and a header board which can be conveniently located on the bench-top and provides connectors for up to four AER senders and four AER receivers. Senders must use the so-called ‘SCX’ multisender AER protocol (Deiss 1994) described in Section 2.4.2 in which request and acknowledge signals are active low and the bus may only be driven while the acknowledge signal is active. Receivers may use either this ‘SCX’ protocol, or they may choose to use a point-to-point protocol (AER 1993) in which request and acknowledge are active high and the bus is driven while request is active.

The PCI-AER board performs the three functions executed by monitor, sequencer, and mapper blocks. The monitor can capture and time-stamp events coming from the attached AER senders via an arbiter and makes those events available to the PC for storage or further on-line processing. When an incoming AE is read by the monitor, a time-stamp is stored along with the address in a FIFO. This FIFO decouples the management of the incoming AEs from read operations on the PCI bus, the bandwidth of which must be shared with other peripherals in the PC such as the network card. Interrupts to the host PC can be generated when the FIFO becomes half-full and/or full, and in the ideal case, the driver will read time-stamped AEs from the monitor FIFO whenever the host CPU receives a FIFO half-full interrupt, at a rate sufficient that the FIFO never fills or overruns, given the rate of incoming AEs. If the CPU fails to empty the FIFO at a sufficient rate, the FIFO will fill up and a FIFO full interrupt will be generated. At this point, incoming events will be lost until such time as the CPU can once again read from the FIFO.

The sequencer allows events originated by the host PC to be sent out to the attached AER receivers. These events may, for example represent a pre-computed, buffered stimulus pattern, but they might also be the result of a real-time computation. Like the monitor, the sequencer is decoupled from the PCI bus using an 8 KWord FIFO. The host writes a sequence of words representing addresses and time delays to the sequencer FIFO. The sequencer then reads these words one at a time from the FIFO and either emits an AE or waits the indicated number of microseconds. FIFO half empty interrupts can be generated to signal the CPU to supply further data to the sequencer. If the CPU fails to supply data to the sequencer at a rate sufficient to prevent the sequencer FIFO becoming empty, this may indicate a failure of the system to generate the desired sequence of events with the desired timing. In this case a sequencer FIFO empty interrupt is raised to signal the underrun. The address events generated by the sequencer pass through the mapper and can therefore be transmitted on any of the four output channels.

The mapper implements programmable inter-chip synaptic connectivity. It maps incoming AEs from attached AER senders and/or the sequencer to one or more outgoing addresses for transmission to the attached AER receivers. It can operate in pass-through, one-to-one, or

one-to-many modes. It uses 2 MWords of on-board SRAM. The mapper too has a FIFO which decouples the asynchronous reception of the incoming AEs from the generation of outgoing AEs. Should this FIFO become full, it is possible that events will be lost, and this eventuality can be signaled to the CPU by means of an interrupt. The mapper, once it has been configured and the look-up table be filled with the required mappings, operates entirely independently of the PCI bus and host CPU, since all of the necessary operations, including table look-up, are performed by one of the FPGAs. A detailed hardware description of the hardware (the hardware User Manual) is available online (Dante 2004).

Other laboratories have chosen a solution targeted to their setup. One such system which also used a centralized communication block is the OR + IFAT system shown in Figure 13.8b. This system is described further in Section 13.3.1. The main module is the FPGA which controls two AER buses: one internal bus for events sent to and from the IFAT multineuron chip and one external bus for events sent to and from the octopus retina or a computer (CPU). Similar to many other AER systems of this size, the FPGA implements the mapping function needed to create the necessary receptive fields for implementing a network desired of the system.

The FPGA was also used to control the parameters of the neurons and synapses on chip, in particular, it was used to implement a different type of synapse for every virtual connection between neurons and to extend the range of synaptic weights through two dynamically controlled external parameters: the probability of sending an event and the number of post-synaptic events sent for every pre-synaptic event (Vogelstein et al. 2007a). The FPGA also implements a one-to-many mapping function for divergent connectivity (Vogelstein et al. 2007b). It uses a flat address space, that is, every targeted and sending unit has a unique address. This mapping solution is used on many early two-chip systems and has the limitation that every spike event produced by the chips has to go through a single mapper on the FPGA.

13.2.3 Composable Architecture Solution

Another communication infrastructure solution is to use multiple standalone boards between components of a system. The boards can be programmed for various functionalities and composed into systems somewhat analogously to commands in the Unix shell pipe mechanism. This solution was used in the CAVIAR system (see Section 13.3.3). In this way, boards can be added as needed between two chips that need to communicate spikes between each other. These boards remove the need for a PC to host the coordination of the flow of spikes from multiple AER chips and the communication between chips can be done in a distributed manner.

Various boards with different specifications were constructed as part of the CAVIAR project. Each of these boards supports one or more of the basic functionalities like mapping and monitoring. Because of the necessity to implement a hierarchical multilayered system modeled after the visual cortex, simple boards were also developed to merge spike streams from multiple chips onto one chip and to split streams from a single chip to multiple chips. In this scheme, local mappers can be implemented on the standalone boards, so that local networks can be established. The address space in this routing scheme is local and the input AEs are broadcast only to the target modules defined by the mapper on the board connecting the modules.

The five boards developed in CAVIAR had specifications which varied in the amount of functionality, the I/O port of choice, the port bandwidth, and the compactness of the board.

The first board was a PCI-AER interfacing PCB capable of sequencing timed AER events out from a computer or, vice versa, capturing and time-stamping events from an AER bus and storing them in computer memory. The second board was a USB-AER board. The third board was the Switch-AER PCB based on a simple CPLD. It splits one AER bus into two, three or four buses, or vice versa, merged two, three or four buses into a single bus. The fourth board, the mini-USB-AER board, had lower performance but used a more compact bus-powered USB interface and was particularly useful for portable demonstrations. The last board, the USBAERmini2 board is a bus-powered USB monitor-sequencer board.

The CAVIAR PCI-AER (Paz-Vicente et al. 2006) and the USBAERmini2 (Berner et al. 2007) boards can perform AE sequencing and monitoring functions but have no hardware mapping capabilities. Although host-based software-driven mapping is feasible, when it is necessary to build AER systems that can operate without requiring a computer, a specific device for this purpose is needed.

CAVIAR PCI-AER²

The CAVIAR PCI-AER tool (Paz-Vicente et al. 2006) is a PCI interface board that allows not only reading an AER stream into a computer memory and displaying it on screen in real-time, but also the opposite; from images available in the computer's memory, it can generate a synthetic AER stream in a similar manner that a dedicated VLSI-AER transmitter chip would (Boahen 1998; Mahowald 1992; Sivilotti 1991).

PCI Interface Design Considerations

Before the development of the CAVIAR PCI-AER interface, the only available PCI-AER interface board was that described in Dante et al. (2005), see Section 13.2.2. This board encompasses all the requirements mentioned for AER sequencing, mapping, and monitoring. However its performance is limited to approximately 1 Meps. In realistic experiments software overheads may reduce this value even further. In many cases these values are acceptable but many AE chips can produce (or accept) much higher spike rates.

When the development of the CAVIAR PCI-AER board was started, using 64-bit, 66 MHz PCI seemed to be an interesting alternative as computers with this type of bus were popular in the server market. When implementation decisions had to be made however, the situation had already altered significantly. Machines with extended PCI buses had almost disappeared and, on the other hand, serial LVDS-based PCI express (PCIe) was emerging clearly as the future standard, but almost no commercial implementations were on the market. Therefore, the most feasible solution at the time was to stay with the common PCI implementation (32-bit, 33 MHz).

The previously available Rome PCI-AER board uses polled and/or interrupt driven I/O to transfer data to and from the board, but does not do DMA or bus mastering. This is the main factor limiting its performance. The design and implementation of the CAVIAR PCI-AER board was developed to include bus mastering, and hardware-based frame to AER conversion (Paz-Vicente et al. 2006). The CAVIAR PCI-AER board can perform sequencing in parallel with monitoring.

² The text in this section is © 2006 IEEE. Reprinted, with permission, from Paz-Vicente et al. (2006).

The physical implementation is in VHDL for an FPGA. Most of the functionality demanded by the users could be supported by the larger devices in the relatively inexpensive SPARTAN-II family. The connection to the PCI bus is via a VHDL bridge (Paz 2003) that manages the protocol of the PCI bus, decodes the accesses to the PCI base addresses, and supports bus mastering and interrupts. A Windows driver and an API that allows the bus mastering capability of the board to be exploited and a Matlab interface were developed.

CAVIAR USB-AER

The CAVIAR USB-AER board permitted stand-alone operation and has several functionalities including hardware mapping. This stand-alone operating mode requires that the user can load the FPGA configuration and the mapper RAM from some form of nonvolatile storage medium that can be easily modified by users. The MMC/SD cards used in digital cameras provide an attractive possibility. However, users can also load the board directly from a computer via USB.

Many AER researchers would like to demonstrate their systems using instruments that can be easily interfaced to a laptop computer. This requirement can also be supported via USB. Thus the board can be used also as a sequencer or a monitor. The CAVIAR USB-AER board used USB 1.1 and was therefore subject to the bandwidth limitations of full speed USB (12 Mbit/s). Nevertheless, thanks to its 2 MB of on-board SRAM, the board could be used as a data logger to store up to 512 K events (including time-stamps) in memory for later, off-line processing. Similarly it could also be used to replay a sequence of computationally generated events to an AER system at up to 10 Meps.

The CAVIAR USB-AER board has also been modified to implement probabilistic mappings (Paz-Vicente et al. 2008). The same board was also used to implement configurable delays (Linares-Barranco et al. 2009a) and also frame to AER conversion.

Interface Features and Architecture

The board can serve as different functional devices according to the VHDL code that is loaded into the FPGA either from an MMC/SD card or from the USB bus. These functionalities are: sequencing – both frame to AER conversion or sequencing pre-stored events with time-stamps from memory; monitoring – both conventional monitoring of a sequence of events with time-stamps and AER to frame conversion; and mapping – one-to-one, one-to-many, with optional output event repetition, probability, and programmable delay. The precise functionality implemented at any one time depends only on the VHDL code which is synthesized and loaded into the FPGA as firmware. A simple architecture is the common basis for all the mapper-related functionalities. This architecture is shown in Figure 13.3.

USBAERmini2³

The USBAERmini2 (Figure 13.4) is a high-speed USB 2.0 AER interface that allows simultaneous monitoring and sequencing of precisely timed AER data. This low-cost (<\$100), two

³ The text in this section is © 2007 IEEE. Reprinted, with permission, from Berner et al. (2007).

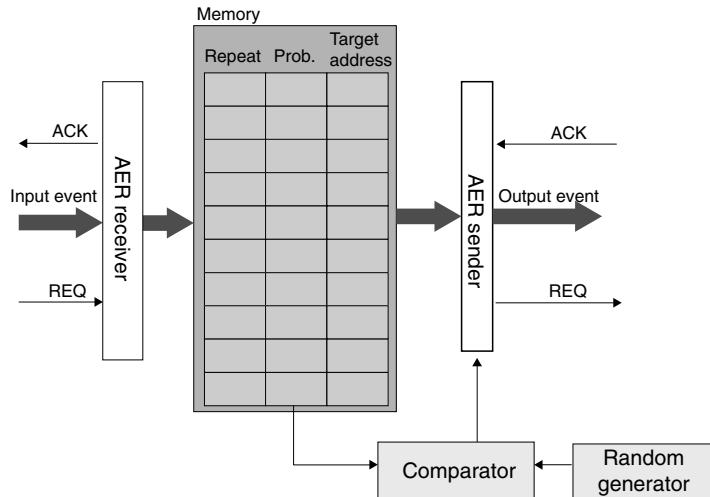


Figure 13.3 USB-AER mapper block diagram of base architecture. Adapted from Linares-Barranco et al. (2009b). With kind permission from Springer Science and Business Media.

chip, bus powered interface can achieve sustained AER event rates of 5 Meps. Several boards can be synchronized, allowing simultaneous synchronized capture from multiple devices. It has three AER ports, one for sequencing, one for monitoring, and one for passing through the monitored events.

None of a number of earlier generations of AER-computer interfaces (Dante et al. 2005; Deiss et al. 1999; Gomez-Rodriguez et al. 2006; Merolla et al. 2005; Paz-Vicente et al.

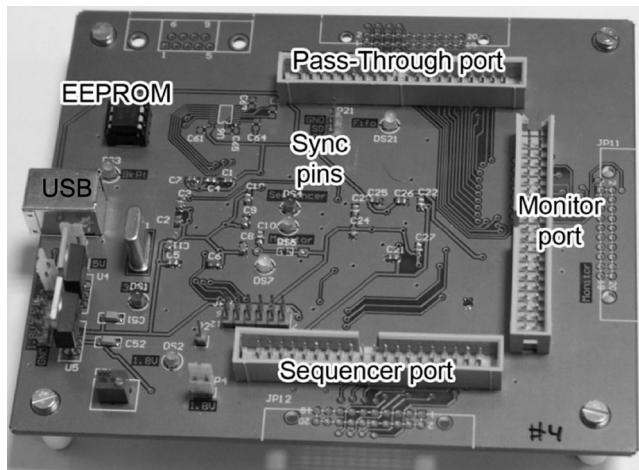


Figure 13.4 USBAERmini2 board. © 2009 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2009)

2006) had the convenience of high-speed, bus-powered USB combined with the synchronous monitoring of AER traffic. For the CAVIAR project (CAVIAR 2002; Serrano-Gotarredona et al. 2005), a heterogeneous mixture of AER chips was assembled into a visual system, and a device was required that could record from all parts of the system simultaneously, was simple to operate and use, was easy and cheap to build, and could be reused in other contexts. It was also desired to have a convenient device that could sequence recorded or synthesized events into AER chips to allow their characterization.

The functionality of the USBAERmini2 board includes sequencing and monitoring capability. The number of jumpers is minimized by the auto-detection of connected devices. A user connects a sending AER device to the monitor port and plugs the board into a USB port on a computer. They are then ready to capture and time-stamped AER traffic from the AER device.

The board uses two main components: a Cypress FX2LP USB 2.0 transceiver and a Xilinx Coolrunner 2 CPLD. The board was designed to be cheap to manufacture. Ten boards were hand-assembled by two people in three days. The components cost less than \$100 per board, including the 2-layer PCB.

The Cypress FX2LP is a USB 2.0 transceiver with an enhanced 8051 microcontroller and a flexible interface to its 4 KB of FIFO buffers, which are committed automatically to/from the USB by the Cypress serial interface engine (SIE). As in Merolla et al. (2005), the FX2LP is used in its slave FIFO mode, which means that the device handles all of the low level USB protocol in hardware. USB bulk transfers are used. Two FIFOs (one for each direction) are configured to be quad buffered and hold 128 events each. To the CPLD, the FX2LP appears as a FIFO source or sink of AER data.

The microcontroller part of the FX2LP controls the communication with the host computer. This involves the setup of the USB communication and the interpretation of commands that are received from the host to the control endpoint (such as *start capturing*, *stop capturing*, or *zero time-stamps*).

The CPLD handshakes with the AER sender and receiver, and records a 16-bit time-stamp whenever an event is received. Addresses and time-stamps are written to or read from the FIFOs in the FX2LP.

The CPLD uses a 16-bit counter for the generation of time-stamps. These time-stamps are unwrapped to 32 bits on the host computer. The time-stamp tick period can be controlled from the host computer. Two time-stamp modes can be used, either a tick of 1 μ s or a tick of one clock cycle, which is $33\frac{1}{3}$ ns. When the time-stamp counter overflows, a special *time-stamp-wrapped* event is sent to the host to tell it that the time-stamp wrap counter has to be incremented. Explicit time-stamps are sent for each event in order to preserve precise event timing information.

Synchronization

Electrical synchronization enables time-stamp-synchronized capture from multiple AER devices. A USBAERmini2 in ‘slave’ mode can be synchronized to a timing ‘master’, which generally is another USBAERmini2. The slave device’s time-stamp counter can then be clocked by the master. As soon as the synchronization state machine detects a time-stamp master at the synchronization input (i.e., the sync input goes high), it resets the time-stamp, signals to the host by sending a USB control transfer message, and changes to slave mode. The host resets its time-stamp wrap counter so that all slaves have the same absolute time-stamp as the master.

Performance

The peak limit on monitoring and sequencing performance is set by the number of clock cycles used by the state machines for a handshake cycle. The monitor state machine requires five clock cycles and the sequencer requires eight clock cycles. The CPLD clock frequency is 30 MHz, so the peak event rate for monitoring is 6 Meps and for sequencing 3.75 Meps.

The measured performance depends on the number and size of host memory buffers. For high frame-rate visualization at low event rate the host buffer size can be set to the USB transaction size of 512 bytes; for highest performance, the buffers need to be at least 4 KB. With a buffer size of 8 KB and four buffers, an event rate of 5 Meps was achieved when capturing events from one board, which is 83% of the limit defined by handshake timing. USB 2.0 high speed mode has a data rate of 480 Mbit/s. Because each event consists of four bytes, 5 Meps is equivalent to 160 Mbit/s. Monitoring with two or three interfaces simultaneously on one computer using the same USB host controller achieves a total rate of 6 Meps, that is 192 Mbit/s, about half the basic data bandwidth of USB 2.0.

The USBAERmini2 board was successfully used to monitor and log 40 K neurons spread over four different AER chips (Serrano-Gotarredona et al. 2005) and was in regular use in three different laboratories. The complete design of this board along with software (excluding device driver) has been open-sourced (jAER Hardware Reference n.d.).

SimpleMonitorUSBXPress

An even simpler, monitoring-only USB interface called the SimpleMonitorUSBXPress (Figure 13.5) was also developed during the CAVIAR project by Delbrück (2007) aiming at applications where simplicity, small size, low cost, and above all portability are important. Although it can only capture events at rates of around 50–100 keps, it is probably the world's smallest and most portable USB AER interface!

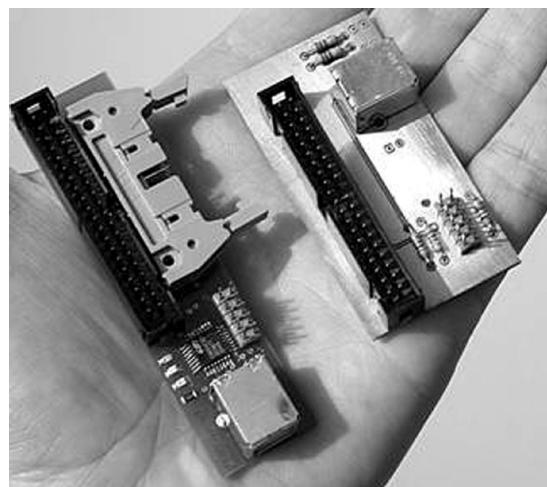


Figure 13.5 Two versions of the SimpleMonitorUSBXPress PCB. Taken from Delbrück (2007)

13.2.4 Daisy-Chain Architecture

The next architecture uses a daisy-chain scheme where each device communicates to only one other chip and events are communicated through point-to-point AER links instead of using a central router which connects a number of transmitters and receivers. This communication infrastructure was used on the ORISYS system which consisted of a retina and several Gabor chips programmed to detect different orientations (Choi et al. 2005; see also Section 13.3.2).

This daisy-chain architecture avoids the need for complex circuits that control bus access and perform routing. Instead, the ORISYS system uses two basic routing functions, the split and the merge, which are included on each Gabor chip and they determine the spike routing by the way the chips are linked together. Using this approach, routing circuit complexity expands automatically to accommodate the number of chips in the system. In addition, unique chip addresses are generated and updated automatically by the split and merge circuits, so that they can distinguish spikes from different chips in one AER address stream. Unlike the previous systems which used various boards to implement the splitter and merging circuits, it implements these circuits directly on-chip and includes an on-chip RAM for internally mapping the addresses.

The split circuit makes two copies of the AER events appearing at its input. One copy is sent into the neuron array through a decoder that reads the originating address of each spike and sends a spike to the neuron with the same row and column address, irrespective of the chip address. The other copy has its chip address incremented by one and is sent off chip via a transmitter. By daisy chaining the Gabor chips through the connection of the split output of one with the split input of the next, the designers can distribute the same silicon retina spike output to all chips.

The merge circuits combine the activity of all chips in the system, by daisy chaining the merge output of one chip with the merge input of the next chip. The merge output of each chip encodes all of the spike activity in the chips up to that point in the chain. Because it increments the chip address of input events, the merge circuit makes it possible to distinguish spikes originating in different chips.

13.2.5 Interfacing Boards using Serial AER

Newer interfacing boards substitute parallel AER links with faster serial links (based on, e.g., standardized SATA cables and connectors). Example PCBs which implement a serial communication protocol as described in Section 2.4.5 include the AEX board and the MeshGrid AER FPGA board. The serial SATA-like interfaces can be exploited by the 2.5 Gbps Rocket I/O interfaces available within the Spartan-6 FPGAs used on the MeshGrid FPGA board, for example.

The AEX Board⁴

The AEX board (Fasnacht et al. 2008) consists of three interface sections: a parallel AER section for connecting neuromorphic chips, a USB 2.0 interface for monitoring and sequencing on a PC, and a Serial AER (SAER) section to interface to other AEX boards or other boards with an SAER interface; and an FPGA used to route data between those interfaces.

⁴ The text in this section is © 2008 IEEE. Reprinted, with permission, from Fasnacht et al. (2008).

The serial AER approach employed here differs from previously proposed solutions (e.g., in Berge and Häfliger 2007) in several respects. Instead of using a high-end FPGA natively supporting serial IO standards, the design uses a low-cost Xilinx Spartan FPGA plus a dedicated *SerDes* chip, an approach which had been previously explored by Miró-Amarante et al. (2006). Such a *Serializer–Deserializer* locally receives data on a parallel bus and then sends it over a serial output at a multiple of the parallel interface speed and *vice versa* for the serial receive path. The usage of such a SerDes chip allows higher event rates to be obtained at significantly lower silicon cost. The FPGA and SerDes used cost about a third of the cost for the cheapest Xilinx Virtex-II Pro series FPGA necessary to implement a system such as that described in Berge and Häfliger (2007). Using the AEX board, event rates that are about three to four times faster than reported in Berge and Häfliger (2007) were achieved. In their approach the receiver simply drops events if it is not ready to receive them. On the AEX board, a flow control scheme is implemented that ensures that all events reach their destination. In the case that the receiver is currently unable to receive an event because it does not have the necessary receive buffer space available, it can tell the sender to stop until space is available. Finally, the FPGA package type chosen allows for in-house assembly and repair, whereas the ball grid array (BGA) package used by Berge and Häfliger (2007) would make this very difficult.

Flow Control

The flow control signal has to fulfill the following requirements: it has to be transmitted over a differential pair; for AC coupling it has to be DC free; and it has to represent two states, receiver busy or receiver ready. The flow control signal was chosen to be a square-wave because it is DC free and can easily be generated by clocked digital logic. The receiver FPGA signals that it is ready to receive by generating a square-wave at half the SerDes clock frequency. If the receiver is running out of FIFO space it signals the sender to stop by generating a square-wave at an eighth of the SerDes clock frequency. These signals can be easily decoded by the sender FPGA by counting the number of clock cycles the flow control signal keeps the same value. If this counter is one to three the sender keeps sending, if it counts to four or more the sender has to stop.

In order to know at what receiver FIFO fill-level to signal a stop condition to the sender, the sum of the forward and backward channel latencies must be known. The Texas Instruments SerDes TLK3101 used has a total link latency of 145 bit times (Texas Instruments 2008), giving 7.25 clock cycles, plus the line delay of the cable. The flow control back channel has a latency equal to the line delay plus 2 cycles for the synchronizer registers, plus 4 to 5 cycles to detect the stop state. This adds up to 14.25 cycles plus two line delays. At a 2 m maximum cable length this is $2 \times 2 \text{ m}/0.5c = 26.6 \text{ ns}$ which is 3.325 cycles (at 125 MHz). (This conservatively assumes the signal propagation speed in SATA cables to be half the speed of light.) Thus the total delay should be less than 18 cycles. The latest time to dispatch the flow control stop signal is thus when 18 words of the 16-bit receiver FIFO remain free.

PCI-Based High-Fanout AER Mapper⁵

AER mappers commonly use SRAM to store the mapping information and FPGAs to handle the I/O interfaces and to control the mapping process, typically having up to 4 MB of SRAM

⁵ The text in this section is © 2011 IEEE. Reprinted, with permission, from Fasnacht and Indiveri (2011).

to store the mapping tables, and using parallel AER interfaces. Fasnacht and Indiveri (2011) presented a new AER mapper that can store mapping tables up to 2 GB in size, and uses the same high-speed serial AER interfaces as the AEX board described above and in Fasnacht et al. (2008).

Mapper System Overview

The design choices were dominated by the requirements to interface the mapper to the pre-existing AEX boards (Fasnacht et al. 2008) that make use of the Serial AER interface and to affordably store and quickly access Look-up Table (LUT) type mapping tables of hundreds of megabytes in size.

As already noted in Section 13.1.3, affordable FPGA systems do not have large amounts of RAM. On the other hand, in PC hardware, gigabytes of very fast DRAM is readily available at prices which are orders of magnitude lower than memory with equivalent specifications on FPGA development platforms. For this reason, in this design PC hardware and FPGAs are combined using the PCI interface, see Figure 13.6.

The PC motherboard uses the Intel P35/ICH9R chip-set. Unlike in the more recent Intel and AMD CPUs, in this architecture the memory controller is not in the CPU, but in the PCI North Bridge. If a PCI device needs to access the main memory the datapath is:

PCI Bus \leftrightarrow North Bridge \leftrightarrow DRAM

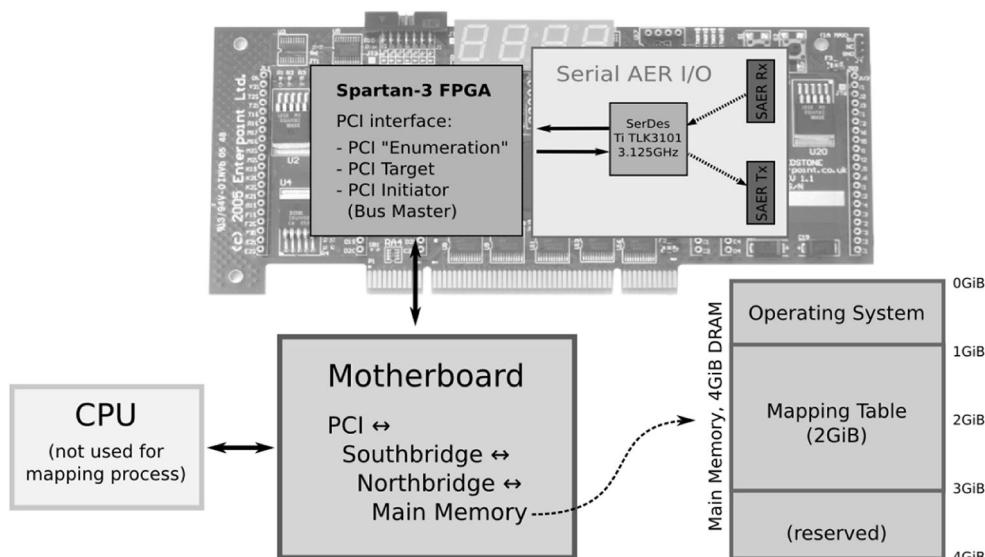


Figure 13.6 PCI-Based High-Fanout Mapper. The basis is a modified PCI FPGA development board containing a Xilinx Spartan-3 FPGA. The FPGA connects to both a custom daughterboard that implements the Serial AER interface and to the PCI bus of a PC Motherboard. The Mapping Table is held in the Main Memory (physical memory map at bottom right) on the motherboard and accessed via the PCI North Bridge. © 2011 IEEE. Reprinted, with permission, from Fasnacht and Indiveri (2011).

(see Shanley and Anderson 1999). This means that the CPU is not involved in the actual mapping operation. The 2 GB of mapping table space is split uniformly into 2^{16} pieces giving 32 KB of RAM per possible 16-bit input AE. With two bytes per AE this allows up to 16,384 words to be stored per input AE. A reserved sentinel address value is used to mark the end of a sequence of output AEs means that one input AE can generate up to 16, 383 output AEs; this is the maximum fanout of the mapper. This is an example of a fixed-length destination address list mapper, albeit one in which the destination lists can be very long. Only one contiguous access to the mapping table RAM is needed in order to get all the output AEs for one input AE. Calculating the memory address for a given input AE is simple; the numeric value of the input AE is multiplied by 32 KB, and the mapping table offset is added. For the FPGA this is almost free as it is simply a bit shift followed by an addition, which can be done in a single cycle operation.

The mapper uses a custom PCI implementation in order to get as much control over the PCI communication as possible. The implementation ignores Arbitration Latency and the master Latency Timer. This means that other PCI bus-masters can be prevented from gaining access to the PC bus for longer than usual. In the case of the PCI mapper this is a desired behavior, as ongoing mapping activity should not be suspended to grant bus access to other PCI devices.

The PCI-FPGA board is extended with a daughterboard carrying the components necessary for the AER input and output interfaces. These AER interfaces use the same SerDes chips and SATA connectors and cables as the AEX board presented in Fasnacht et al. (2008) and in Section 13.2.5 above. This interface allows AEs to be transferred at very high speeds between boards in a multichip, multiboard setup. Neuromorphic sensors and processing chips are connected to the mapper through this interface and the AEX board. The Serial AER interface achieves event rates of 156 MHz with 16-bit AEs or 78 MHz with 32-bit AEs.

Typical Usage in Multichip Setups

A practical multichip setup such as that in Neftci et al. (2010) using this mapper typically consists of a number of neuromorphic chips, each of which is directly connected to an AEX board and an AER mapper. These components are connected using their Serial AER interfaces in a ring topology. Each AEX board has a certain address space assigned. If incoming AEs fall into that space, they are sent to the local chip. Otherwise they are forwarded (kept in the ring). This allows the mapper to send AEs to all chips, and allows all chips to send AEs to the mapper. The full network connectivity is controlled at one single point, that is the mapper. Several examples of multichip systems have been built with this mapper, using up to five multineuron AER chips and in none of them was the mapper considered to be the limiting factor.

Performance

The measured latency of 774 ns for a fanout of 64 is the same as for a fanout of 1. The mapper takes 983.7 ns to map and send out 64 events and 15.27 μ s for 1024 events. The mapper actually emits two destination AEs per PCI bus-cycle (30 ns). This is to be expected, given that the AEs are 16-bit and the PCI bus transfers 32 bits per cycle. The bandwidth bottleneck is the maximum limit of the PCI bus.

13.2.6 Reconfigurable Mesh-Grid Architecture

In CAVIAR, as described in Section 13.2.3, a bandwidth-efficient interboard (or chip) interconnection scheme was used, as all interconnections were local to the sender and receiver boards (or chips). This maximizes communication efficiency, but at the cost of cumbersome reconfigurability. Whenever a change in interconnections is required, bus connectors need to be unplugged and plugged in elsewhere mechanically by hand.

One way to simplify multichip and multiboard systems' reconfigurability is to physically assemble them in fixed, hardwired, 2D-grid arrangements, while implementing a logical reconfigurable interconnection layer on top of the physical layer. This idea is used in the SpiNNaker system (described in Section 16.2.1) where the logical interconnection layer defines the interconnectivity among all neurons in the system. However, SpiNNaker's approach differs from the one developed by Zamarreño-Ramos et al. (2013), where the logical layer only describes the interconnectivity among AER modules (chips or PCBs).

Figure 13.7 (a) shows a 2D grid of modules, where each module can be any neuron/synapse AER block within a chip, an FPGA, or a PCB. Example AER modules can be any sensor chip, convolution module, WTA module, multineuron learning chip module, and so on. Each module is accompanied by a programmable router. The address events have two parts. The lower, less-significant part defines the 'local' information of the event, which is only meaningful to the sender and receiver AER modules, as it was in the CAVIAR approach; the upper, more-significant part of the address defines the (x, y) coordinate of the source or destination module within the grid, depending on whether source or destination coding is used. The routers look only at the upper part of the traveling event and send it to neighboring modules until the events reach their destination modules. The logical network is defined by the data programmed in the routing tables of the individual routers. With this approach, any network topology can be mapped onto the 2D grid of modules. This technique was tested on Virtex-6

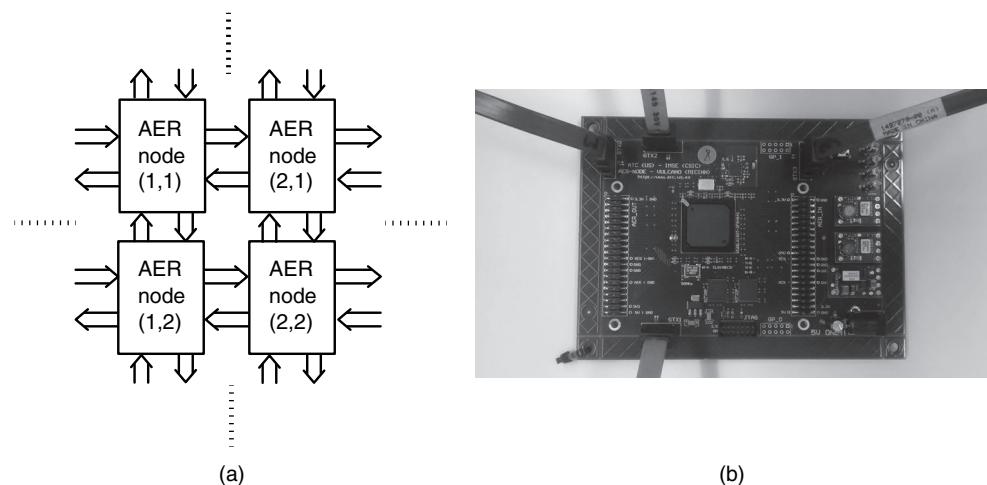


Figure 13.7 (a) A 2×2 grid arrangement of AER modules within an arbitrarily large reconfigurable router-based system. © 2013 IEEE. Reprinted, with permission, from Zamarreño-Ramos et al. (2013). (b) Spartan-6, four SATA serial-AER interconnect, multipurpose node board for 2D grid arrangements

FPGAs (Zamarreño-Ramos et al. 2013). Up to 48 AER 64×64 pixel convolution modules with routers could be fitted within one Virtex-6. Inter-module event hop time was below 200 ns, while inter-FPGA event hop time was about 500 ns. A new multipurpose MeshGrid AER FPGA board was developed as shown in Figure 13.7b. It contains a Spartan-6 FPGA, four SATA connectors for serial AER 2D-grid expansion, and a number of generic purpose pins.

This board can also be equipped with custom-made ASIC chips that use serial I/O interfaces (Zamarreño-Ramos et al. 2011, 2012). Serial I/O interfaces are readily available in many commercial FPGAs. These ASIC chips use an LVDS bit-serial intercommunication scheme where the sender and receiver are turned off during inter-event pauses, and turned back on very quickly when a new event needs to be transmitted. The reported maximum bit-serial intercommunication speed was 0.71 Gbps in a relatively old 0.35 μ m CMOS technology, while using Manchester encoding which reduces effective speed to half the physical speed. Here, there are two key types of circuit blocks: (a) A serializer-deserializer pair to convert back and forth from 32-bit parallel AER events to their serial counterparts, and using handshaking wires to signal when to turn the communication circuits off and back on; and (b) a driver/receiver pair that drives and reads a differential pair of 100Ω impedance microstrips. All blocks (serializer, deserializer, driver pad, and receiver pad) had a turn off and on time of less than 1-bit transmission time. Consequently, there is no penalty introduced by the capability of turning circuits off and back on. The blocks can communicate 32-bit events at peak rates of up to 62.5 Meps (Iakymchuk et al. 2014).

13.3 Medium-Scale Multichip Systems

This section describes three medium-scale systems (OR+IFAT, ORISYS, and CAVIAR) which illustrate the various roles played by the multineuron chips in these early multichip AER systems. These systems have the following modules in common: a front-end retina sensor and subsequent multineuron chips which process the retina events. The individual sensors in the systems, the Octopus Retina, the Parvo-Magno retina, and the Dynamic Vision Sensor, were discussed in Chapter 3. Each of the three systems also demonstrate the use of different solutions for the hardware infrastructure approaches discussed in Section 13.2.6. The present section also describes some of the applications demonstrated on these systems.

13.3.1 Octopus Retina + IFAT

The combined Octopus Retina (OR) plus integrate-and-fire transceiver (IFAT) system is one example of the two-chip AER vision systems developed by various groups during the late 1990s (Arias-Estrada et al. 1997; Higgins and Shams 2002; Indiveri et al. 1999; Ozalevli and Higgins 2005; Venier et al. 1997). Such systems were usually configured to demonstrate cortical-like responses, such as orientation selectivity and motion selectivity, using small numbers of neurons. These multichip systems were also focused on supporting modular, multilayered, hierarchical networks of brain or bio-inspired architectures such as convolutional networks and HMAX (Fukushima 1989; LeCun et al. 1998; Neubauer 1998; Riesenhuber and Poggio 2000). Systems such as the OR+IFAT system described here have also been used to demonstrate higher-level functions such as attention, feature coding, salience detection,

foveation, and simple object recognition. These functions capitalize on the AE domain to maintain arbitrary and reconfigurable connectivity between units in the multichip architecture.

System Description

The system, consisting of an 80×60 Octopus Retina (Section 3.5.4) and the IFAT board, is shown in Figure 13.8 (Vogelstein et al. 2007a). The IFAT board itself holds two multineuron chips, an FPGA, 128 MB of digital memory in a $4\text{ MB} \times 32\text{-bit}$ array, and an 8-bit digital-to-analog converter (DAC) required to operate the multineuron chips. The combined multineuron system is composed of 4800 random-access integrate-and-fire (I&F) neurons implemented on two custom aVLSI chips, each of which contains 2400 cells (Vogelstein et al. 2004, 2007a). All 4800 neurons are identical; each implements a conductance-like model of a general-purpose synapse using a switched-capacitor architecture (described in Chapter 8).

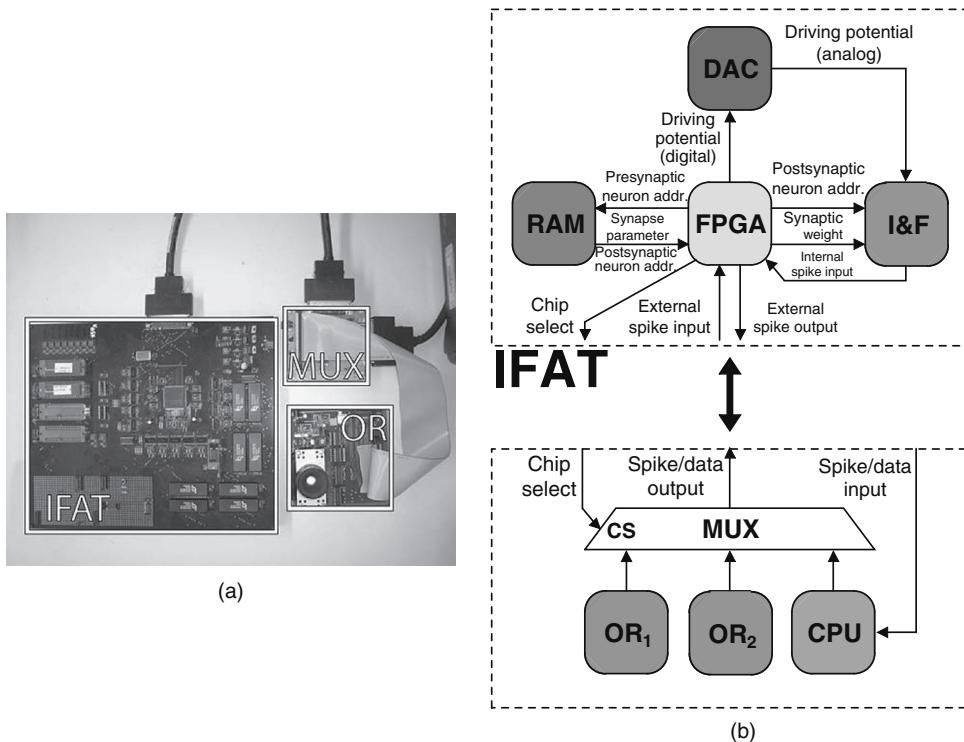


Figure 13.8 (a) Picture of the combined Octopus Retina (OR) + IFAT system. (b) Architecture of the system. The FPGA on the IFAT board receives spikes from either of the two Octopus Retinas (ORs) or the computer (CPU), depending on which is selected by the multiplexer (MUX). Outgoing addresses from the IFAT are sent to the computer for visualization or storage. Adapted from Vogelstein et al. (2007a). Reproduced with permission of MIT Press

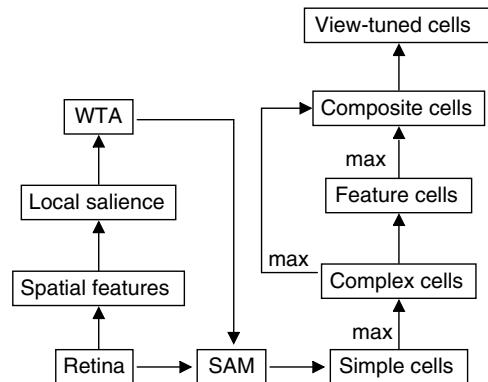


Figure 13.9 Hierarchical model of visual information processing based on work by Riesenhuber and Poggio (2000). Spatial features are extracted from retinal images using a small set of oriented spatial filters, whose outputs are combined to form estimates of local salience. The region with maximum salience is selected by a winner-take-all network and used to foveate the image by spatial acuity modulation. A large set of simple cells with many different preferred orientations is then used to process this bandwidth-limited signal. The simple cells' outputs are combined with a 'max' function to form spatially invariant complex cells, and the resulting data are combined in various ways to form feature cells, composite cells, and, finally, 'view-tuned cells' that selectively respond to a particular view of an object. Adapted from Vogelstein et al. (2007a). Reproduced with permission of MIT Press

Example Result – Spatial Feature Extraction

A target application of this system and other multichip vision AER systems is the ability of these systems to implement a hierarchical network of visual processing such as the HMAX network shown in Figure 13.9. To construct such a hierarchical system would require many more boards than the single IFAT system. The capability of this board to implement a spatial feature extraction module is described next.

Figure 13.10 illustrates how the IFAT system is used to perform spatial feature extraction by emulating eight different simple cell types. Simple cells are the first stage of processing in the visual cortex (Kandel et al. 2000). They act as oriented spatial filters that detect local changes in contrast, and their receptive fields (RFs) and preferred orientations are both functions of the input they receive from the retina.

Note that because the OR output is proportional to light intensity, these simple cells respond to intensity gradients, not to contrast gradients. In this example, each cortical cell integrates inputs from four pixels in the OR, two of which make excitatory synapses and two of which make inhibitory synapses. The excitatory and inhibitory synaptic weights are balanced so that there is no net response to uniform light. These receptive fields are generated by using the mapping function on the FPGA board.

Because each of the silicon cortex and retina chips only contain 4800 neurons, there is necessarily a trade-off between the spacing of similarly oriented simple cells throughout the visual field and the number of differently oriented simple cells with overlapping receptive fields. For the images in Figure 13.10, this trade-off was resolved in favor of increased

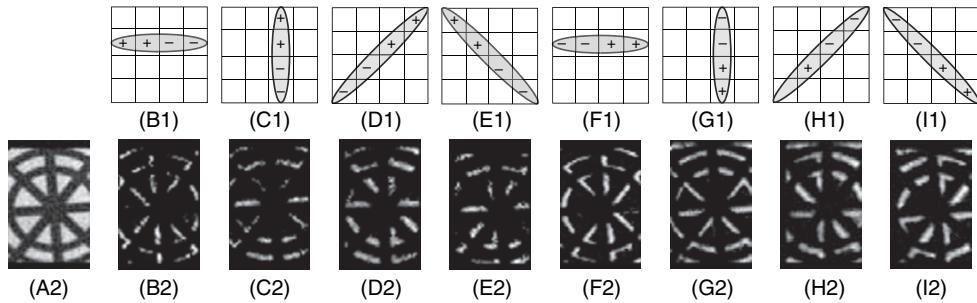


Figure 13.10 (B1)–(I1) Orientation-selective kernel compositions in the IFAT simple cell network. Each simple cell has a 4×1 receptive field and receives two excitatory (+) and two inhibitory (−) inputs from the silicon retina. (A2) Original image captured by silicon Octopus Retina. (B2)–(I2) Frames captured from real-time video sequences of retinal images processed by simple cell networks implemented on the IFAT system. Adapted from Vogelstein et al. (2007a). Reproduced with permission of MIT Press

resolution; each frame was captured from a different configuration of the system in which all 4800 simple cells had identical preferred orientations. By allowing lower resolution RFs, the system can be configured to simultaneously process two or four different orientations.

13.3.2 Multichip Orientation System

The second system is the multichip orientation system, ORISYS, developed in the laboratories of Shi and Boahen. It uses the 60×96 Parvo-Magno silicon retina by Zaghloul and Boahen (2004). The retina (also described in Chapter 3) generates spike outputs that mimic the responses of ON-sustained and OFF-sustained retinal ganglion cells on a 30×48 array of retinal positions. The events between the custom chips of the ORISYS system are communicated using the word-serial protocol (Boahen 2004a, 2004b) discussed in Section 2.4.4, that is, the y-address is first transmitted followed by the x-addresses of all active events on that row. The Gabor chip implemented neurons with spatial receptive fields similar to a Gabor function (Choi et al. 2004). These receptive field profiles have the same form as the Gabor function commonly used to model orientation selective cortical neurons (Daugman 1980; Jones and Palmer 1987), except that the modulating function is not Gaussian.

System Architecture

Figure 13.11 shows the block diagram of a feedforward architecture where individual Gabor chips are preprogrammed for four different orientations. The output of the retina is broadcast to several Gabor chips.

Each Gabor chip can process ON and OFF spikes from a 32×64 array of retinal positions (Choi et al. 2004). Every retinal position is handled by four neurons on the Gabor chip. Each of the four neurons computes a weighted sum of the spike rates from the ON and OFF ganglion

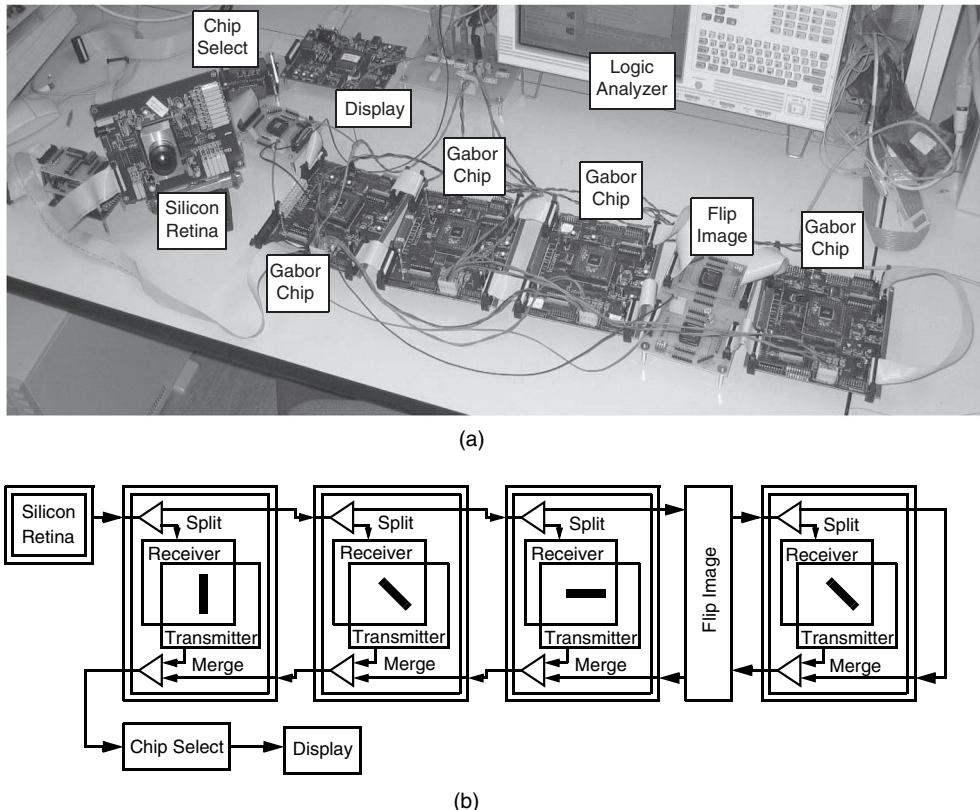


Figure 13.11 Feedforward implementation of ORISYS. Each box with the double line border represents a chip containing a retinotopic array of neurons. Gabor chips are represented by the larger boxes with the dark bar indicating the tuned orientation of the neurons on the chip. Boxes with single line borders indicate circuits that manipulate AER encoded spike trains. The ‘flip image’ circuit remaps spike addresses to flip the input and output images of the fourth chip horizontally, resulting in neurons tuned to 135°. The ‘Chip Select’ block passes only spikes originating from a desired chip (a) Photograph of setup (b) Block diagram. © 2005 IEEE. Reprinted, with permission, from Choi et al. (2005)

cells in a small neighborhood of that retinal position, half-wave rectifies it, and encodes the result as an output spike rate. The weighting function used in the sum is equivalent to the neuron’s receptive field (RF) profile. The four neurons are denoted by EVEN-ON, EVEN-OFF, ODD-ON, and ODD-OFF, and differ according to their RF symmetry (EVEN/ODD) and polarity (ON/OFF). The ON and OFF neurons encode the positive and negative half-wave rectified sums.

The neurons in this system capture many of the important characteristics of orientation-tuned cortical neurons. The model of linear filtering followed by a nonlinearity has been shown to account for the responses of a large proportion of V1 neurons in the visual cortex (Albrecht and Geisler 1991; Heeger 1992a, 1992b).

Example Result – Orientation Selectivity

Demonstrating orientation selectivity is a frequent application of AER systems which comprise both a retina and a multineuron module. Two approaches are frequently used in the configuration of an orientation-selective system. The first approach follows the ice-cube model (Hubel 1988; Hubel and Wiesel 1972) where each pixel extracts a different orientation (Cauwenberghs and Waskiewicz 1999; Liu et al. 2001). In the second approach, the neurons of each chip extract the same orientation. Therefore, multiple orientations require multiple chips. These two approaches trade-off orientation resolution and spatial resolution. The ORISYS system follows the second approach (Choi et al. 2004, 2005). It overcomes the limitation of the IFAT system in which only one orientation can be implemented on the multineuron chip at any one time and hence is not suitable for extracting multiple orientations from a scene in real time. The pixels on each of the multineuron chips directly implement a Gabor convolution function thus negating the necessity for creating such an orientation-selective receptive field for the neurons in the AE domain. By doing so, the AER bandwidth requirement is reduced.

The performance of this system in implementing orientation-selective RFs is illustrated in Figure 13.12 which shows the outputs of the neurons in response to a dark ring on a bright background using a feedforward system configuration. The four Gabor-type chips were tuned to similar spatial frequencies and bandwidths, but different orientations: 0°, 45°, 90°, and 135°. The 135° orientation was achieved by tuning the chip to 45° and then flipping the input and output images horizontally.

In total, this system contains 32 768 orientation-tuned neurons tuned to four orientations, two spatial phases, two polarities (ON/OFF), and 32×64 retinal positions. Neurons from different chips respond to parts of the ring, depending on where their tuned orientations match

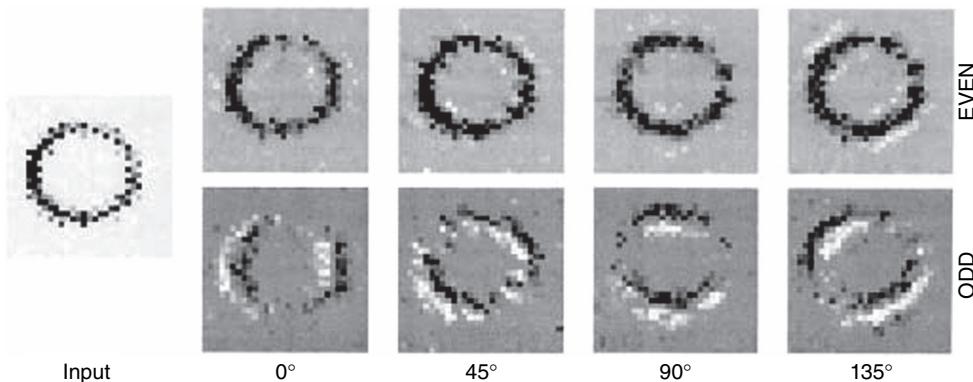


Figure 13.12 Responses of the feedforward ORISYS system to a black ring. The image on the left shows the output of the Parvo-Magno retina (Section 3.5.3). Each image position represents one retinal position in the upper-left 30×30 corner of the array. The image intensity encodes the difference between the spike rates of the ON and OFF neurons at each position. The eight remaining images show the outputs of the EVEN (top row) and ODD (bottom row) neurons on the Gabor chips. For the EVEN and ODD responses, black corresponds to differential spike rates ≤ -40 Hz and -80 Hz, respectively. White corresponds to differential spike rates $\geq +40$ Hz for both EVEN and ODD responses. © 2005 IEEE. Reprinted, with permission, from Choi et al. (2005)

that of the ring. Transistor mismatch adds variation in the neural responses across position, due to changes in the gain, tuning, and background firing rates of the neurons (Choi et al. 2004; Tsang and Shi 2004).

The total spike rate in the array is limited by the time which it takes to transmit one address event, $T_{\text{cyc}} = 357$ ns. Subsequent events encoded in the same burst require only $T_{\text{bst}} = 106$ ns. For low loads where each burst contains only one x-address, the link capacity is $1/T_{\text{cyc}} = 2.8$ million spikes per second. The link capacity in burst-mode is $1/T_{\text{bst}} = 9.4$ million spikes per second.

13.3.3 CAVIAR

The third system, called CAVIAR (Convolution AER VIision Architecture for Real-time), was the largest multichip multilayer AER real-time, frame-free vision system (Figure 13.13) built at the time, with a combined total of 45K neurons and 5M synapses (Serrano-Gotarredona et al. 2009). This system has four custom mixed-signal AER chips, five custom digital AER interface components and it performs up to 12 G synaptic operations per second. It is capable of achieving millisecond object recognition and tracking latencies, illustrating the computational efficiency of AER systems. The CAVIAR system is more complex than the previous systems in that there are three layers of processing including the convolution stage and the interfacing of the system output with a robotic motor output controlling a laser pointer.

System Architecture

The CAVIAR vision system (Figure 13.13) is composed of the following custom chips: the DVS temporal contrast retina chip, a set of programmable kernel convolution chips, a 2D winner-take-all (WTA) object chip, and a delay line chip and learning chip. It also includes

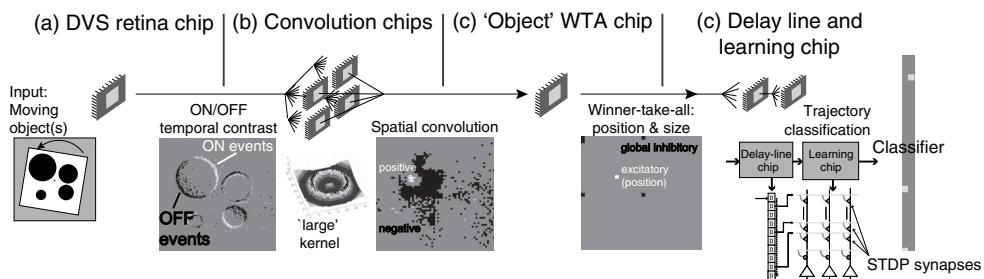


Figure 13.13 CAVIAR system overview. A bio-inspired system architecture performing feedforward sensing plus processing with the following conceptual structure: (a) a sensing layer, (b) a set of low-level processing layers usually implemented through projection fields (convolutions) for feature extraction and combination, and (c) a set of high level processing layers that operate on ‘abstractions’ and progressively compress information through, for example, dimension reduction, competition, and learning. An example output from each VLSI component is shown in response to the rotating stimulus. The basic functionalities of AER communication between chips are performed using custom AER boards (see Section 13.2.3). © 2009 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2009)

the set of AER remapping, splitting, and merging FPGA-based modules, and computer-AER interfacing FPGA modules for generating and/or capturing AER spikes discussed in Section 13.2.3. The hardware infrastructure was developed to support a set of AER modules (chips and interfaces) that are connected in series and in parallel to embody an abstract hierarchical multilayered architecture.

In an example object tracking application, moving objects in the field of view of the retina cause spikes. Each spike from the retina causes a splat of each convolution chip's kernel onto its own integrator array. When the integrator array pixels exceed positive or negative thresholds they in turn emit spikes (Serrano-Gotarredona et al. 2006). The resulting convolution spike outputs are noise-filtered by a winner-take-all (WTA) object chip (Oster et al. 2008). The WTA output spikes, whose addresses represent the location of the ‘best’ circular object, are fed into a configurable delay line chip that maps spikes in time into spikes over space. This spatial pattern of temporal delayed spikes are then learned by a competitive Hebbian learning chip (Häfliger 2007). The delay line chip takes temporal streams of spike inputs and projects them into a spatial dimension, thus allowing the competitive Hebbian learning chip to classify temporal patterns.

The WTA spikes can be used to control a mechanical or electronic tracking system that stabilizes the programmed object in the center of the field of view.

We discuss in more detail two of the multineuron chips (the convolution chip and the WTA object chip) in the CAVIAR system to illustrate the types of function that can be demonstrated on a multineuron chip.

AER Programmable Kernel 2D Convolution Chip

The neurons on the convolution chip perform spike-based convolution in a way that is different from other chips that implement on-chip convolution circuits such as the hard-wired elliptic (Venier et al. 1997), Gabor-shaped kernels (Choi et al. 2004), or x/y separable kernel filtering (Serrano-Gotarredona et al. 1999a).

The convolution kernel is stored in an on-chip RAM leading to two advantages: first, the shape or size of the convolution kernel is only restricted to the size of the on-chip RAM. Second, the external AER bus between the retina and the convolution chip is not used to generate the receptive field, therefore the bandwidth of the bus is not occupied by the mapping activity. The chip is an AER transceiver with an array of event integrators. For each incoming event, integrators within a projection field around the addressed pixel compute a weighted event integration. The weight of this integration is defined by the convolution kernel (Serrano-Gotarredona et al. 2006). This event-driven computation puts the kernel onto the integrators.

Figure 13.14 shows the block diagram of the convolution chip. The main parts of the chip are: (1) an array of 32×32 pixels. Each pixel contains a binary-weighted signed current source and an integrate-and-fire signed integrator (Serrano-Gotarredona et al. 2006). The current source is controlled by the kernel weight read from the RAM and stored in a dynamic register. (2) A 32×32 kernel RAM. Each kernel weight value is stored with signed 4-bit resolution. (3) A digital control block that handles the sequence of operations for each input event. (4) A monostable. For each incoming event, it generates a pulse of fixed duration that enables the integration simultaneously in all the pixels. (5) x-Neighborhood block. This block performs a displacement of the kernel in the x direction. (6) Arbitration and decoding circuitry that generates the output address events.

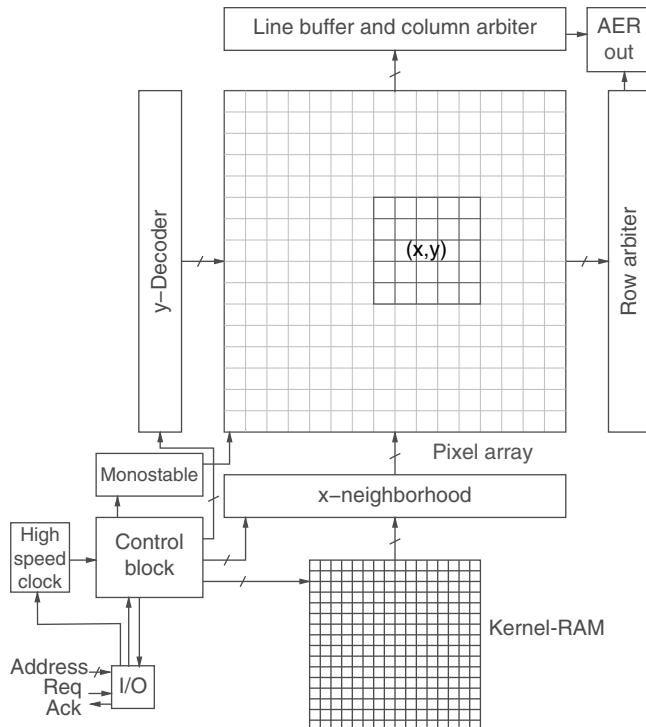


Figure 13.14 Architecture of CAVIAR convolution chip. © 2009 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2009)

The chip operation sequence is as follows: (1) Each time an input AE is received, the digital control block stores the (x, y) address and acknowledges reception of the event. (2) The control block computes the x -displacement that has to be applied to the kernel and the limits in the y addresses where the kernel has to be copied. (3) The control block copies the kernel from the kernel RAM row by row to the corresponding rows in the pixel array. (4) Once the kernel copy is finished, the control block activates the generation of a monostable pulse. In this way, in each pixel a current weighted by the corresponding kernel weight is integrated during a fixed time interval. Afterwards, kernel weights in the pixels are erased. (5) When the integrator voltage in a pixel reaches a threshold, that pixel asynchronously sends an event. The pixel integrates both positive and negative events and resets its voltage upon reception of an acknowledgement from the periphery.

The maximum input AER rate is 50 Meps and the maximum output rate is 25 Meps. The input event throughput depends on the kernel size and the internal clock frequency. The event cycle time is $(4 + 2n_k)T_{\text{clock}}$, where n_k is the number of programmed kernel lines (or row lines) and T_{clock} is the internal clock period. The maximum sustained input event throughput varies between 33 Meps for a one-line kernel to 3 Meps for a full 32-line kernel. AER events can be fed-in at a peak rate of up to 50 Meps.

AER 2D Winner-Take-All Chip

The AER ‘Object’ chip consists of a network of 16×16 VLSI integrate-and-fire neurons with excitatory and inhibitory synapses. It implements a winner-take-all operation which is a candidate for a cortical microcanonical function (Oster et al. 2008). By configuring the network connectivity for winner-take-all computation, the chip reduces the dimensionality of the input space by preserving the strongest input and suppressing all other inputs. The ‘Object’ chip receives outputs of four convolution chips (Figure 13.15) and computes the winner (strongest input) in two dimensions. It first determines the strongest input in each feature map, and then it determines the best feature map. The computation to determine the strongest input in each feature map is carried out using a two-dimensional winner-take-all circuit shown in one of the four central blocks in Figure 13.15. The network is configured so that it implements a hard winner-take-all, that is, only one neuron is active at a time. The activity of the winner is proportional to the winner’s input activity (Oster and Liu 2004). Each excitatory input spike charges the membrane of the post-synaptic neuron until one neuron in the array – the winner – reaches threshold and is reset. All other neurons are then inhibited via a global inhibitory

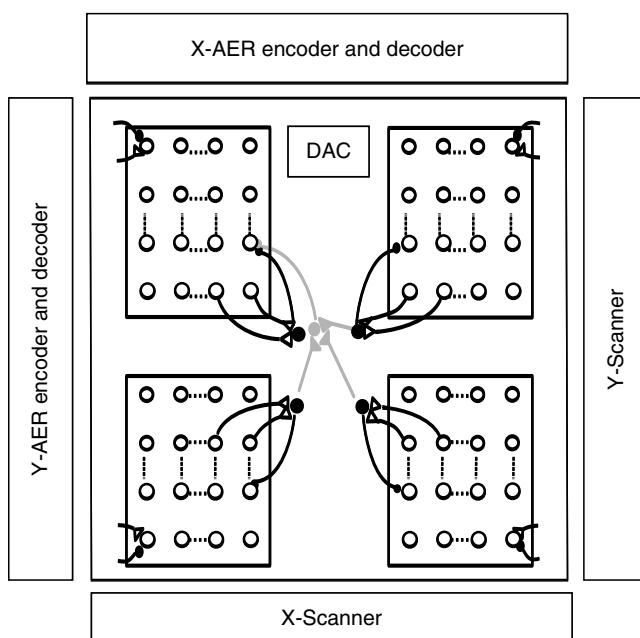


Figure 13.15 ‘Object’ chip with two levels of competition on four WTA networks. The digital-to-analog converter (DAC) sets the synaptic weights. The scanner blocks are used to read off the membrane potentials of the neurons. The AER encoder and decoder blocks are used to transmit the spikes off-chip and to receive spikes on-chip. Open circles are excitatory neurons, filled circles are inhibitory neurons. Curved lines between two neurons indicate a connection (real or virtual). Triangular terminations of these lines indicate excitatory synapses and circular terminations indicate inhibitory synapses. Light gray lines and circles indicate the elements, which carry out the second level of competition. © 2008 IEEE. Reprinted, with permission, from Oster et al. (2008)

neuron which is driven by all the excitatory neurons. Self-excitation provides hysteresis for the winning neuron by facilitating the selection of this neuron as the next winner.

Because of the moving stimulus, the network has to determine the winner using an estimate of the instantaneous input firing rates. The number of spikes that the neuron must integrate before eliciting an output spike can be adjusted by varying the weights of the input synapses.

To determine the winning feature map, the authors use the activity of the global inhibitory neuron (which reflects the activity of the strongest input within a feature map) of each feature map in a second layer of competition (see Figure 13.15). By adding a second global inhibitory neuron to each feature map and by driving this neuron from the outputs of the first global inhibitory neurons of all feature maps, only the strongest feature map will survive. The output spikes of the ‘Object’ chip encode both the spatial location of the stimulus and the identity of the winning feature.

Example Result – Tracking

CAVIAR’s capabilities were tested on a demonstration system (Figure 13.16) that could simultaneously track two objects of different size.

A mechanical rotor (1) holds a rotating white piece of paper with two circles of different radius and some distracting geometric figures. The vision system follows the two circles only, and discriminates between the two. A pair of servomotor driven mirrors (2) changes the point of view of the AER retina (3), which sends outputs to a monitor PCB (4), and a mapper PCB (5) before reaching the convolution PCB with four convolution chips (6). The latter PCB’s output is sent through another monitor PCB (7) and mapper PCB (8) to the 2D WTA ‘Object’ chip (9). This output is received by a monitor PCB (10) which sends a copy to a microcontroller (11) that controls the mirror motors to center the detected circle. Another copy of the WTA output

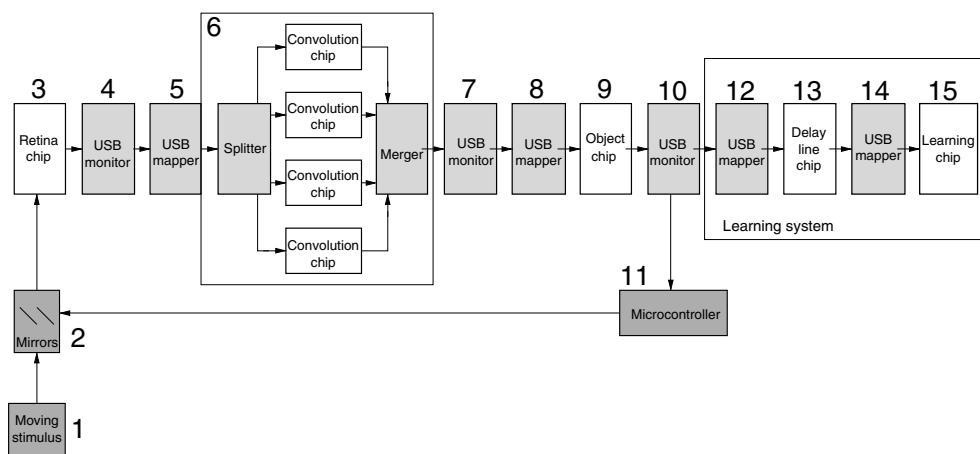


Figure 13.16 Experimental setup of multistage AER CAVIAR system for tracking a circle (white boxes include custom-designed chips, light gray boxes are interfacing PCBs described in Section 13.2.3, dark gray boxes are remaining modules). © 2009 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2009)

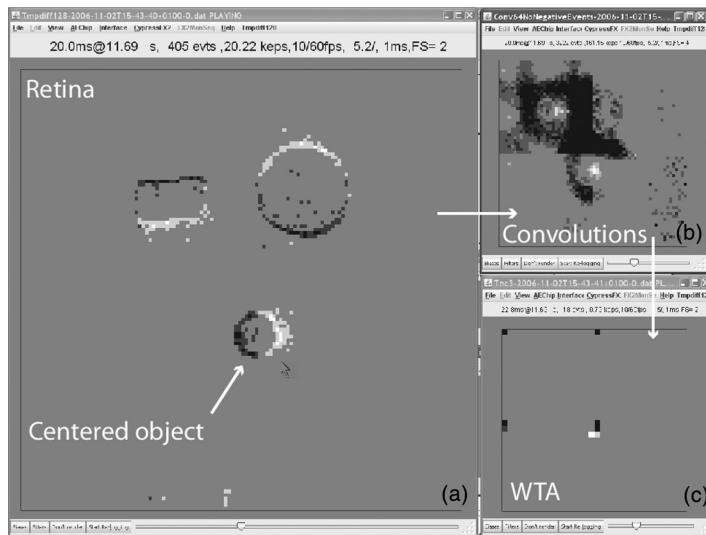


Figure 13.17 A 20-ms snapshot of outputs of the various custom chips of the CAVIAR tracking system. The retina central point of view is changed dynamically to follow the small circle, which is then always centered in the field of view. © 2009 IEEE. Reprinted, with permission, from Serrano-Gottaredona et al. (2009)

is sent to the learning system which consists of a mapper (12), a delay line chip (13), another mapper (14), and a learning classifier chip (15), and which learns to classify trajectories into different classes.

Figure 13.17(a) shows a histogram reconstructed from the DVS retina output captured by monitor PCB (4) in Figure 13.16. White dots represent positive sign events (dark-to-light transitions) and black dots represent negative sign events (light-to-dark transitions), allowing identification of the direction of motion of the geometric figures, which is clockwise in this case. Figure 13.17b also shows a histogram image reconstructed from the 64×64 convolution PCB output captured by monitor PCB (7) in Figure 13.16. In this case, the kernel was programmed to detect the small circle. Positive sign events (white) show where the small circle is centered, while the negative events (dark) show where it is not. The convolution output includes some noise, which is filtered out by the WTA operation. The convolution output pixels are transformed from size 64×64 to 32×32 (by grouping 2×2 pixels into one) by the mapper (8) in Figure 13.16. Figure 13.17c also shows the output of the WTA computing stage, where all noise has been filtered out. The white pixels show the center of mass of the small circle and the dark pixels show the activities of the local and global inhibitory units of each quadrant.

13.4 FPGAs

There is a very active community of neuromorphic engineers working with FPGA devices. These devices allow very much fast system design, debugging and testing workflows compared

to full-custom VLSI chip designs (both digital and analog). FPGA devices have continued to improve every year in features, performance, and resources since their invention by Ross H. Freeman (Freeman 1989), who, together with Bernard Vonderschmitt, co-founded the Xilinx company. The FPGA was conceived of as an evolution of the Complex Programmable Logic Device (CPLD). A CPLD is a set of logic blocks distributed in macrocells that can be connected programmatically using an interconnection matrix based mainly on multiplexors. The logic blocks available on a CPLD are very similar to those available on a Programmable Logic Device (PLD), a matrix of OR and AND gates that can be connected to obtain combinational digital circuits. Macrocells on CPLDs also include registers and polarity bits. The main characteristics of the evolution from CPLD to FPGA are the greater flexibility of the interconnectivity between logic cells, the inclusion of SRAM memory bits, and the inclusion of other embedded resources, such as multipliers, adders, clock multipliers, and so on. In the beginning, FPGAs could not work at very high clock speeds (but only at speeds on the order of units to tens of megahertz), but today one can buy a Virtex UltraScale FPGA that takes advantage of 3D Stacked Silicon Interconnect (SSI) technology, with up to 4.4 million logic cells, around 100 Mbits of RAM, 2800 digital signal processing blocks, up to 1500 IO pins, 100 Gbit Ethernet ports, and 16 Gbps and/or 32 Gbps low-voltage differential signaling (LVDS) serial interfaces for PCIe, SATA, or any other standard. Neuromorphic engineers have been using FPGA devices for supplementary tasks at the beginning and today for huge systems development on very powerful platforms.

For neuromorphic system communications support there have been several FPGA based solutions, for example the PCI-AER interface (see Section 13.2.2) developed by Dante (2004) with a performance of up to 1 Meps (Dante et al. 2005) used an FPGA for AER handshaking, time-stamp management, and event mapping. Under the CAVIAR project (2002–2006) a set of AER tools based on FPGAs was developed and distributed in the neuromorphic community by the Robotic and Technology of Computers Lab of the University of Seville (Gomez-Rodriguez et al. 2006; Serrano-Gotarredona et al. 2009), see Section 13.2.3 and Section 13.3.3. The Spartan II was the most used FPGA for sequencing, monitoring, and mapping events in real time using an on-board SRAM for stand-alone demonstrations or using USB when communicating and debugging from a personal computer or laptop. Fasnacht et al. (2008) developed the AEX board (described in Section 13.2.5) for communicating neuromorphic devices and to debug them using a host computer. This board uses a Spartan3 FPGA as a communication core and several commercial chips for interfacing to high-speed serial communications (Ethernet and USB2.0). Cauwenberghs' laboratory (at the Institute of Neural Computation, University of California) developed a Hierarchical AER (HiAER, see Section 16.2.2) communication routing architecture for neuromorphic systems using a Spartan6 FPGA to implement the communication at every node of the hierarchy. As part of this work, Park et al. (2012) presented a two-level communication hierarchy for connecting four IFAT neuromorphic chips. Under the FACETS project a neuromorphic wafer system was developed, where a set of synapse-and-neuron blocks (HICANN, see Section 16.2.4) are connected through an intra-wafer high-density routing grid. This grid can be connected to the outside world (another wafer or a host PC) using a packet-based protocol implemented on Virtex5 FPGA and specialized digital network chips (DNC). This communication infrastructure can manage up to 2.8 Geps traffic on the system (Hartmann et al. 2010).

In 2007, work incorporating neuromorphic processing using FPGAs started to appear in the literature, for example, Cassidy et al. (2007), who developed an array of Leaky-Integrate and Fire (LIF) neurons on FPGA and tested it by developing auditory Spatiotemporal Receptive

Fields (STRFs), a neural parameter optimizing algorithm, and an implementation of the Spike Time-Dependant Plasticity (STDP) learning rule.

Hasler's laboratory at the Georgia Institute of Technology, developed FPAAs, that is Field Programmable *Analog* Arrays, that could be used for implementing analog neuromorphic systems (Hall et al. 2005), and Petre et al. (2008) developed an automated way to program them using Simulink.

Later, several works implementing neuromorphic vision processing on FPGA-based systems began to be reported, for example, frame-based convolutional networks (Boser et al. 1992) were implemented on FPGAs under the NeuFlow system developed by Farabet et al. (2011), the architecture of which provides many processing elements that share a smart cache that accelerates the processing in a powerful pipeline. Orchard et al. (2013) developed an implementation of a biologically inspired spatiotemporal energy model for motion estimation in a Xilinx Virtex6 FPGA using frame-based visual information. Sabarad et al. (2012) developed a systolic array-based architecture which includes a run-time reconfigurable convolution engine which can perform multiple, variable-sized convolutions in parallel, synthesized on a Virtex6 platform. Al Maashri et al. (2011) developed a hardware architecture for accelerating the cortically-inspired, visual object classification algorithm, HMAX (Riesenhuber and Poggio 2000), on a four Virtex5 FPGA platform. Okuno and Yagi (2012) developed a real-time, frame-based visual processing system based on an adaptive image sensor with logarithmic compression that can be adjusted by control logic plus an embedded power-PC processor running on a VirtexIIpro FPGA connected to a silicon retina (Shimonomura et al. 2011). Spike-based convolutional processors, equivalent to those developed on VLSI chips (Serrano-Gotarredona et al. 1999b), for FPGAs (Linares-Barranco et al. 2009b), were lately combined in a mesh Network on Chip (NoC) of up to 64 convolution units on a Spartan-6 platform (Zamarreño-Ramos et al. 2013) that can be scaled by connecting several such platforms via 2.5 Gbps serial links (Iakymchuk et al. 2014).

Bio-inspired, spike-based neural network models have been synthesized successfully on FPGAs to improve accuracy and allow for large-scale neuromorphic algorithm implementations, for example Gomar and Ahmadi (2014), where a high-accuracy implementation of biological neural networks based on LIF, Adaptive Exponential Integrate and Fire model (AdEx) and Izhikevich neuron models were implemented and tested on a VirtexIIpro FPGA.

Even spike-based processing blocks for neuro-inspired motor control (Perez-Peña et al. 2013) have been developed for Spartan3 and Spartan6 FPGAs in such a way that a set of building blocks can be connected to develop custom spike-based processing systems.

13.5 Discussion

We have seen in the examples presented in Section 13.2, the same basic monitoring, sequencing, and mapping functionalities have been re-implemented many times over the years with increasing technological sophistication to achieve ever better performance and/or usability, with occasional extra features also being implemented (see Figure 13.18).

The SCX can be seen as a very ambitious, forward-looking prototype. It incorporated several advanced features, such as its domain buses, which would theoretically have allowed for the construction of large (at least $O(10^5)$ neuron) multiboard systems. However, the chips

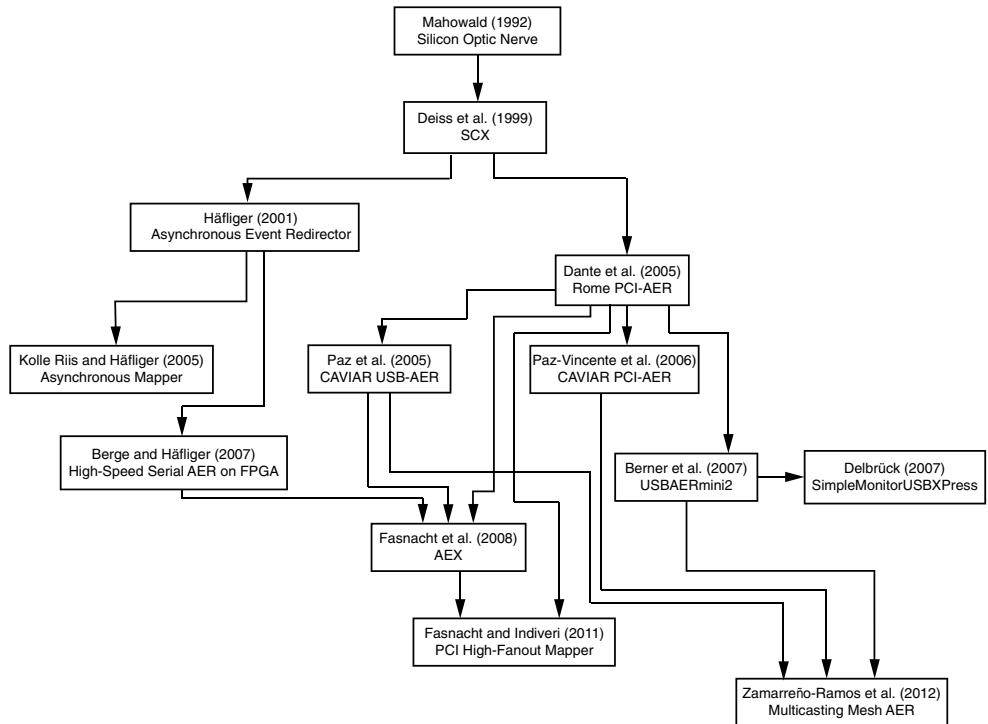


Figure 13.18 Hardware infrastructure time-line

of the day had orders of magnitude fewer neurons and no such large scale system was ever constructed using the SCX. The SCX design was such that its multineuron chips were plugged directly into the SCX board and as such very tightly coupled to one particular chip package type, pin-out, and chip parameter update mechanism. Therefore, despite the generality of the SCX's overall architecture, the SCX framework could not keep up with the progress in chip designs. The SCX as a VME bus based design was also very bulky, and it would never have been very convenient to provide one on the desk of each researcher who might want to use such a system.

The Rome PCI-AER board design (Section 13.2.2) kept the infrastructure components separate from the neuromorphic chips – the chips are expected to be mounted on their own carrier boards and connect to standard ribbon cable headers on a desktop header board which is in turn connected to the PCI bus based main board. With this decoupling from the chip design, the Rome PCI-AER board has achieved a useful life of almost a decade, despite the evolution of chip designs over this time. Also, being PCI-based means that it is much easier and cheaper to provide any and all interested researchers with a device than would ever have been the case with the VME based SCX. In the end, over 20 boards were produced and distributed in several countries. The Rome PCI-AER board was well supported with extensive user documentation, including for its Linux driver and accompanying library. However, because it was an attempt

at ‘one tool fits all’ approach, it has a large dimensional configuration space, which makes it rather difficult to use correctly, despite the documentation. Its largest failing though was its monitoring and sequencing performance. Due to its inability to perform DMA and bus mastering data transfers, the host CPU has to transfer monitor and sequencer data across the PCI bus, and only sustained event rates of around 1 Meps could be supported.

The CAVIAR project (Sections 13.2.3 and 13.3.3) introduced a flexible palette of specialized interface boards, including boards which can operate standalone for cases, where it is undesirable to have a desktop PC involved in the system. Where a connection to a PC is required, the CAVIAR project turned in part to the USB bus. Indeed it is much more convenient to be able to move a USB-based device from system to system – unlike PCI, the host computer does not need to be opened – and it becomes possible to use a laptop as a host. CAVIAR’s PCI-AER board learned the lesson from the Rome PCI-AER board and was DMA and bus-mastering capable to gain higher performance at the expense of greater development effort. The provision of multiple types of boards (one PCI-based and several USB-based variants) also meant that a much larger development effort was necessary than for the single Rome PCI-AER board. And yet, unlike for the Rome PCI-AER board (where the lowest level of user interaction was expected to be at the level of the software library API, or in extremis at the level of the driver itself), for some use-cases, the end-user of some of the CAVIAR boards was expected to achieve the promised flexibility by modifying the VHDL code defining the behavior of the FPGAs on the boards. In this way, the CAVIAR USB-AER board was modified to implement probabilistic mapping and mappings with configurable delays, features which other systems had not offered.

The USBAERmini2 introduced the important ‘early packet’ feature and the ability to synchronize the time-stamps across multiple boards, and was well supported and integrated into the jAER project. Because of this and its low cost and simplicity, it has also been continuously used for about a decade in other projects.

More recently, the Mesh-Grid architecture (Section 13.2.6), and the AEX board (Section 13.2.5) and its associated High-Fanout Mapper have moved toward establishing high-speed serial links over industry standard SATA cables as a standard for inter-chip communication, though communication of AEs to a host PC remains USB based. The High-Fanout mapper uses the PCI bus in a novel way. The host PC here is used more or less only to provide power and memory to a PCI device, which is implemented using a commercially available FPGA board with custom daughterboards providing this with SAER capability. This probably points the way to future developments for small scale AE hardware infrastructure systems. Such systems are likely to be increasingly constructed using as far as possible off-the-shelf modules (e.g., FPGA evaluation kits) together with the minimum possible of custom parts to actually interface to the neuromorphic chips. This approach helps to keep down development efforts and costs.

At the same time, particular systems such as retinas and cochleas have spawned their own intense AER infrastructure development efforts, which focus on tight integration, miniaturization, and specialization; for example, the latest silicon retina developments integrate the AER interface, multicamera synchronization, ADC, DAC, and inertial measurement units onto the same small PCB (Delbrück et al. 2014; jAER Hardware Reference n.d.).

Larger scale projects, however, face a different set of challenges, and these are discussed in Chapter 16.

References

- AER. 1993. The address-event representation communication protocol [sic], AER 0.02.
- Al Maashri A, DeBole M, Yu CL, Narayanan V, and Chakrabarti C. 2011. A hardware architecture for accelerating neuromorphic vision algorithms. *IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 355–360.
- Albrecht G and Geisler WS. 1991. Motion selectivity and the contrast response function of simple cells in the visual cortex. *Visual Neurosci.* **7**(6), 531–546.
- Arias-Estrada M, Poussart D, and Tremblay M. 1997. Motion vision sensor architecture with asynchronous self-signaling pixels. *Proc. 7th Intl. Work. Comp. Arch. for Machine Perception (CAMP)*, pp. 75–83.
- Berge HKO and Häfliger P. 2007. High-speed serial AER on FPGA. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 857–860.
- Berner R, Delbrück T, Civit-Balcells A, and Linares-Barranco A. 2007. A 5 Meps \$100 USB2.0 address-event monitor-sequencer interface. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2451–2454.
- Boahen KA. 1998. Communicating neuronal ensembles between neuromorphic chips. In: *Neuromorphic Systems Engineering* (ed. Lande TS) *The International Series in Engineering and Computer Science*, vol. 447. Springer. pp. 229–259.
- Boahen KA. 2004a. A burst-mode word-serial address-event link—I: transmitter design. *IEEE Trans. Circuits Syst. I, Reg. Papers* **51**(7), 1269–1280.
- Boahen KA. 2004b. A burst-mode word-serial address-event link—II: receiver design. *IEEE Trans. Circuits Syst. I, Reg. Papers* **51**(7), 1281–1291.
- Boser BE, Sackinger E, Bromley J, leCun Y, and Jackel LD. 1992. Hardware requirements for neural network pattern classifiers: a case study and implementation. *IEEE Micro* **12**(1), 32–40.
- Cassidy A, Denham S, Kanold P, and Andreou A. 2007. FPGA based silicon spiking neural array. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 75–78.
- Cauwenberghs G and Waskiewicz J. 1999. A focal-plane analog VLSI cellular implementation of the boundary contour system. *IEEE Trans. Circuits Syst. I* **46**(2), 1327–334.
- CAVIAR. 2002. CAVIAR project, <http://www.imse-cnm.csic.es/caviar/> (accessed August 5, 2014).
- Choi TYW, Shi BE, and Boahen K. 2004. An ON-OFF orientation selective address event representation image transceiver chip. *IEEE Trans. Circuits Syst. I* **51**(2), 342–352.
- Choi TYW, Merolla PA, Arthur JV, Boahen KA, and Shi BE. 2005. Neuromorphic implementation of orientation hypercolumns. *IEEE Trans. Circuits Syst. I* **52**(6), 1049–1060.
- Dante V. 2004. *PCI - AER Adapter Board User Manual*. 1.1 edn. Istituto Superiore di Sanità Rome, Italy. http://www.ini.uzh.ch/~amw/pcaer/user_manual.pdf (accessed August 5, 2014).
- Dante V, Del Giudice P, and Whatley AM. 2005. Hardware and software for interfacing to address-event based neuromorphic systems. *The Neuromorphic Engineer* **2**(1), 5–6.
- Daugman JG. 1980. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Res.* **20**(10), 847–856.
- Deiss SR. 1994. *Address-Event Asynchronous Local Broadcast Protocol*. 062894 2e edn. Applied Neurodynamics (ANdt). <http://appliedneuro.com/> (accessed August 5, 2014).
- Deiss SR, Douglas RJ, and Whatley AM. 1999. A pulse-coded communications infrastructure for neuromorphic systems [chapter 6]. In: *Pulsed Neural Networks* (eds. Maass W and Bishop CM). MIT Press, Cambridge, MA. pp. 157–178.
- Delbrück T. 2007. SimpleMonitorUSBXPress resources. http://www.ini.uzh.ch/~tobi/caviar/SimpleMonitor_USBXPress/index.php (accessed August 5, 2014).
- Delbrück T, Villanueva V, and Longinotti L. 2014. Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision. *Proc. 2014. Intl. Symp. Circuits Syst. (ISCAS 2014)*.
- Farabet C, Martini B, Corda B, Akselrod P, Culurciello E, and LeCun Y. 2011. NeuFlow: a runtime reconfigurable dataflow processor for vision. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 109–116.
- Fasnacht DB and Indiveri G. 2011. A PCI based high-fanout AER mapper with 2 GB RAM look-up table, 0.8 μ s latency and 66 MHz output event-rate. *Proc. IEEE 45th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6.
- Fasnacht DB, Whatley AM, and Indiveri G. 2008. A serial communication infrastructure for multi-chip address event systems *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 648–651.

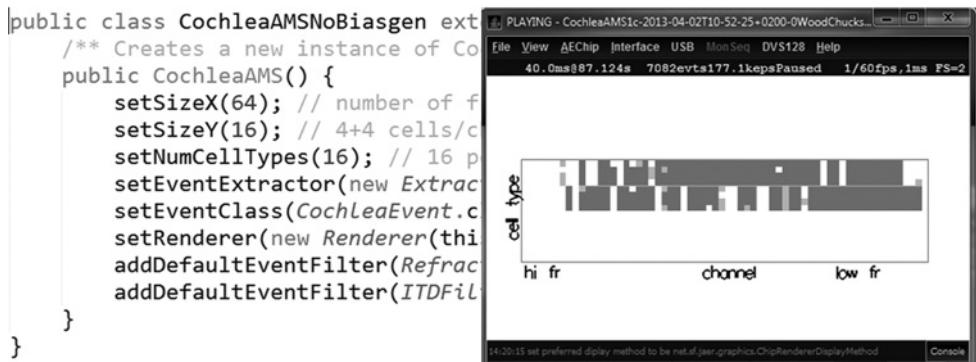
- Freeman RH. 1989. Configurable electrical circuit having configurable logic elements and configurable interconnects U.S. Patent No. US4870302 A.
- Fukushima K. 1989. Analysis of the process of visual pattern recognition by the neocognitron. *Neural Networks* **2**(6), 413–420.
- Gomar S and Ahmadi A. 2014. Digital multiplierless implementation of biological adaptive-exponential neuron model. *IEEE Trans. Circuits Syst. I: Regular Papers* **61**(4), 1206–1219.
- Gomez-Rodriguez F, Paz R, Linares-Barranco A, Rivas M, Miro L, Vicente S, Jimenez G, and Civit A. 2006. AER tools for communications and debugging. *Proc. IEEE Intl. Symp. Circuits Syst. (ISCAS)*, pp. 3253–3256.
- Häfliger P. 2001. Asynchronous event redirecting in bio-inspired communication. *Proc. 8th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)* **1**, 87–90.
- Häfliger P. 2007. Adaptive WTA with an analog VLSI neuromorphic learning chip. *IEEE Trans. Neural Netw.* **18**(2), 551–572.
- Hall TS, Twigg CM, Gray JD, Hasler P, and Anderson DV. 2005. Large-scale field-programmable analog arrays for analog signal processing. *IEEE Trans. Circuits Syst. I: Regular Papers* **52**(11), 2298–2307.
- Hartmann S, Schiefer S, Scholze S, Partzsch J, Mayr C, Henker S, and Schiffny R. 2010. Highly integrated packet-based AER communication infrastructure with 3Gevent/s throughput. *Proc. 17th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 950–953.
- Heeger DJ. 1992a. Half-squaring in responses of cat striate cells. *Visual Neurosci.* **9**(5), 427–443.
- Heeger DJ. 1992b. Normalization of cell responses in cat striate cortex. *Visual Neurosci.* **9**(2), 181–197.
- Higgins CM and Shams SA. 2002. A biologically inspired modular VLSI system for visual measurement of self-motion. *IEEE Sensors J.* **2**(6), 508–528.
- Hubel DH. 1988 *Eye, Brain, and Vision*. WH Freeman, New York.
- Hubel DH and Wiesel TN. 1972. Laminar and columnar distribution of geniculo-cortical fibers in the macaque monkey. *J. Comp. Neurol.* **146**(4), 421–450.
- Iakymchuk T, Rosado A, Serrano-Gotarredona T, Linares-Barranco B, Jiménez-Fernández A, Linares-Barranco A, and Jiménez-Moreno G. 2014. An AER handshake-less modular infrastructure PCB with x8 2.5 Gbps LVDS serial links. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1556–1559.
- Indiveri G, Whatley AM, and Kramer J. 1999. A reconfigurable neuromorphic VLSI multi-chip system applied to visual motion computation *Proceedings of 7th International Conference on Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems (MicroNeuro)*, pp. 37–44.
- jAER Hardware Reference. n.d. *jAER Hardware Reference Designs*. <http://jaerproject.net/Hardware/> (accessed August 5, 2014).
- Jones JP and Palmer LA. 1987. An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *J. Neurophys.* **58**(6), 1233–1258.
- Kandel ER, Schwartz JH, and Jessell TM. 2000. *Principles of Neural Science*. 4th edn. McGraw-Hill.
- Kolle Riis H and Häfliger P. 2005. An asynchronous 4-to-4 AER mapper. *Lecture Notes in Computer Science*, vol. 3512. Springer. pp. 494–501.
- LeCun Y, Bottou L, Bengio Y, and Haffner P. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324.
- Lehky SR and Sejnowski TJ. 1988. Network model of shape-from-shading: neural function arises from both receptive and projective fields. *Nature* **333**, 452–454.
- Lin J, Merolla P, Arthur J, and Boahen K. 2006. Programmable connections in neuromorphic grids. *Proc. 49th IEEE Int. Midwest Symp. Circuits Syst.*, pp. 80–84.
- Linares-Barranco A, Jimenez-Moreno G, Linares-Barranco B, and Civit-Balcells A. 2006. On algorithmic rate-coded AER generation. *IEEE Trans. Neural Netw.* **17**(3), 771–788.
- Linares-Barranco A, Gomez-Rodriguez F, Jimenez G, Delbrück T, Berner R, and Liu S. 2009a. Implementation of a time-warping AER mapper. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2886–2889.
- Linares-Barranco A, Paz R, Gómez-Rodrguez F, Jiménez A, Rivas M, Jiménez G, and Civit A. 2009b. FPGA implementations comparison of neuro-cortical inspired convolution processors for spiking systems. In: *Bio-Inspired Systems: Computational and Ambient Intelligence*. Lecture Notes in Computer Science, vol. 5517. Springer. pp. 97–105.
- Liu SC, Kramer J, Indiveri G, Delbrück T, Burg T, and Douglas R. 2001. Orientation-selective aVLSI spiking neurons. *Neural Netw.* **14**(6/7), 629–643.
- Mahowald M. 1992. *VLSI Analogs of Neural Visual Processing: A Synthesis of Form and Function*. PhD thesis. California Institute of Technology, Pasadena, CA.

- Merolla P, Arthur J, and Wittig J. 2005. The USB revolution. *The Neuromorphic Engineer* **2**(2), 10–11.
- Miró-Amarante L, Jiménez A, Linares-Barranco A, Gómez-Rodríguez F, Paz R, Jiménez G, Civit A, and Serrano-Gotarredona R. 2006. A LVDS serial AER link. *Proc. IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 938–941.
- Neftci E, Chicca E, Cook M, Indiveri G, and Douglas R. 2010. State-dependent sensory processing in networks of VLSI spiking neurons *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2789–2792.
- Neubauer C. 1998. Evaluation of convolution neural networks for visual recognition. *IEEE Trans. Neural Netw.* **9**(4), 685–696.
- Okuno H and Yagi T. 2012. Image sensor system with bio-inspired efficient coding and adaptation. *IEEE Trans. Biomed. Circuits Syst.* **6**(4), 375–384.
- Orchard G, Thakor NV, and Etienne-Cummings R. 2013. Real-time motion estimation using spatiotemporal filtering in FPGA. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 306–309.
- Oster M and Liu SC. 2004. A winner-take-all spiking network with spiking inputs. *Proc. 11th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 203–206.
- Oster M, Wang YX, Douglas R, and Liu SC. 2008. Quantification of a spike-based winner-take-all VLSI network. *IEEE Trans. Circuits Syst. I: Regular Papers* **55**(10), 3160–3169.
- Ozalevli E and Higgins CM. 2005. Reconfigurable biologically-inspired visual motion systems using modular neuromorphic VLSI chips. *IEEE Trans. Circuits Syst. I: Regular Papers* **52**(1), 79–92.
- Park J, Yu T, Maier C, Joshi S, and Cauwenberghs G. 2012. Live demonstration: hierarchical address-event routing architecture for reconfigurable large scale neuromorphic systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 707–711.
- Paz R. 2003. Análisis del bus PCI. Desarrollo de puentes basados en FPGA para placas PCI. Trabajo de investigación para obtención de suficiencia investigadora. Sevilla.
- Paz R, Gomez-Rodriguez F, Rodriguez MA, Linares-Barranco A, Jimenez G, and Civit A. 2005. Test infrastructure for address-event-representation communications. Lecture Notes in Computer Science, vol. 3512. Springer. pp. 518–526.
- Paz-Vicente R, Linares-Barranco A, Cascado D, Rodriguez MA, Jimenez G, Civit A and Sevillano JL. 2006. PCI-AER interface for neuro-inspired spiking systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3161–3164.
- Paz-Vicente R, Jimenez-Fernandez A, Linares-Barranco A, Moreno G, Gomez-Rodriguez F, Miro-Amarante L, and Civit-Balcells A. 2008. Image convolution using a probabilistic mapper on USB-AER board. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1056–1059.
- Perez-Peña F, Morgado-Estevez A, Linares-Barranco A, Jimenez-Fernandez A, Gomez-Rodriguez F, Jimenez-Moreno G, and Lopez-Coronado J. 2013. Neuro-inspired spike-based motion: from dynamic vision sensor to robot motor open-loop control through Spike-VITE. *Sensors* **13**(11), 15805–15832.
- Petre C, Schlottmann C, and Hasler P. 2008. Automated conversion of Simulink designs to analog hardware on an FPAAs. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 500–503.
- Riesenhuber M and Poggio T. 2000. Models of object recognition. *Nat. Neurosci.* **3**, 1199–1204.
- Sabarat J, Kestur S, Park MS, Dantara D, Narayanan V, Chen Y, and Khosla D. 2012. A reconfigurable accelerator for neuromorphic object recognition. *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 813–818.
- Serrano-Gotarredona R, Oster M, Lichtsteiner P, Linares-Barranco A, Paz-Vicente R, Gómez-Rodríguez F, Riis HK, Delbrück T, Liu SC, Zahnd S, Whatley AM, Douglas R, Häfliger P, Jimenez-Moreno G, Civit A, Serrano-Gotarredona T, Acosta-Jiménez A, and Linares-Barranco B. 2005. AER building blocks for multi-layers multi-chips neuromorphic vision systems. *Advances in Neural Information Processing Systems 18 (NIPS)*, pp. 1217–1224.
- Serrano-Gotarredona R, Oster M, Lichtsteiner P, Linares-Barranco A, Paz-Vicente R, Gomez-Rodriguez F, Camunas-Mesa L, Berner R, Rivas M, Delbrück T, Liu SC, Douglas R, Häfliger P, Jimenez-Moreno G, Civit A, Serrano-Gotarredona T, Acosta-Jiménez A, and Linares-Barranco B. 2009. CAVIAR: a 45K-neuron, 5M-synapse, 12G-connects/sec AER hardware sensory-processing-learning-actuating system for high speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* **20**(9), 1417–1438.
- Serrano-Gotarredona R, Serrano-Gotarredona T, Acosta-Jiménez A, and Linares-Barranco B. 2006. A neuromorphic cortical-layer microchip for spike-based event processing vision systems. *IEEE Trans. Circuits Syst. I: Regular Papers* **53**(12), 2548–2566.
- Serrano-Gotarredona T, Andreou A, and Linares-Barranco B. 1999a. AER image filtering architecture for vision-processing systems. *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.* **46**(9), 1064–1071.

- Serrano-Gotarredona T, Andreou AG, and Linares-Barranco B. 1999b. Programmable 2D image filter for AER vision processing. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* 4, pp. 159–162.
- Shanley T and Anderson D. 1999. *PCI System Architecture*. PC System Architecture Series, 4th edn. Mindshare, Inc. / Addison-Wesley, Boston, MA.
- Shimonomura K, Kameda S, Iwata A, and Yagi T. 2011. Wide-dynamic range APS-based silicon retina with brightness constancy. *IEEE Trans. Neural Netw.* **22**(9), 1482–1493.
- Sivilotti M. 1991. *Wiring Considerations in Analog VLSI Systems with Application to Field-Programmable Networks*. PhD thesis. California Institute of Technology, Pasadena, CA.
- Texas Instruments. 2008. *TLK3101 Datasheet: 2.5 to 3.125 Gbps Transceiver (Rev. B)*, <http://www.ti.com/product/tlk3101#technicaldocuments> (accessed August 5, 2014).
- Tsang EKC and Shi BE. 2004. A preference for phase-based disparity in a neuromorphic implementation of the binocular energy model. *Neural Comput.* **16**(8), 1597–1600.
- Venier P, Mortara A, Arreguit X, and Vittoz EA. 1997. An integrated cortical layer for orientation enhancement. *IEEE J. Solid-State Circuits* **32**(2), 177–186.
- Vogelstein RJ, Mallik U, Culurciello E, Etienne-Cummings R, and Cauwenberghs G. 2004. Spatial acuity modulation of an address-event imager. *Proc. 11th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 207–210.
- Vogelstein RJ, Mallik U, Culurciello E, Cauwenberghs G, and Etienne-Cummings R. 2007a. A multichip neuromorphic system for spike-based visual information processing. *Neural Comput.* **19**(9), 2281–2300.
- Vogelstein RJ, Mallik U, Vogelstein JT, and Cauwenberghs G. 2007b. Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses. *IEEE Trans. Neural Netw.* **18**(1), 253–265.
- Whatley AM. 1997. *Silicon Cortex Software Design Rev. 6*, <http://www.ini.uzh.ch/~amw/scx/scx1swod.pdf> (accessed August 5, 2014).
- Zaghoul KA and Boahen KA. 2004. Optic nerve signals in a neuromorphic chip I: outer and inner retina models. *IEEE Trans. Biomed. Eng.* **51**(4), 657–666.
- Zamarreño-Ramos C, Serrano-Gotarredona T, and Linares-Barranco B. 2011. An instant-startup jitter-tolerant Manchester-encoding serializer/deserializer scheme for event-driven bit-serial LVDS interchip AER links. *IEEE Trans. Circuits Syst. I: Regular Papers* **58**(11), 2647–2660.
- Zamarreño-Ramos C, Serrano-Gotarredona T, and Linares-Barranco B. 2012. A $0.35\text{ }\mu\text{m}$ sub-ns wake-up time ON-OFF switchable LVDS driver-receiver chip I/O pad pair for rate-dependent power saving in AER bit-serial links. *IEEE Trans. Biomed. Circuits Syst.* **6**(5), 486–497.
- Zamarreño-Ramos C, Linares-Barranco A, Serrano-Gotarredona T, and Linares-Barranco B. 2013. Multicasting mesh AER: a scalable assembly approach for reconfigurable neuromorphic structured AER systems. Application to ConvNets. *IEEE Trans. Biomed. Circuits Syst.* **7**(1), 82–102.

14

Software Infrastructure



Software used in neuromorphic systems can be categorized according to the role(s) it performs. Each of these roles has particular features and presents particular challenges. Optimization, and Application Programming Interface (API) design are important, especially for software that is directly involved in processing streams of address-events. Several example software systems are briefly presented.

14.1 Introduction

The software used in AER systems is an often neglected component of such systems and historically, relatively little has been published on the subject (with a few exceptions, e.g., Dante et al. 2005; Delbrück 2008; Oster et al. 2005), although it forms an essential part of all but the simplest of systems. The software in AER systems generally covers one or more of the following roles: chip and system description, configuration, AE stream handling, mapping, and placement and routing.

Chip and System Description software generally consists of databases and/or description languages, which enable knowledge of the properties of relatively disparate hardware to be

maintained and interrogated by one or more of the remaining kinds of software in a uniform manner.

By *Configuration Software*, we mean software which is not necessarily involved once the AER system is up and running; but which is used to configure, for instance various parameters and bias values in mixed-signal chips; and software which is used for parameter tuning and calibration.

By *AE Stream Handling Software*, we mean software which actively participates in the handling of the streams of AEs on their way to and from hardware devices, to replay or generate stimuli for the hardware, or to capture AEs for the purposes of display, statistical analysis, later replay, and so on, or even to carry out algorithmic processing on the AE streams, as further discussed in Chapter 15.

Mapping Software is software which controls the mapping of AEs between address spaces (one-to-one mappings) or to perform fan-out (one-to-many mappings).

Placement and Routing Software is involved in configuring large scale AER systems consisting of many identical hardware devices with multiple possible paths for AEs to flow between them. The task of such software is to optimally distribute or *place* neural populations across the available hardware and to determine how to optimally *route* the AE traffic between them. This task is analogous to the placement and routing problems which arise in the design of printed circuit boards, integrated circuits in general, and field programmable gate arrays. Software for doing this is beyond the scope of this chapter, but see for example Brüderle et al. (2011) and Ehrlich et al. (2010) (in which placement and routing is referred to as *mapping*), and Chapter 16.

14.1.1 Importance of Cross-Community Commonality

To promote the development of neuromorphic systems to a scale that can deal with real-world problems, the community relies on exchange and cooperation between different laboratories. Much effort has been put in to facilitate the communication between the researchers themselves, for example at the annual Telluride and CapoCaccia workshops or by the Institute of Neuromorphic Engineering (Cap n.d.; Ins n.d.). Multi-lab efforts such as those in for example the CAVIAR (CAVIAR 2002), ALAVLSI (e.g., Chicca et al. 2007), and later FACETS (Brüderle et al. 2011) projects were facilitated by having common chip setup descriptions, configuration interfaces, AE stream formats, and so on.

14.2 Chip and System Description Software

One of the challenges facing those who build neuromorphic systems is the complexity of the hardware. Even the simplest of systems may have tens of biases and parameters which must be adjusted to reach a desired operating regime. Frequently, compromises in the chip designs due for instance to area constraints lead to certain, but not all, parameters being shared across diverse structures on the chip. For example, on a hypothetical multineuron chip, all the weights of synapses of a certain type may be forced to be controlled by a single parameter across all neurons on the chip, whereas the weights of other synapses might be individually adjustable. Some neurons may be able to use on-chip connections and others not. The addresses emitted by a 1000-neuron chip might not range from 0–999, but from, say, 48–2046 with only the even numbered addresses being used, while the corresponding synapses might be addressed with

the neurons numbered in the opposite order in bits 5..14 (i.e., from 31 968 down to 0) with the index of the synapse along the neuron in the low order 5 bits. In hardware designs, all things are possible, and the software needs to take account of this!

The number of parameters grows with the size of systems. And inherent in the use of analog technology is the issue of mismatch, which means that two instances of what is notionally the same chip will generally require slightly different bias parameters to operate in approximately the same regime, so the parameter values for one chip are likely to be unique to that very chip and cannot be used for other chips of the same type. The proper design of bias generators (see Chapter 11) can help with this problem, but many of the current academic research developments still require custom settings or calibration for individual chips, and these settings must be maintained together with the hardware components. Furthermore, different chips will be mounted on different boards, and in an inhomogeneous system, each board type may be addressed in different ways.

In order to master all of this complexity, it helps build up a database of some kind describing the parameters, chips, boards, and so on involved, or potentially involved in a particular setup. For very simple systems, this could be done with simple text files, but typically some form of XML (Extensible Markup Language) is used.

Given a database (in whatever form) containing knowledge about the hardware system, it can be interrogated by all of the other software in a uniform manner to determine how to address given synapses and neurons, how to connect them, how to automatically generate graphical user interface (GUI) controls, how to display results, and so on.

14.2.1 Extensible Markup Language

XML, (see, e.g., Bradley 2002) lends itself well to describing typically hierarchically organized neuromorphic systems, since it itself is a hierarchical yet versatile format which can be easily extended. The files can be edited with a standard text editor and displayed in a browser. The syntax is understandable by anyone who is familiar with the concepts of HTML. Most importantly, one can easily import and process XML documents in a variety of programming environments, for example MATLAB, Java, and Python.

An entry in XML format consists of *tags* and *attributes* of the form

```
<tag attribute1="value1" attribute2="value2" ... > content /tag>
```

If no content is given, this can be shortened to

```
<tag attribute1="value1" attribute2="value2" ... />.
```

Hierarchical structures are built by nesting tags as content into other tags. Files can include arbitrary tags and attributes that are ignored during further processing, so any kind of additional content can be added.

14.2.2 NeuroML

A particularly important flavor of XML used in neural network modeling is NeuroML (Gleeson et al. 2010). NeuroML is structured into three levels and includes MorphML for describing the morphology of biological neurons, ChannelML for describing channel and synapse properties,

and NetworkML for describing networks of neurons. Not all of its levels are necessarily relevant in describing neuromorphic, hardware-based systems, but the structure of NeuroML makes it possible to use only the components which are relevant.

14.3 Configuration Software

To facilitate testing and operation of neuromorphic systems with a multitude of parameters, a software infrastructure to easily interface to the hardware is desirable. Without this, the setup and operation of the chips require too much skill and experience. Also it is often the case that parameters have to be tuned to a narrow operating range, and a working set of values established after many testing sessions should not be easily lost and should be easy to retrieve at a later time. Persistent storage of parameters between runtime sessions is essential so that users can tweak parameters and return to the same state later.

When hand-tuning is required, a GUI automatically constructed from a database description of the system is extremely useful. An example of such a system is presented in Section 14.6.1 below.

Hand-tuning, however, does not scale to larger systems. In larger systems it is more desirable to perform parameter estimation and calibration automatically. Until recently, automated methods for mapping VLSI circuit bias voltages to neural network type parameters were based on heuristics and result in *ad-hoc* custom-made calibration routines. For example, in Brüderle et al. (2009) the authors perform an exhaustive search of the parameter space to calibrate their hardware neural networks, using the simulator-independent description language ‘PyNN’ (Davison et al. 2008).

This type of brute-force approach is possible because of the accelerated nature of the hardware used, but it becomes intractable for real-time hardware or for very large systems, due to the massive amount of data that must be measured and analyzed to carry out the calibration procedure. An alternative model-based approach is proposed in Neftci et al. (2011), where the authors fit data from experimental measurements with equations from transistors, circuit models, and computational models to map the bias voltages of VLSI spiking neuron circuits to the parameters of the corresponding software neural network. This approach does not require the extensive parameter space search techniques, but new models and mappings need to be formulated every time a new circuit or chip is used, making its application quite laborious.

An example of automatic parameter estimation and calibration software is presented in Section 14.6.4.

14.4 Address Event Stream Handling Software

AE systems by their nature lend themselves rather well to their integration with this kind of software, as all of the information being conveyed has been converted to a digital form. However, applying software to AE processing is challenging from the point of view of latency and bandwidth. (For a formal treatment of latency and bandwidth, see Chapter 2 and Section 2.3 in particular.)

It is more efficient to process a buffer full of AE data than to process each event as it arrives, and even if the software is designed following the best principles for real-time systems, latencies may be bounded but not always consistent. Therefore, in order to retain the timing information present in an incoming AE stream on its asynchronous bus as it enters the

synchronous world of a computer, it is essential that the interface hardware stores not only the address embodied in the event, but also its arrival time to sufficient resolution in the form of a time-stamp. Thus, software handling AEs in a computer is usually dealing on the input side with data consisting of pairs of addresses and time-stamps. On the output side, a slightly different form is often required by the hardware, namely pairs of addresses and inter-spike intervals, that is times which are relative to the output of the preceding AE rather than to some clock which runs continuously.

14.4.1 Field-Programmable Gate Arrays

Even given highly optimized code in a hard real-time environment, conventional software ‘in-the-loop’ is unlikely to be able to keep up with the demands of processing all of the spikes flowing through an AE system. It is for this reason that *mappers* are usually constructed using FPGAs rather than CPUs to perform the mapping function. Of course, the FPGAs must be ‘programmed’ using a hardware description language (HDL), but this kind of logic design is outside of our scope here.

14.4.2 Structure of AE Stream Handling Software

Inevitably some AEs must leave the core of an AE system and enter a conventional computer, whether for debugging, monitoring, or control purposes. And often it is desirable to be able to feed a pre-computed AE stream, for instance a test stimulus, into an AE system from a computer. In these cases software must be written that directly handles an AE stream. This software typically contains drivers which talk to and provide an abstraction of the specific hardware being used, a library that presents a well-defined, stable API and applications written on top of this library, possibly with a GUI, to provide a means to monitor and record what is going on inside the AE system and possibly further process the AE output; or indeed to provide input such as a test stimulus to the AE system.

The capture and algorithmic processing of AE data on computers is central to jAER, which is introduced in Section 14.6.3, and this algorithmic processing is discussed separately in Chapter 15.

14.4.3 Bandwidth and Latency

Bandwidth and latency issues have already been discussed primarily in relation to hardware in Chapters 2 and 13, but of course bandwidth is also almost always an important consideration in AE stream handling software. Although it may be inherent in the nature of spike-based communication that the loss of a spike or two should not be critical, in experimental situations it is usually desired not to drop any spikes, but to be able to faithfully record all of the output from a hardware system under test. Latency may or not be an issue, depending on the nature of the system. Once AEs have been time-stamped by hardware, the original inter-spike intervals can always be recovered, so the latency between reading the AE and its time-stamp from the hardware and further processing steps within the computer is not generally an issue while such processing remains within the computer. However, if these AEs or subsequent AEs arising from their processing are to be reinjected into the same AE system from which they came, then the latency is likely to be critical, as providing spikes back into a not purely feed-forward

neural network too late, that is, after some essentially arbitrary delay, may have non-negligible effects. That being said, depending on the nature of the system, if it operates on biological neural timescales, jitter of up to a few hundred microseconds may be acceptable.

So how is high bandwidth and low latency to be achieved? Probably the most important concern is to minimize the copying of data (i.e., the AEs). In a naïve approach, within a monitor driver the data are copied from hardware buffers to kernel buffers, and then when demanded by the overlying application it is copied again to user space buffers. Each of these copies requires CPU time. In order to eliminate this, direct memory access (DMA) can be used (if the hardware supports it, and it should) to copy the data from hardware buffers. The use of DMA eliminates the CPU from being involved in the first copy, except insofar as it needs to set up the DMA transfer. If buffers can be memory mapped into user space, the second copy can be eliminated. Memory mapping of buffers into user space also avoids having to make time-consuming transitions into and out of kernel mode to read the data. One difficulty that can arise here is that the user space application needs a means to know how much data are available in the buffers, that is, what portion of the buffers contain valid data. The hardware and/or driver must make this information available by some means.

Always waiting until a given quantity (a buffer-full) of data is available before signaling to user space that data is available does not work for general purpose AE monitoring, since an AE system may sometimes produce very few output events for extended periods of time (e.g., a silicon retina observing an unchanging scene) and these output events would not be available to the application until later. As mentioned in the description of the early packet feature of the USBAERmini2 in Section 13.2.3 and in Section 15.2.1, it is important that hardware interfaces send their contents at a minimum rate (e.g., 1 kHz) even if their FIFOs are not full.

Typically, *overlapping* or *asynchronous* input and output must be performed to decouple data acquisition from processing. Separate threads or even processes are used to process the AE data and to perform the actual data acquisition and data output. If overlapping I/O is used, a processing thread can work on newly acquired data passed to it from an acquisition thread, while the acquisition thread is waiting for new data. Otherwise, the processor might be idle while waiting for new data to be captured.

If the data rate from a device is too high, it can overwhelm processing capability, possibly resulting in an ever-increasing backlog of data being held in memory. One way to deal with this is to allocate a certain maximum time for processing a buffer of data. If this time is exceeded, the rest of the data are discarded. This way, at least the most recent data are processed, even though some data are discarded. Another way to implement this approach in hardware is by only capturing events up to some determined rate. The rest of the events are then discarded by the hardware without being transmitted to the processor.

Similar, reciprocal arguments apply on the AE output (sequencer) side.

14.4.4 Optimization

It may be that to achieve the best possible throughput and lowest latency, the software will need to be optimized. However, one should not be tempted to micro-optimize from the word go. When Knuth (1974) wrote

[...] programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

This might have been (at least mostly) hyperbole, but in trying to optimize too early or too broadly, one can end up trying to second guess what optimizations the compiler or interpreter will perform and it is very easy to waste a great deal of time trying to optimize areas of code that do not need optimizing. The risk is that the code will end up being difficult to read, understand, and maintain.

On modern processors it is very difficult to predict how a given small change to code will affect its runtime performance, primarily because of cache effects (Drepper 2007). For instance, new buffers should not be allocated and old buffers freed. Re-using old buffers for new data not only avoids the pure processing time overhead associated with the allocation and freeing operations, it also means that there is a higher chance that the memory used remains cached.

Optimization needs to be performed in an empirical way. First, the performance of the software needs to be measured. A *profiler* can be used to determine which routines are taking up the most time and therefore where optimization effort is best spent. After each would-be optimization has been coded, the performance should be measured again to ensure that performance has indeed been improved, and not made worse. The need to perform such measurements should be considered during the design of the software and consideration should be given to building in a means of ‘instrumentation,’ although this should of course itself be designed to have a minimal impact on performance.

14.4.5 Application Programming Interface

For interoperability between different systems developed in different laboratories or between different generations of hardware from the same laboratory it is desirable that a common API be used, so that application-level software can be ported from one hardware base to another with as little effort as possible. Developing a good API is a difficult task, particularly where it should, to some degree, be ‘future-proof’ in a field in which the technology is developing rapidly. However, some aspects of dealing with AE streams are likely to remain unchanging over time, irrespective of whether the underlying hardware is PCI based or USB based for example. The fundamental operations of reading and writing an AE stream, and perhaps imposing hardware-level filters on such streams to select which AEs or ranges of AEs are monitored, remain the same. However the hardware has to play its part if true compatibility is to be achieved. If some hardware formats its AE streams differently from others, it is probably of insufficient benefit to spend processor cycles reformatting the data stream in the API layer for consistency between the two types of hardware.

Further points that are sometimes neglected in the development of APIs in this area are:

- for any property for which there is a ‘set’ function, there should also be a corresponding ‘get’ function;
- the ability to correlate the time-stamps applied by the hardware to the computer’s clock, and/or so-called ‘wall-clock’ time, necessary to relate AE data to stimuli presentation, other (e.g., oscilloscope) measurements, and so on;
- scalability, in terms of allowing for multiple hardware instances; and
- reentrancy, such that a library can be safely used by multiple threads.

14.4.6 Network Transport of AE Streams

For integration with other software components, some means of transferring AE streams across a network is useful. It is particularly useful to transport raw streams of AEs by UDP (Postel

1980) or TCP (Braden 1989; Postel 1981). UDP is not a reliable protocol in the technical sense, meaning that data may be lost or suffer errors or duplication. However, such occurrences are rare on modern networks and UDP is preferred over TCP for real-time applications requiring low latency and overhead. Also, UDP is connectionless, so transmitters and receivers can appear and disappear in any order.

For an introduction to writing programs that communicate across a network using UDP or TCP, see for example, Stevens et al. (2003).

14.5 Mapping Software

As explained in Section 14.4.1, mappers are usually constructed using FPGAs rather than CPUs to perform the mapping function so the ‘software’ (so far as the term is applicable at all) that actually performs the mapping function is typically written in VHDL, which is outside of the scope here. It is of course possible to create mappers which are algorithmic mappers, that is to say the destination addresses they emit are determined by applying some fixed arithmetical rules to the incoming source addresses to achieve say some fixed pattern of fan-out or projective field. But most mappers are constructed as more general purpose table-lookup driven mappers in which the incoming source addresses are merely looked up in a table in RAM in which the corresponding list of destination addresses is to be found. These general purpose mappers are much more flexible, but a software interface is needed to program the various mappings into them. This is the software we are concerned with, in Section 14.5.

One might consider mapping software to be an extension of the category of configuration software, but there are significant differences. First, the underlying hardware is different. Configuration software as defined here is typically dealing with setting parameters and biases on mixed-signal chips by manipulating digital to analog converters (DACs) or sending signals to on-chip bias generators, and is not generally directly involved in AE processing (although some chips require that parameterization information is conveyed into the chip by piggy-backing on an AE stream). Mapping software is however typically directed at writing lookup tables into RAM which will be read by the actual mapper hardware. Second, the parameters and biases set by the configuration software usually remain static for the course of an experiment, whereas mappings may need to be added and removed on-line as a result of learning algorithms being applied to the AE data.

Here again, many of the same considerations apply regarding constructing a useful API as were mentioned for the AE stream handling software APIs in Section 14.4.5. A common set of fundamental operations for a mapping API are:

- set a new mapping from a source address to a list of destination addresses, replacing an existing mapping if necessary.
- delete a mapping for a given source address such that the arrival of that source address no longer produces any output.
- determine the current mapping for a given source address.
- add additional destination event addresses to an existing mapping.
- remove a set of destination event addresses from an existing mapping.

As with the AE stream handling software APIs, where there is a ‘set’ function there should be a corresponding ‘get’ function, and scalability and reentrancy should also be considered. If

mappings need to be added and removed for on-line learning, the speed with which this can be done may be important, and the code paths involved in doing so may then be considered *hot paths*, which merit the same use of optimization as the direct AE stream handling software.

If the mapper lookup tables are implemented with flexible, variable destination event list lengths, then considerably more sophisticated memory management will need to be performed to keep track of free space within the mapper's lookup tables and to enable the addition and removal of individual destination event addresses to and from a given arbitrary length destination event list than would be the case with fixed length destination event lists.

14.6 Software Examples

Even single chips and small systems composing a few neuromorphic chips require configuration. In the earliest days of neuromorphic engineering, configuration consisted of connecting AER chips by ribbon cables and carefully turning potentiometers to set bias voltages. Nowadays, much, if not all, of the configuration is digitally programmable. Chapter 11 describes on-chip bias generators, but the values of these biases must be loaded onto the chips somehow. And Chapter 13 discusses mapper hardware, but the mapper lookup tables must be loaded from somewhere. This section discusses examples of software solutions to these needs.

14.6.1 ChipDatabase – A System for Tuning Neuromorphic aVLSI Chips¹

Oster (2005) described a software infrastructure for interfacing with the Rome PCI-AER board (Dante et al. 2005) and for setting up the configuration of biases on particular chips. An XML-based system called ChipDatabase is used to set biases on neuromorphic aVLSI chips: an XML file describes the chip pinout and how the pins are connected to digital/analog computer interfaces; and a MATLAB-based GUI is created to provide an intuitive method for tuning the biases of neuromorphic chips. This setup facilitates the exchange of aVLSI chips by defining a common interface, allowing remote tuning and the sharing of bias settings over the web.

The ChipDatabase project created a GUI (see example in Figure 14.1) that allowed the user to set biases from a standard MATLAB working environment, without knowing about the underlying hardware interfaces. It defined a standardized documentation of chips, adapter boards, and setup that included names instead of cryptic pin numbers, a description of bias functionalities, default voltages, and so on, all together in a flexible, easy-to-use, and extendable database format (XML). It also provided mechanisms to easily exchange the documentation and tuning settings between different researchers over a common web interface and it used computer controlled DAC hardware, controlled by a high-level mathematical language, to allow easy and complete characterization of chips. The same environment and data acquisition systems could be used for the remote tuning of chips when they were exchanged between different laboratories.

All these capabilities come at a price, of course. A lot of information has to be entered before the GUI for a particular chip can be used. However, it is also necessary and useful that every chip has a common standard description. ChipDatabase also distinguished between the

¹ The text in this section is chiefly adapted from Oster (2004). Reproduced with permission of Matthias Oster.

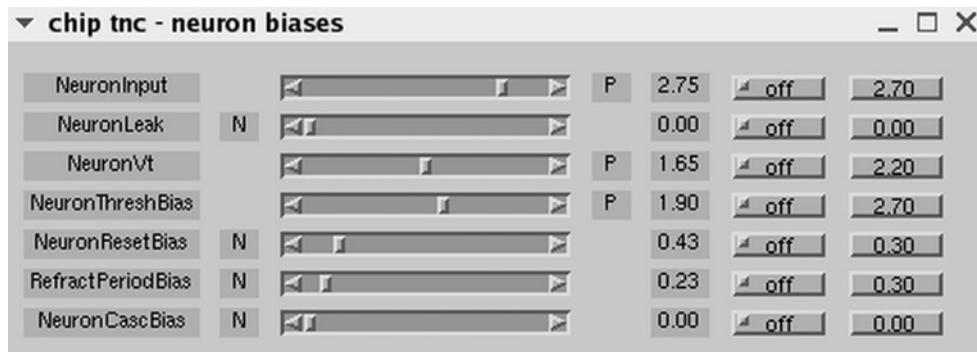


Figure 14.1 Example bias group window in the ChipDatabase Graphical User Interface (GUI). For each bias in a group, it contains the bias name and a slider to graphically set the bias value. The N or P marker determines the direction in which the voltage value decreases. The text field shows the value that is currently set. If the off button is checked, the bias is set to the ‘off’ value. When the button is unchecked, the voltage that was active when switching off is restored. The rightmost button is a push button to set the current voltage to the predefined default voltage from the chip class definition. The working settings can be saved to and restored from a file. From Oster (2004). Reproduced with permission of Matthias Oster

definition of the chip itself and the test setup consisting of, for example, the boards on which the chips are mounted, which are normally built by different developers.

The ChipDatabase setup was used in several projects (CAVIAR, ALAVLSI, and in the other academic projects). The CAVIAR and ALAVLSI systems used the DAC boards developed in the CAVIAR project as the underlying DAC hardware, whereas some other projects used dedicated boards with additional functionality.

As a good example of the software design principles of *information hiding* and *encapsulation*, the functionality of the different DAC interface boards used in various projects is hidden from the database code by different hardware ‘drivers’. These drivers also encapsulate the low-level communication functions that are dependent on the operating system (OS), for example, to access different OS-specific code on Windows and Linux machines. The GUI code simply calls one of four commands which have to be supplied by the drivers:

```
setchannel (dacboardid, channel, value, type)
value = getchannel (dacboardid, channel, type)
setchannels (dacboardid, values, type)
values = getchannels (dacboardid, type)
```

The use of device descriptors, sub-device types, and channel numbers is also a good step in the direction of the adoption of existing interfaces, in that it was designed to be compatible with the interface defined by the `comedi` project (Hess and Abbott 2012), a project which provides drivers for many data acquisition cards. This would make it easy to provide a generic interface to the standardized `comedi` functions and thus make the ChipDatabase software usable with any of the data acquisition cards supported by `comedi`. Adopting existing standards and

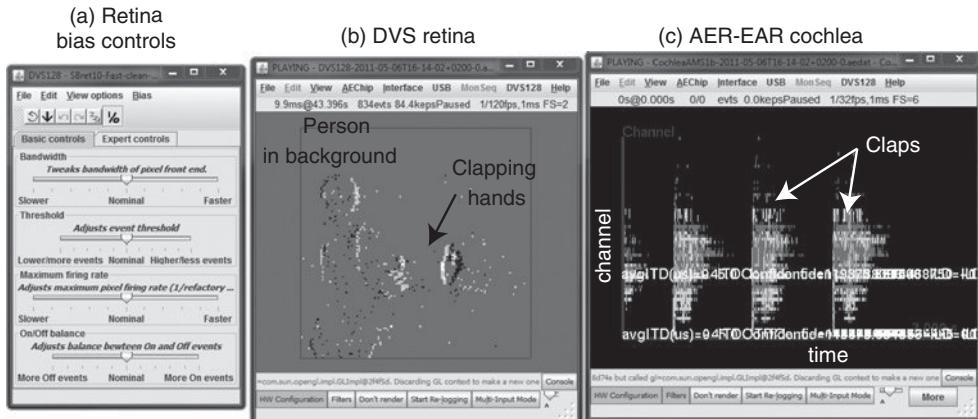


Figure 14.2 Example jAER windows showing the synchronized playback of the AE output from a retina chip and a cochlea chip. (a) User-friendly control of DVS silicon retina biases. (b) DVS output rendered as a 2D histogram of events over the last 20 ms. (c) Output from an AER-EAR silicon cochlea rendered as a spike rastergram. As one of the people seen in the DVS view claps their hands together, bursts of cochlea spikes are produced

interfaces rather than taking a not-invented-here approach (i.e., reinventing the wheel) can not only save initial development time, but also make future, perhaps originally unforeseen, integration with other software much easier.

14.6.2 Spike Toolbox²

One example of software that creates AE streams for injection into AE systems, and also for monitoring the AE streams produced by such systems is the Spike Toolbox (Muir 2008). This is a custom MATLAB toolbox for the off-line generation, manipulation, and analysis of digital spike trains. It allows arbitrary spike trains to be easily generated with control over temporal structure and allows the trains to be manipulated as an opaque objects.

The toolbox also has links for stimulating external spike-based communications devices, using either the Rome PCI-AER hardware (Section 13.2.2) or a so-called ‘spike server’. The toolbox can monitor spikes from devices such as spiking retinas directly from MATLAB and can be configured for arbitrary hardware addressing schemes.

14.6.3 jAER

In contrast to previous software packages, the Java-based software project jAER (usually pronounced ‘jay-er’) is focused on real-time processing of AER sensor output (Delbrück 2008; jAER 2007). Figure 14.2 shows jAER rendering the output from two chips simultaneously. Other systems integrated in jAER include silicon retinas from several groups, convolution chips, AER monitor and sequencer boards developed in the CAVIAR project, silicon cochleas

² The text in this section is taken from Muir (2008). Reproduced with permission of Dylan Muir.

as described in Chapter 4, servo motor controllers, and several special-purpose optical sensors. jAER has been used for chip testing, algorithm development, and the creation of entire robots such as the robotic goalie (Delbrück and Lichtsteiner 2007) and the DVS-based pencil balancer (Conradt et al. 2009).

jaER applications are mostly written in Java, currently the most popular programming language (TIOBE 2013). It allows plugging in one or more AER device with USB interfaces, and then viewing the events coming from the devices, logging them to disk and playing them back. Network transport of events via TCP or UDP are also supported and have been used for example to interface 10 DVS silicon retinas in a permanent installation in a railway station in Switzerland. Here the event streams from several DVS retinas are fused together to form a single very wide view of a railway station passenger underpass (Derrer et al. n.d.).

UDP control of jaER allows users to control experiments from a dynamic programming environment such as Matlab. A variety of pre-built ‘filters’ allow for reducing noise, extracting low-level features, tracking objects, and controlling servo motors. Generally an application in jaER is written as a pipeline of previously developed filters nested inside a custom filter. Java introspection mechanisms are used to automatically build GUI control panels that allow control and persistent storage of parameters.

jaER also supports chips with the programmable bias current generators discussed in Chapter 11 to provide persistent GUI control of chip biases (see Section 11.5). jaER also serves as the repository for full-design kits for on-chip bias generators, with sample schematics, chip layout, board design, firmware, and host-side software.

jaER’s internal model of processing based on temporally-ordered packets of time-stamped event objects has prompted a new way of thinking about how to perform computer vision and audition tasks on the basis of applying iterative algorithms to these packets. These algorithms are discussed in Chapter 15.

14.6.4 Python and PyNN

The Python-based PyNN (pronounced the same as ‘pine’) simulator-independent framework for building neuronal network models (Davison et al. 2008) has in recent years practically revolutionized the use of software in modeling neural networks and helped bring the Python programming language (Python 2012) to prominence in the field (Gewaltig et al. 2009). Working in Python brings the advantage of giving the programmer access to a vast collection of libraries that have been developed in other fields for scientific computing and plotting, amongst other things. It is platform independent and easy to extend using other programming languages.

Python for Neural Networks

PyNN was developed with the intention of providing a common high-level API for multiple neural network simulators (e.g., NEURON, Hines and Carnevale 2003, NEST, Diesmann and Gewaltig 2002, PCSIM, Pecevski et al. 2009, and Brian, Goodman and Brette 2008). This allows a network model to be written once and then be run on any of the supported simulators. PyNN not only supports modeling networks at the level of populations of neurons, layers, columns, and the connections between them, but also supports dealing with individual neurons and synapses. It provides a set of simulator-independent models of neurons, synapses

and synaptic plasticity, and various connectivity algorithms, while still allowing connectivity to be specified by the user.

Given suitable back-ends, PyNN can also be used to interface to neuromorphic hardware, as has been done as part of the FACETS project (Brüderle et al. 2009, 2011) and in the SpiNNaker project (Galluppi et al. 2012). This gives the advantage that modelers can move their models directly from the simulator of their choice onto neuromorphic hardware without having to learn about the details of the hardware implementation.

pyNCS and pyTune³

An alternative approach, also based on Python, has been implemented by Sheik et al. (2011) to simplify the configuration of multichip neuromorphic VLSI systems, and automate the mapping of neural network model parameters to neuromorphic circuit bias values.

Sheik et al. (2011) proposed a modular framework for the tuning of parameters on multichip neuromorphic systems. On the one hand, the modularity of the framework allows the definition of a wide range of models (network, neural, synapse, circuit) that can be used in the parameter translation routines; on the other hand, the framework does not require detailed knowledge of the hardware/circuit properties, and can optimize the search and evaluate the effectiveness of the parameter translations by measuring experimentally the behavior of the hardware neural network. This framework was implemented using Python, and makes use of its object-oriented features.

The framework consists of two software modules: pyNCS (Stefanini et al. 2014) and pyTune. The pyNCS toolset allows the user to interface the hardware to a workstation, to access and modify the VLSI chip bias settings, and to define the functional circuit blocks of the hardware system as abstract software modules. The abstracted components represent computational neuroscience relevant entities (e.g., synapses, neurons, populations of neurons, etc.), which do not depend directly on the chip's specific circuit details and provide a framework that is independent of the hardware used. The pyTune toolset allows users to define abstract high-level parameters of these computational neuroscience relevant entities, as functions of other high- or low-level parameters (such as circuit bias settings). This toolset can then be used to automatically calibrate the properties of the corresponding hardware components (neurons, synapses, conductances, etc.), or to determine the optimal set of high- and low-level parameters that minimize arbitrarily defined cost-functions.

Using this framework, neuromorphic hardware systems can be automatically configured to reach a desired configuration or state, and parameters can be tuned to maintain the system in the optimal state.

The pyNCS Toolset

At the lowest level, dedicated drivers are required to interface custom neuromorphic chips to computers. Although custom drivers must be developed for each specific hardware, they can be cast as Python modules and integrated as plug-ins in the pyNCS toolset. Once the drivers are implemented, pyNCS creates an abstraction layer to simplify the configuration of the hardware

³ The text in this section is © 2011 IEEE. Reprinted, with permission, from Sheik et al. (2011).

and its integration with other software modules. The experimental setup is then defined using information provided by the designer on the circuit functional blocks, their configuration biases, and the chip's analog and digital input and output channels. The setup, the circuits, and their biases are encapsulated into abstract components controllable via a GUI or an API.

Experiments (equivalent to software simulation runs) can be defined, set-up, and carried out, using methods and commands analogous to those present in software neural simulators such as Brian (Goodman and Brette 2008) or PCSIM (Pecevski et al. 2009).

pyNCS uses a client-server architecture, thereby allowing multiclient support, load sharing, and remote access to the multichip setups. Thanks to this client-server architecture; multiple clients can control the hardware remotely, regardless of the OS used.

The pyTune Toolset

The pyTune toolset is a Python module which automatically calibrates user-defined, high-level parameters and optimizes user-defined cost-functions. The parameters are defined using a dependency tree that specifies lower-level sub-parameters in a recursive hierarchical way. This hierarchical scheme allows the definition of complex parameters and related cost-functions. For example, synaptic efficacies in neural network models can be related to the bias voltages which control the gain of synaptic circuits in neuromorphic chips. Using the pyTune toolset it is possible to automatically search a space of bias voltages and set a desired synaptic efficacy by measuring the neuron's response properties from the chip.

This automated parameter search can be applied to more complex scenarios to optimize high-level parameters related to network properties. For example, the user can specify the mapping between low-level parameters and the gain of a winner-take-all network (Yuille and Geiger 2003), or the error of a learning algorithm (Hertz et al. 1991).

The pyTune toolset relies on the translation of the problem into parameter dependencies. The user defines each parameter by its measurement routine (`getValue` function) and its sub-parameter dependencies. At the lowest level, the parameters are defined only by their interaction with the hardware, that is they represent circuit biases. The user can choose a minimization algorithm from those available in the package or can define custom methods to do the optimization that sets the parameters' values. Optionally one can also define a specific cost function that needs to be minimized. By default, the cost function is computed as $(p - p_{\text{desired}})^2$ where p is the current measured value of the parameter and p_{desired} is the desired value. Explicit options (such as maximum tolerance for the desired value, maximum number of iteration steps, etc.) can also be passed as arguments to the optimization function.

Finally, the sub-parameters' methods are mapped by the appropriate plug-in onto the corresponding driver calls in the case of a hardware system, or onto method calls and variables in the case of a system simulated in software. Each mapping specific to a system has to be separately implemented and included in pyTune as a plug-in.

The pyTune toolset can be used to adjust the corresponding parameters and obtain the desired neural properties irrespective of temperature effects, mismatch between different instances of the chips and other sources of heterogeneity because it relies on measuring outputs from the chips.

Modularity and Integration with other Python Tools

Thanks to the modularity of pyNCS and pyTune, they are in principle completely compatible with other existing Python tools such as PyNN, and can be considered as an additional useful

tool that can be included in the increasing number of Python applications developed for the neuroscience and neuromorphic engineering community.

14.7 Discussion

Here in this chapter we have been principally concerned with software used in relatively small systems. In the domains of chip and system description software, mapping software, and even in the challenging area of AE stream handling software; the necessary techniques and technologies are essentially known and available. To some extent creating software in these areas is a matter of following best practices in database design, API design, buffering, optimization. In medium and large scale systems, such as those described in Chapter 16, there are significant scalability challenges, particularly as touched upon in Section 14.3 and Section 14.6.4 in the realms of configuration software, and placement and routing software, as mentioned in Section 14.1. However, the kind of software which has been described in the present chapter remains an indispensable ingredient, also in larger systems. And it formed part of the inspiration for the algorithmic processing of AE event streams described in Chapter 15.

References

- Braden R. 1989. Requirements for internet hosts – communication layers. RFC 1122, RFC Editor. <http://www.rfc-editor.org/rfc/rfc1122.txt> (accessed August 6, 2014).
- Bradley N. 2002. *The XML Companion*. 3rd edn. Addison-Wesley.
- Brüderle D, Müller E, Davison A, Muller E, Schemmel J, and Meier K. 2009. Establishing a novel modeling tool: a Python-based interface for a neuromorphic hardware system. *Front. Neuroinformat.* **3**, 17, doi:10.3389/neuro.11.017.2009.
- Brüderle D, Petrovici MA, Vogginger B, Ehrlich M, Pfeil T, Millner S, Grübl A, Wendt K, Müller E, Schwartz MO, de Oliveira D, Jeltsch S, Fieres J, Schilling M, Müller P, Breitwieser O, Petkov V, Muller L, Davison A, Krishnamurthy P, Kremkow J, Lundqvist M, Muller E, Partzsch J, Scholze S, Zühl L, Mayr C, Destexhe A, Diesmann M, Potjans T, Lansner A, Schüffny R, Schemmel J, and Meier K. 2011. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biol. Cybern.* **104**(4), 263–296.
- Cap. n.d. Capo Caccia Cognitive Neuromorphic Engineering Workshop. <http://capocaccia.ethz.ch/> (accessed August 6, 2014).
- CAVIAR. 2002. CAVIAR Project, <http://www.imse-cnm.csic.es/caviar/> (accessed August 6, 2014).
- Chicca E, Whatley AM, Dante V, Lichtsteiner P, Delbrück T, Del Giudice P, Douglas RJ, and Indiveri G. 2007. A multi-chip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity. *IEEE Trans. Circuits Syst. I* **54**(5), 981–993.
- Conradt J, Cook M, Berner R, Lichtsteiner P, Douglas RJ, and Delbrück T. 2009. A pencil balancing robot using a pair of AER dynamic vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 781–784.
- Dante V, Del Giudice P, and Whatley AM. 2005. Hardware and software for interfacing to address-event based neuromorphic systems. *The Neuromorphic Engineer* **2**(1), 5–6.
- Davison AP, Brüderle D, Eppler JM, Kremkow J, Muller E, Pecevski DA, Perrinet L, and Yger P. 2008. PyNN: a common interface for neuronal network simulators. *Front. Neuroinformat.* **2**, 11. doi:10.3389/neuro.11.011.2008.
- Delbrück T. 2008. Frame-free dynamic digital vision. *Proceedings of International Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, University of Tokyo, March 6–7. pp. 21–26.
- Delbrück T and Lichtsteiner P. 2007. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 845–848.
- Derrer R, Jauslin S, Vehovar M, Suisseplan Ingenieure AG, Vehovar & Jauslin Architektur AG, Keller S, Gössling L, Delbrück T, Brändli C, Steinweber P, Schilling M, Santana E, and Schnick-Schnack-Systems GmbH. n.d. Atelier Derrer | Gravity – Bahnhof Aarau, <http://www.lightlife.de/gravity-bahnhof-aarau/> (accessed August 6, 2014).

- Diesmann M and Gewaltig M. 2002. Nest: an environment for neural systems simulations. In: *Forschung und wissenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001* (eds Plessner T and Macho V), vol. 58. Gesellschaft für wissenschaftliche Datenverarbeitung, pp. 43–70.
- Drepper U. 2007. What every programmer should know about memory. Technical report, Red Hat Inc, <http://people.redhat.com/drepper/cpumemory.pdf> (accessed August 6, 2014).
- Ehrlich M, Wendt K, Zühl L, Schüffny R, Brüderle D, Müller E, and Vogginger B. 2010. A software framework for mapping neural networks to a wafer-scale neuromorphic hardware system. *Proc. Artificial Neural Netw. Intell. Inf. Process. Conf. (ANNIIP)*, pp. 43–52.
- Galluppi F, Davies S, Rast A, Sharp T, Plana LA, and Furber S. 2012. A hierarchical configuration system for a massively parallel neural hardware platform. *Proceedings of the 9th Conference on Computing Frontiers (CF '12)*, pp. 183–192.
- Gewaltig MO, Hines M, Kötter R, Diesmann M, Davison AP, Muller E, and Bednar JA. 2009. Python in neuroscience, http://www.frontiersin.org/Neuroinformatics/researchtopics/Python_in_neuroscience/8 (accessed August 6, 2014).
- Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TM, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, and Silver RA. 2010. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput. Biol.* **6**(6), e1000815.
- Goodman DF and Brette R. 2008. Brian: a simulator for spiking neural networks in Python. *Front. Neuroinformat.* **2**, 5. doi:10.3389/neuro.11.005.2008.
- Hertz J, Krogh A, and Palmer RG. 1991. *Introduction to the Theory of Neural Computation*. Addison Wesley, Reading, MA.
- Hess FM and Abbott I. 2012. Comedi: linux control and measurement device interface, <http://www.comedi.org/> (accessed August 6, 2014).
- Hines ML and Carnevale NT. 2003. The NEURON simulation environment. In: *The HandBook of Brain Theory and Neural Networks* (ed. Arbib MA), 2nd edn. MIT Press, Cambridge, MA, pp. 769–773.
- Ins. n.d. Institute of Neuromorphic Engineering, <http://www.ine-web.org/> (accessed August 6, 2014).
- jAER. 2007. jAER Open Source Project, <http://jaerproject.org> (accessed August 6, 2014).
- Knuth DE. 1974. Computer programming as an art. *Commun. ACM* **17**(12), 667–673.
- Muir D. 2008. Spike toolbox for matlab, <http://spike-toolbox.ini.uzh.ch/> (accessed August 6, 2014).
- Neftci E, Chicca E, Indiveri G, and Douglas R. 2011. A systematic method for configuring VLSI networks of spiking neurons. *Neural Comput.* **23**(10), 2457–2497.
- Oster M. 2004. ChipDatabase – a system for tuning neuromorphic aVLSI chips, <http://www.ini.ethz.ch/~mao/ChipDatabase/ChipDatabase.pdf> (accessed August 6, 2014).
- Oster M. 2005. Tuning aVLSI chips with a mouse click. *The Neuromorphic Engineer* **2**(1), 9.
- Oster M, Whatley AM, Liu SC, and Douglas RJ. 2005. A hardware/software framework for real-time spiking systems. Springer Lecture Notes in Computer Science (ed. Duch W, Kacprzyk J, Oja E and Zadrożny S), vol. 3696. Springer GmbH, Heidelberg, pp. 161–166.
- Pecevski D, Natschläger T, and Schuch K. 2009. PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. *Front. Neuroinformat.* **3**, 11. doi:10.3389/neuro.11.011.2009.
- Postel J. 1980. User datagram protocol. RFC 768, RFC Editor, <http://www.rfc-editor.org/rfc/rfc768.txt> (accessed August 6, 2014).
- Postel J. 1981. Transmission control protocol. RFC 793, RFC Editor, <http://www.rfc-editor.org/rfc/rfc793.txt> (accessed August 6, 2014).
- Python. 2012. Python programming language – official website, <http://www.python.org/> (accessed August 6, 2014). Python Software Foundation.
- Sheik S, Stefanini F, Neftci E, Chicca E, and Indiveri G. 2011. Systematic configuration and automatic tuning of neuromorphic systems. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 873–876.
- Stefanini F, Neftci EO, Sheik S, and Indiveri G. 2014. PyNCS: a microkernel for high-level definition and configuration of neuromorphic electronic systems. *Front. Neuroinformat.* **8**(73), 1–14.
- Stevens WR, Fenner B and Rudoff AM. 2003 *Unix Network Programming*. The Sockets Networking API, vol. 1, 3rd edn. Addison-Wesley, Reading, MA.
- TIOBE. 2013. TIOBE programming community index, <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> (accessed August 6, 2014).
- Yuille AL and Geiger D. 2003. Winner-take-all networks. In: *The Handbook of Brain Theory and Neural Networks*, 2nd edn. MIT Press, Cambridge, MA, pp. 1228–1231.

15

Algorithmic Processing of Event Streams

```
@Description("Subsamples x and y addresses")
public class SubSampler extends EventFilter2D {
    /** Process the packet.
     * @param in the input packet
     * @return out the output packet
     */
    synchronized public EventPacket filterPacket(EventPacket in) {
        OutputEventIterator oi=out.outputIterator(); // get the iterator to return output events
        for(BasicEvent e:in){ // for each input event
            BasicEvent o=(BasicEvent)oi.nextOutput(); // get an unused output event
            o.copyFrom(e); // copy the input event to the output event
            o.x = o.x>>bits; // right shift the x and y addresses
            o.y = o.y>>bits;
        }
        return out; // return the output packet
    }
}
```

This chapter describes event-driven algorithmic processing of AE data streams on digital computers. Algorithms fall into categories such as noise-reduction filters, event labelers, and trackers. The data structures and software architectures are also discussed as well as requirements for software and hardware infrastructure.¹

15.1 Introduction

As AER sensors such as the retinas of Chapter 3 and the cochleas of Chapter 4 and their supporting hardware infrastructure have evolved, it has become clear that the most rapid way to take advantage of this new hardware is by developing algorithms for processing the

¹ Parts of this material stem from Delbrück (2008).

data from the devices on conventional computers. This approach allows rapid development and performance that scales at least as fast as Moore's law. Such an option is really the only tenable one for integration into practical systems as well, because conventional digital processors will continue to serve as a foundation for building integrated systems for the foreseeable future.

This approach is somewhat in contrast with past and many of the current developments of neuromorphic engineering, that has focused on complete neuromorphic systems like the ones described in Chapter 13, which eschew connections with programmable computers except to the extent of providing supporting infrastructure. But although there has been a lot of developments of such pure neuromorphic systems, they have proven difficult to configure and use. By treating event-based hardware more like standard computer peripherals, applications can be rapidly developed for practical purposes.

In the development of the applications, there is a need for a style of digital signal processing that starts with the AER events as input. These time-stamped digital addresses now are processed in order to achieve a desired aim. For instance, a robotic car that uses silicon retina output can be developed to drive safely along a road, by following the noisy and incomplete lane markings. Or a binaural AER cochlea can be used as an input sensor for a battery-powered wireless sensor that detects and classifies significant events such as the sound and direction of gunshots. These are just two potential examples, neither of which has yet been built, but they serve to illustrate that there are practical (and quite feasible) problems that can drive the commercial development of neuromorphic technology without demanding a complete remake of electronic computing.

In this 'event-driven' style of computation, each event's location and time-stamp are used in the order of arrival. Algorithms can take advantage of the capabilities of synchronous digital processors for high-speed iteration, branching logic operations, and moderate pipelining.

Over the last 10 years, there has been significant development using this approach, and this chapter presents instances of these algorithms and applications. The discussion starts with a description of the required software infrastructure, and then presents examples of existing algorithms. These examples are followed by some remarks on data structures and software architecture. Finally, the discussion comes back to the relationship between existing algorithms and conventional signal processing theory based on Nyquist sampling and the need for new theoretical developments in this area to support the data-driven style of signal processing that underlies the event-based approach.

Some examples of existing algorithms based on silicon retina output include commercial development of vehicle traffic and people counting using DVS silicon retinas (Litzenberger et al. 2006a; Schraml et al. 2010), fast visual robots such as a robotic goalie (Delbrück and Lichtsteiner 2007) and a pencil balancer (Conradt et al. 2009b), hydrodynamic and microscopic particle tracking (Drazen et al. 2011; Ni et al. 2012), low-level motion feature extraction (Benosman et al. 2012), low-level stereopsis (Rogister et al. 2012), and stereo-based gesture recognition (Lee et al. 2012). Examples of event-based audio processing include auditory localization (Finger and Liu 2011) and speaker identification (Li et al. 2012).

Based on these developments, it has become clear that methods for processing events have evolved into the following classes:

- *Filters* that clean up the input to reduce noise or redundancy.
- Low-level feature detectors that we will call *labelers*, that attach additional labels to the events which are intermediate interpretations of their meanings. For example, a silicon retina event can acquire interpretation, such as contour orientation or direction of motion.

Based on these extended types, global metrics such as image velocity are easily computed by integrating these labels.

- *Trackers* that detect, track, and potentially classify objects.
- *Cross-correlators* that cross-correlate event streams from different sources.

In order to filter and process event streams, some memory structures are needed. For some algorithms, particularly for filters and labelers, it is often the case that they use one or several topographic memory maps of event times. These maps store the last event time-stamps for each address.

Because events in a computer are just data objects, unlike the binary spikes of the nervous system, digital events can carry arbitrary payloads of additional information, which are called *annotations* here. Events start with precise timing and source address in the AER device. As they are processed by a pipeline of algorithms, extraneous events are discarded, and as events are labeled they can gain additional meaning. Event annotations can be attached by treating the event as a software object. This object is not the same as cell type in cortex. In cortical processing one usually considers that increased dimensionality is implemented by expanding the number of cell types. But because events as considered here can carry arbitrary payloads, algorithmic events are assigned increasing interpretation by larger annotations. Multiple interpretations can then be transported by multiple events instead of activity on multiple hardware units. For instance, a representation of orientation that is halfway between two principal directions can still be represented as near-simultaneous events, each one signifying a different and nearby orientation. In addition, this annotated event can carry scalar and vector values. For example, the ‘motion events’ produced by the algorithm described in Section 15.4.3 add information about the speed and vector direction of the computed optical flow.

15.2 Requirements for Software Infrastructure

One can imagine a future software infrastructure like the one shown in Figure 15.1. This software must handle multiple input streams from several neuromorphic sensors such as retinas and cochleas, and must process data from each stream in a pipeline. It must also combine streams to perform sensor fusion and finally produce output, either in the form of motor commands or analytical results. Ideally, the processing load is distributed efficiently over threads of execution and processing cores.

The organization of events in memory data structures is important for efficiency of processing and flexibility of software development. As an example, the architecture used in jAER (2007) illustrated in Figure 15.2 shows how events are bundled in buffers called packets. A packet is a reusable memory object that contains a list of event objects. These event objects are memory references to structures that contain the event with its annotations. A particular event-based filter or processor maintains its own reused output packet that holds the results. These packets are reused because the cost of object creation is typically hundreds of times more expensive than accessing existing objects. The packets are dynamically grown as necessary, although this expensive process only occurs a few times during program initialization. This way, heap memory use is low because the reused packets are rarely allocated and need not be garbage-collected, which is important particularly for real-time applications.

The event packets are analogous to frames, but are different. A frame of data represents a fixed point or range in time, and the contents of the frame consists of samples of the

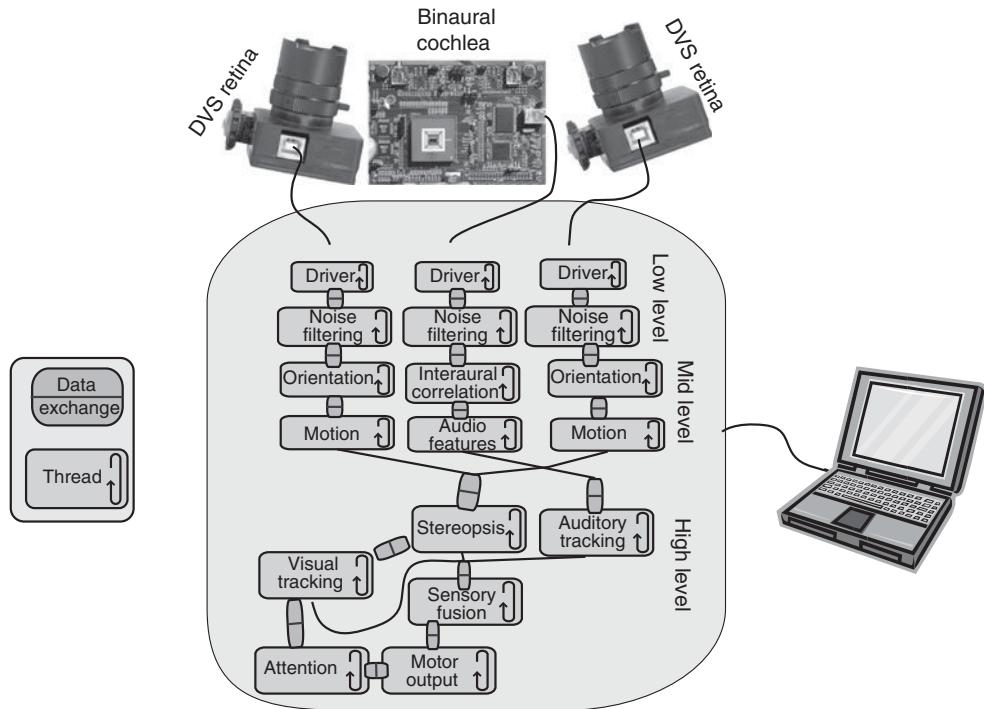


Figure 15.1 Possible software architecture for handling multiple AER input streams (here a stereo pair of silicon retinas and a binaural silicon cochlea) and processing the data to finally produce a motor output

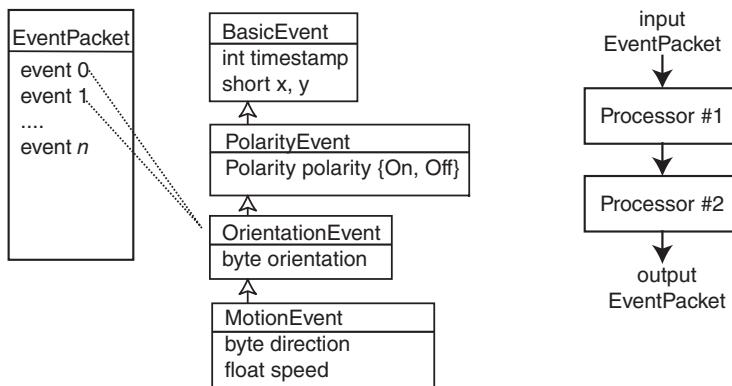


Figure 15.2 Structure of an event packet, holding events that are derived from a base class to contain more extended annotation. Packets are processed by a pipeline of processors, each operating on the output of a previous stage

analog input. By contrast, a packet can represent a variable amount of real time depending on the number of events and their timing, and the events in a packet will tend to carry more identical amounts of useful information than samples in a frame, as long as the sensor reduces redundancy in its input.

15.2.1 Processing Latency

Processing latency is important for real-time applications especially when results are fed back to a controller. Examples of these systems are robots and factory production lines. Latency is also important to minimize for human–machine interaction. In an embedded implementation where the processor processes each event directly from the AE device as it is received, latency is simply algorithmic. But more typically, there will be some type of buffer holding a packet of events between the AE device and the processor. Using a software architecture that processes events in packets, the latency is the time between the last events in successive packets, plus the processing time. For example, suppose a light turns on and the system must react to it. The system must fill a hardware buffer (or time out and send it only partially filled), transmit the events, and then process the packet. If the light turns on just after a previous packet has been transmitted, then the total latency is the time to fill the buffer (or some minimum time), plus the transmission time, plus the processing time, plus finally the time required to activate the actuator.

Hardware interfaces between AER sensors and computers can be built to ensure that these packets get delivered to the host with a minimum frequency, for example, 1 kHz. Then the maximum packet latency is 1 ms. But the latency can be much smaller if the event rate is higher. For example, the USB chip used with the DVS silicon retina from Chapter 3 has hardware FIFO buffers of 128 events. If the event rate is 1 MHz, then 128 events fill the FIFO in 128 μ s. Then the USB interface running at 480 Mbps requires about 10 μ s to transmit the data to the computer and the total latency to get the data into the computer is under 200 μ s. Compared with a 100 Hz camera, this represents a factor of 50 times smaller latency.

15.3 Embedded Implementations

When software algorithms are embedded in processors adjacent to AE devices, there are somewhat different considerations than when the software is implemented in a more general framework on a computer connected to the device by some kind of remote interface such as USB. In an embedded implementation, the processor, which is generally either a microcontroller (Conradt et al. 2009a) or a DSP (Litzenberger et al. 2006b), is directly connected to the AE device. In Litzenberger et al. (2006b) and Hofstatter et al. (2011), a hardware FIFO is used to buffer AEs before reaching the processor, allowing more efficient processing of buffers of events. A DSP was also used in the VISE system reported by Grenet et al. (2005). In embedded implementations, it is generally best to inline all processing stages for events as much as possible. This avoids function calling overhead and it is often possible to avoid the need to time-stamp events, because each event is processed in real time as it is received.

On cheaper and lower power processors, only fixed point computation is available, and in many chips only multiply and not divide is offered. For instance, Conradt et al. (2009a) reported an implementation of a pencil balancing robot where each DVS sensor output was directly processed by a fixed-point, 32-bit microcontroller burning about 100 mW. Here, the computation of pencil angle and position was performed entirely in fixed-point arithmetic. A single

divide operation was required for every actual update of the balancer hand at a rate of 500 Hz and this divide was split up into pieces, so that it would not result in large gaps in processing events. The update of the LCD panel showing the DVS retina output was cleverly scheduled, so that only two pixels of the LCD screen were updated on the reception of each event—one at the location of the event and the other scanning over the LCD pixel array and decaying the value there. That way, a 2D histogram of event activity could be efficiently maintained without requiring large periods of time during which event processing would be interrupted.

Another highly developed example of an embedded implementation is reported in Schraml et al. (2010). This system uses a pair of DVS retinas and computes stereo correspondence on an FPGA in order to detect ‘falling people’ events, which are important for elderly care. The approach taken is like the one reported in Benosman et al. (2012) where conventional machine vision methods are applied to short-time buffers of accumulated events.

15.4 Examples of Algorithms

This section presents examples of some existing methods for processing events, all from jAER (2007), and mostly for the DVS discussed in Chapter 3. It starts with noise filtering and then continues with low-level visual feature extraction, followed by visual object tracking, followed by examples of audio processing using the AER-EAR silicon cochlea discussed in Chapter 4.

15.4.1 Noise Reduction Filters

It may be beneficial to preprocess data in some way by either transforming or discarding events. As trivial instances, it might be necessary to reduce the size of the address space (e.g., from 128×128 to 64×64) or transform an image by rotating it. These two operations consist of simply right shifting the x and y addresses (subsampling) or multiplying each (x, y) address by a rotation matrix.

Preprocessing can also be very beneficial to remove ‘noise’ addresses. As an example of noise filtering, we will describe an algorithm (open-sourced as `BackgroundActivityFilter` in the jAER project) that removes uncorrelated background activity from the DVS sensor described in Chapter 3. These events can arise from thermal noise or junction leakage currents acting on switches connected to floating nodes. The filter only passes events that are supported by recent nearby (in space) events. Background activity is uncorrelated and is largely filtered away, while events that are generated by the world, say by a moving object seen by a silicon retina, even if they are only single pixel in size, mostly pass through. This filter uses two maps of event time-stamps to store its state, that is, because the sensor has 128×128 pixels, each with ON and OFF output events, two arrays of 128×128 pixels containing integer time-stamp values are used to store the time-stamps. This filter has a single parameter dT that specifies the support time for which an event will be passed, meaning the maximum time that can pass between this event and a nearby past event to allow this event to pass the filter. The steps of the algorithm for each event in a packet are as follows:

1. Store the event’s time-stamp in all neighboring addresses in the time-stamp memory—for instance, the 8 pixel addresses surrounding the event’s address, overwriting the previous values.

2. Check if the event's time-stamp is within dT of the previous value written to the time-stamp map at this event's address. If a previous event has occurred recently, pass the event to the output, otherwise discard it.

Because branching operations are potentially expensive in modern CPU architectures, two optimizations are used here. First, this implementation avoids the time-stamp difference check on all the neighbors simply storing an event's time-stamp in all neighbors. Then only a single conditional branch is necessary following the iteration to write the time-stamps to the map. Second, the time-stamp maps are allocated so that they are larger than the input address space by at least the neighborhood distance. Then during iteration to write the time-stamp to the event's neighboring pixels, there is no need to check array bounds.

Typical results of the background activity filter operating on a DVS retina output are shown in Figure 15.3. The input data are from a walking fruit fly observed from above. The background activity filter removes almost all of the uncorrelated background activity while leaving the spatiotemporally correlated events from the fly. In the jAER implementation running on a Core-i7 870 3 GHz PC, each event is processed in about 100 ns.

Other examples of useful filters are a ‘refractory filter’ that limits event rates for individual addresses, an ‘x-y-type filter’ that only passes events from a particular region of the address space, and a ‘depressing synapse filter’ that passes events with decreasing probability as the average event rate at an address increases. This filter tends to equalize activity across addresses

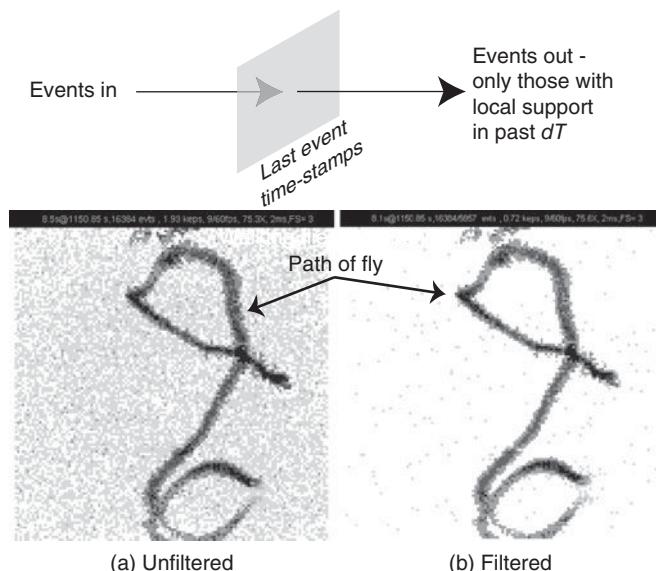


Figure 15.3 Background noise filter example. The input is from a DVS retina viewing a scene with a single walking fruit fly over a 9 s period. The data are rendered as a 2D histogram of collected events, like an image of the collected event addresses. (a) Without background filtering, the uncorrelated background activity rate is about 3 kHz and is visible as the gray speckle. The path of the fly is visible as darker pixels. (b) Using the filter, the background rate is reduced to about 50 Hz, a factor of 60 times smaller, while the activity from the fly is unaffected. The gray scale is the same for both plots

and can limit input from redundant sources, say flickering lights from a silicon retina or spectral bands from a silicon cochlea.

15.4.2 Time-Stamp Maps and Subsampling by Bit-Shifting Addresses

In the filtering example just presented, the incoming event times are written to a time-stamp map. This map is a 2D array of the most recent event time-stamps at each address. This map is like a picture of the event times. A moving edge will create a landscape in the time-stamp map that looks like a gradual slope up to a ridge, followed by a sharp cliff that falls down to ancient events that represents retina output from some prior edge, which is probably not relevant anymore. Filtering operations can inspect the time-stamp map at the most recent events around the source address.

When filtering events out or labeling them (as discussed in Section 15.4.3), one very useful operation is subsampling by bit-shifting. This operation has the effect of increasing the area of the time-stamp map that can be inspected by an event filter without increasing the cost. Right-shifting the x and y addresses by n bits will end up filling each element of a time-stamp map with the most recent time-stamp in blocks of $2^n \times 2^n$ input addresses. While filtering a packet, the same number of events need to be processed, but now operations on the time-stamp map that iterate over neighborhoods of the incoming event address effectively cover an area of the input address space that is $(2^n)^2$ times larger. The effect is the same as if the receptive field area is increased by this amount, but at no increase of the cost of iteration over larger neighborhoods.

15.4.3 Event Labelers as Low-Level Feature Detectors

Once noise or redundancy reduction in the event stream is achieved by filtering out events, now the next step could be to detect features, for instance, the edge orientation or the direction and speed of motion of edges. (These algorithms are `SimpleOrientationFilter` and `DirectionSelectiveFilter` in the jAER project.) These ‘event labeling’ algorithms result in output streams of new types of events, which are now labeled with the additional annotation detected during processing.

An example of this type of feature detector is a ‘motion labeler’ that measures local normal optical flow. The results of this algorithm are illustrated in Figure 15.4. The steps of motion measurement consist first of determining the edge orientation and then determining from what direction this edge has moved and how fast. By this method only the so-called ‘normal flow’ can be determined, because motion parallel to the edge cannot be determined.

The method of orientation labeling was inspired by the famous Hubel and Wiesel arrangement of center-surround thalamic receptive fields that are lined up to produce an orientation-selective cortical simple-cell receptive field (Hubel and Wiesel 1962). One can think of the simple cell as a coincidence detector for thalamic input. A moving edge will tend to produce events that are correlated more closely in time with nearby events from the same edge. The orientation labeler detects these coincident events and determines the edge orientation by the following steps (see also Figure 15.4):

1. Store the event time in the time-stamp map at the event address (x,y) . An example of input events are shown in Figure 15.4a. There is one time-stamp map for each type of input event,

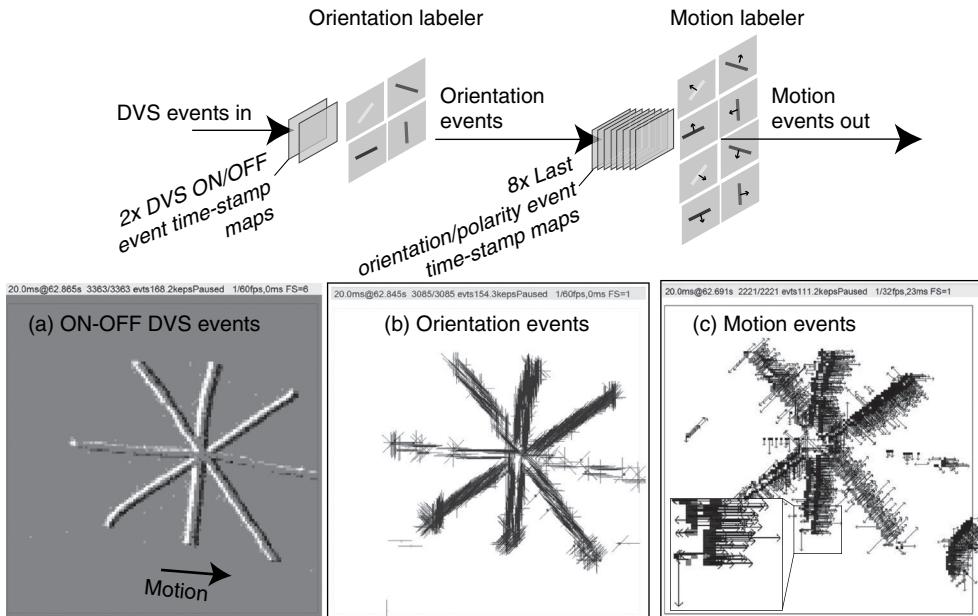


Figure 15.4 Orientation features extraction followed by local normal flow events. (a) The input is 3000 events covering 20 ms of DVS activity from the cross pattern which is moving to the right. (b) The orientation events are shown by line segments along the detected orientation. The length of the segments indicates the receptive field size. (c) The motion events are shown by vector arrows. They indicate the normal flow velocity vector of the orientation events. The inset shows a close-up of some motion vectors

for example, two maps for the DVS sensor ON and OFF event polarities. There is a separate map for each retina polarity so that an ON event can be correlated only with previous ON events and an OFF event can be correlated only with past OFF events.

2. For each orientation, measure the correlation time in the area of the receptive field. Correlation is computed as the sum of absolute time-stamp differences (**SATD**) between the input event and the stored time-stamps in the corresponding (ON or OFF) time-stamp map. Typically, the receptive field size is 1×5 pixels centered on the event address, where in that case four time-stamp differences must be computed. The array offsets into the memory of past event times are precomputed for each orientation receptive field.
3. Output an event labeled with the orientation of the best correlation result from step 2, but only if it passes an SATD threshold test that rejects events with large SATD, representing poor correlation. The results of this extraction of edge orientation events are shown in Figure 15.4b.

The next step of the motion algorithm is to use these ‘orientation events’ to determine normal optical flow (again see Figure 15.4). The steps of this algorithm are as follows:

1. Just as for the orientation labeler, first the input events (the orientation events) overwrite previous event times in time-stamp maps. One map is used to store each input event type;

- for example, for four orientation types and two input event polarities, eight maps would be used.
2. For each orientation event, a search over the corresponding time-stamp map is performed in a direction orthogonal to the edge to determine the likely direction of motion of the edge. The SATD is computed for each of two possible directions of motion. The direction with smaller SATD is the direction from which the edge moved. During computation of SATD, only events with a time-stamp difference smaller than a parameter value are included in the SATD. This way, old events that do not belong to the current edge are not counted.
 3. Output a ‘motion direction event’ labeled with one of the eight possible directions of motion and with a speed value that is computed from the SATD. The speed is computed by computing the time of flight of the current orientation event to each prior orientation event in the direction opposite the direction of motion in the range of the receptive field size, and then taking the average. Typically the range is five pixels, but as for the orientation labeler, this size is adjustable. During this computation of the average time of flight, outliers are rejected by only counting times that are within a time limit. An example of the motion labeler output is shown in Figure 15.4c. The small and thin vectors show the local normal optical flow, while the large vectors show the lowpass-filtered average translation, expansion, and rotational values over the recent past.

The motion labeler algorithm described above was developed in 2005 as part of the jAER project. Benosman et al. (2012) reported a different approach that is interesting because rather than searching over maps of past events, it uses a gradient-based approach based on a straightforward translation of the popular and effective (Lucas and Kanade 1981) optical flow algorithm operating on accumulated-event image patches of size $n \times n$ pixels (where $n = 5$ was used) computed over a short-time window of a reported 50 μs of DVS activity. This way, the algorithm can determine with a linear regression both scalar orientation and speed to result in a local normal flow vector. By using this approach, spatial and temporal gradients are computed at a high effective sample rate while computational cost was reduced by a factor of 25 compared to the frame-based Lucas–Kanade method. This method results in more accurate optical flow than the motion labeler, but it costs about 20 times as much because a set of n^2 simultaneous linear equations must be solved for each motion vector. Processing each DVS event on a fast year-2012 PC requires about 7 μs of CPU time compared with 350 ns for the motion labeler.

15.4.4 Visual Trackers

The task of object tracking is well-suited to activity-driven event-based systems, because moving objects generate spatiotemporal energy that generates events, and these events can then be used for tracking the objects.

Cluster Tracker

As an example of a tracking algorithm we will discuss a relatively simple tracker called the cluster tracker (`RectangularClusterTracker` in the jAER project). The basic cluster tracker tracks the motion of multiple moving compact objects (Delbrück and Lichtsteiner 2007; Litzenberger et al. 2006b), for instance, particles in a 2D fluid, balls on a table, or cars on a highway. It does

this by using a model of an object as a spatially connected and compact source of events. As the objects move, they generate events. These events are used to move the clusters. Clusters are spawned when events are detected at a place where there are no clusters, and clusters are pruned away after they receive insufficient support.

The cluster has a size that is fixed or variable depending on the application, and that can also be a function of location in the image. In some scenarios such as looking down from a highway overpass, the class of objects is rather small, consisting of vehicles, and these can all be clumped into a single restricted size range. This size in the image plane is a function of height in the image because the images of vehicles near the horizon are small and those of ones passing under the camera are maximum size. Additionally, the vehicles near the horizon all appear about the same size because they are viewed head-on. In other scenarios, all the objects are nearly the same size. Such is the case when looking at marker particles in a fluid experiment or falling raindrops.

There are several advantages of the cluster tracker compared with conventional frame-based trackers. First, there is no correspondence problem because there are no frames, and events update clusters asynchronously. Second, only pixels that generate events need to be processed and the cost of this processing is dominated by the search for the nearest existing cluster, which is typically a cheap operation because there are few clusters. And lastly, the only memory required is for cluster locations and other statistics, typically about a hundred bytes per cluster.

The steps for the cluster tracker are outlined as follows. It consists firstly of an iteration over each event in the packet, and secondly of global updates during the packet iteration, at a fixed interval of time.

A cluster is described by a number of statistics, including position, velocity, radius, aspect ratio, angle, and event rate. For example, the event rate statistic describes the average rate of events received by the cluster over the past τ_{rate} milliseconds, that is, it is a lowpass-filtered instantaneous event rate.

First, for each event in the packet:

1. Find the nearest existing cluster by iterating over all clusters and computing the minimum distance between the cluster center and the event location.
2. If the event is within the cluster radius of the center of the cluster, add the event to the cluster by pushing the cluster a bit toward the event and updating the last event time of the cluster. Before the cluster is updated, it is first translated by using its estimated velocity and the time of the event. This way, the cluster has an ‘inertia,’ and the events serve to update the velocity of the cluster rather than its position. Subsequently, the distance the cluster is moved by the event is determined by a ‘mixing factor’ parameter that sets how much the location of the event affects the cluster. If the mixing factor is large, then clusters update more quickly but their movement is noisier. A smaller mixing factor causes smoother movement but rapid movements of the object can cause tracking to be lost. If the cluster is allowed to vary its size, aspect ratio, or angle, then update these parameters as well. For instance, if size variation is allowed, then events far from the cluster center make the cluster grow, while events near the center make it shrink. The cluster event rate statistic and the velocity estimate are also updated using lowpass filters.
3. If the event is not inside any cluster, seed a new cluster if there are spare unused clusters to allocate. A cluster is not marked as visible until it receives a minimum rate of events. The maximum number of allowed clusters is set by the user to reflect their understanding of the application.

Second, at periodic update intervals (e.g., 1 ms, which is a configurable option), the following cluster tracker update steps are performed. The decision on when to do an update is determined during iteration over the event packet. Each event's time-stamp is checked to see if it is larger than the next update time. If it is larger, then the update steps as outlined next are performed, and after the update, the next update time is incremented by the update interval.

The steps for the periodic cluster tracker update are as follows:

1. Iterate overall clusters, pruning out those clusters that have not received sufficient support. A cluster is pruned if its event rate has dropped below a threshold value.
2. Iterate overall clusters to merge clusters that overlap. This merging operation is necessary because new clusters can be formed when an object increases in size or changes aspect ratio. This merge iteration continues until there are no more clusters to merge. Merged clusters usually (depending on tracker options) take on the statistics of the oldest cluster, because this one presumably has the longest history of tracking the object.

An example of an application of this kind of tracker is the robotic goalie in Delbrück and Lichtsteiner (2007), which was later extended to include self-calibration of its arm. Figure 15.5 shows the setup and a screen shot taken during operation of the goalie. The goal in developing this robot was to demonstrate fast, cheap tracking, and quick reaction time. As shown in Figure 15.5a, a person would try to shoot balls into the goal and the goalie would block the most-threatening ball, which was determined to be first crossing the goal line. A fast shot would cover the 1 m distance to the goal in under 100 ms. To achieve this, the robot needed to track the balls to determine their positions and velocities, so that it could move the arm under open-loop control to the correct position to block the ball. During idle periods the robot also moved its arm to random positions while tracking it, in order to calibrate the mapping from servo position to visual space. That way, the robot could set the arm to a desired visual location in the image. Figure 15.5b is a screenshot captured during operation. Three balls were being tracked and one of them, the attacking ball, was circled to indicate it was the one being blocked. Each ball also has an attached velocity vector estimate and there are a number

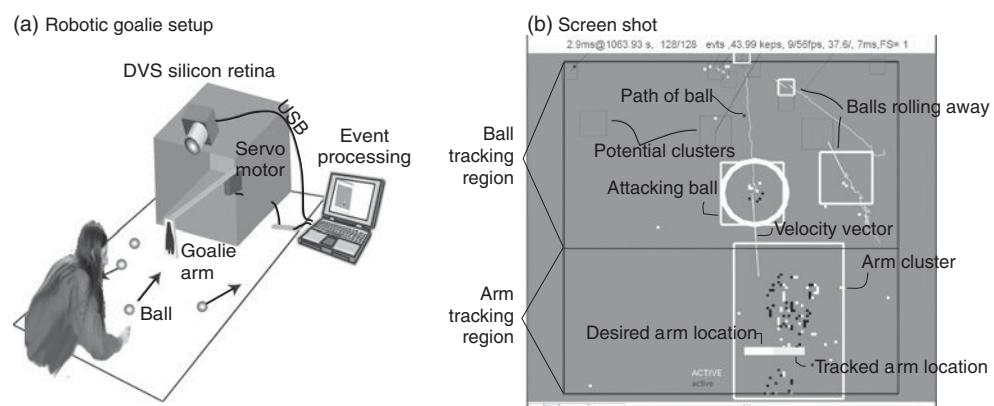


Figure 15.5 The robotic goalie. (a) The setup shows a DVS silicon retina and the 1-axis goalie arm. (b) A screenshot shows tracking of multiple balls and the goalie arm

of other potential ball clusters that did not receive sufficient support to be classified as balls. The scene was statically segmented into a ball tracking region and an arm tracking region. In the arm region, the arm is tracked by a larger tracker cluster and in the ball region the tracker was configured so that the cluster size matched the ball size based on perspective. During that snapshot of 2.9 ms of activity there were 128 DVS events at a rate of 44 keps. The event rate was fairly linear with the ball speed, so slow moving balls generated fewer events and fast moving ones more events. Because the tracker was event-driven, fast moving balls were tracked with the same performance as slow moving ones. The larger the goalie hand size, the easier it is to block the balls. Using a goalie hand size of 1.5 times the ball diameter, the goalie was measured to block 95% of the balls. The goalie ran on a 2006-era laptop processor at a processor load of under 4%. The reaction time—defined as the time interval from starting to track a ball to a change in the servo control signal—was measured to be less than 3 ms.

More Tracking Applications of the DVS

Other applications of tracking have customized tracking to deal with specific scenarios. For example, Lee et al. (2012) reported a stereopsis-based hand tracker using DVS retinas for gesture recognition. Here the average stereo disparity was determined by cross-correlation of 1D histograms of activity in two DVS sensors. Next, events from the DVS sensors were transformed to bring the peak activities into registration. (This way, the dominant object, which was the moving hand, was focused on a single region, while background movements of the person were suppressed.) Next, a tracker that cooperatively detected connected regions of activity found the moving hand area. This tracker was based on laterally coupled I&F neurons. Neurons were connected to the DVS input and to their neighbors with excitatory connections, so that a connected region of activity was preferred over disconnected regions. The median location of activity in the largest connected region of active I&F neurons was used as the hand location.

In a second example, the pencil balancer robot reported by Conradt et al. (2009b) used an interesting tracker based on a continuous Hough transform. Each retina event was considered to draw a ridge of activity in the Hough transform space (Ballard 1981) of pencil angle and base position. An efficient algebraic transform allowed update of the continuous pencil estimate in Hough space without requiring the usual awkward quantization and peak-finding of the conventional Hough transform.

In a third example, the hydrodynamic particle tracker reported in Drazen et al. (2011) used a method based on the cluster tracker described above to track the fluid flow markers, but extra processing steps were applied to deal with particles with crossing trajectories.

And in the fourth example, Ni et al. (2012) applied a discretized circular Hough transform for microparticle tracking, in a manner quite similar to the CAVIAR hardware convolution in Serrano-Gotarredona et al. (2009). This tracker was not sufficiently accurate on its own, so it was followed by computation of the centroid of recent event activity around the tracker location to increase accuracy.

Finally, Bolopion et al. (2012) reported using a DVS together with a CMOS camera to track a microgripper and the particle being manipulated to provide high-speed haptic feedback for gripping. It used an iterative closest point algorithm based on Besl and McKay (1992) to minimize Euclidean distance between events in the last 10 ms and the gripper shape model by rotation and translation of the gripper model.

Action and Shape Recognition with the DVS

Some work has been performed on classification of events such as detecting people falling down, which is important for elderly care. Fu et al. (2008) reported a simple fall detector based on classifying DVS time-space event histograms from a single DVS looking head-on or sideways at a person. Later, Belbachir et al. (2012) demonstrated an impressive fall detector that used a stereo pair of DVS sensors and that allowed an overhead view of the person falling.

This same group developed a full-custom embedded DSP based on a MIPS core with native AE time-stamping interfaces (Hofstatter et al. 2009) that was used to do high-speed classification of flat shapes using single and dual-line DVS sensor outputs (Belbachir et al. 2007; Hofstatter et al. 2011).

Applications of Event-Based Vision Sensors in Intelligent Transportation Systems

The VISe vision sensor described in Section 3.5.5 was applied in lane detection and lane departure warning in a highly developed manner as reported in Grenet (2007). This report, although not available online, is worth reading for its completeness and the systematic design of the system, including the implementation of several different tracking states. The system switches between bootstrapping, search and tracking states and uses a Kalman filter during the tracking phase. It also detected the difference between continuous and dashed lane markings.

A major application of the DVS has been in intelligent transportation systems, in particular in monitoring traffic on highways. Specialized algorithms have been developed that use an embedded commercial DSP that processes DVS output (Litzenberger et al. 2006b). Using this system Litzenberger et al. (2006a) reported vehicle speed measurement and Litzenberger et al. (2007) reported car counting. In the latest work, this group also was able to classify cars versus trucks on highways (Gritsch et al. 2009) during nighttime, based on headlight separation.

15.4.5 Event-Based Audio Processing

The event-based approach can be particularly beneficial in applications requiring precise timing. One example of this is from binaural auditory processing, where cross correlation is used to determine the interaural time difference (ITD), ultimately to determine the azimuthal directions of sound sources. Processing of cochlea spikes from the AEREAR2 cochlea for localization is described in Section 4.3.2 of Chapter 4. An event-based method reported in Finger and Liu (2011) for estimating ITD (ITDFilter in the jAER project) shows how rolling buffers of events from two sources can be cross-correlated. In this method, the ITDs between events from one ear (one cochlea channel from one ear) are computed to previous events from the other ear (the corresponding cochlea channel from the other ear). Only ITDs up to some limit, for example ± 1 ms, are considered. Each of these ITDs is weighted (as described below) and stored in a decaying histogram of ITD values. The locations of the peaks in the histogram signify the ITDs of sound sources.

It is harder to localize sound in reverberant spaces, because the sound can take multiple paths to reach a listener. However, sound onsets can be used to disambiguate the source direction because the onset of a sound can be used to determine the direct path. A special feature used in the ITD algorithm was to multiply the value added to the histogram by the temporal duration of silence (no events) prior to the current event. This way, sound onsets are weighted more strongly. Using this feature improved the performance dramatically.

Finger and Liu (2011) reported that the system could localize the direction of a sound source in a reverberant room to 10° of precision using a microphone separation of about 15 cm, representing a precision of about 60 μs in estimating the ITD. Using the event-based approach to cross correlation reduced computational cost by a factor of about 40 compared with conventional cross correlation of regular sounds samples at a sample rate that achieved the same performance in the room (Liu et al. 2013). The event-based approach results in a latency of sound source localization for human speech of under 200 ms.

15.5 Discussion

An event-driven approach to signal processing based on neuromorphic AER sensors and actuators has the potential of realization of small, fast, low-power embedded sensory-motor processing systems that are beyond the reach of traditional approaches under the constraints of power, memory, and processor cost. However, in order to bring this field to the same level of competence as the conventional Nyquist-based signal processing, there is clearly need for theoretical developments. For instance, it is not clear how very basic elements like analog filters can be designed using an event-based approach. In conventional signal processing, the assumption of linear time invariant systems, regular samples, and the use of the z-transform allow the systematic construction of signal processing pipelines as described in every standard signal processing textbook. No such methodology exists for event-based processing. Although some authors write a kind of formal description of an event-based processing algorithm as a set of equations, so far these equations are only descriptions of an iterator with branching, and they cannot be manipulated to allow derivation of different forms or combinations in the same way that are allowed by regular algebras. And to bring the event-based approach together with rapid advances in machine learning, there is clearly a need to bring learning into the event-based approach. These and other developments will make this area interesting over the next few years. One intriguing possibility is that the most powerful algorithms turn out to be the ones that run best on the kinds of large-scale neuromorphic hardware systems described in Chapter 16.

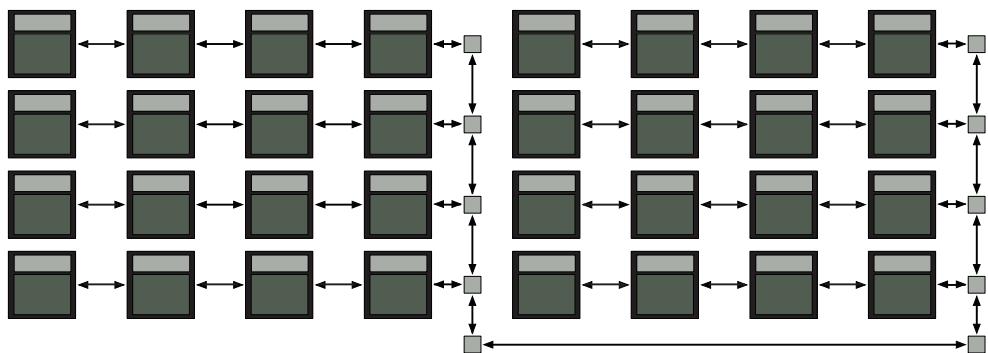
References

- Ballard DH. 1981. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recogn.* **13**(2), 111–122.
- Belbachir AN, Litzenberger M, Posch C, and Schon P. 2007. Real-time vision using a smart sensor system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1968–1973.
- Belbachir AN, Litzenberger M, Schraml S, Hofstatter M, Bauer D, Schon P, Humenberger M, Sulzbachner C, Lunden T, and Merne M. 2012. CARE: a dynamic stereo vision sensor system for fall detection. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 731–734.
- Benosman R, Ieng SH, Clercq C, Bartolozzi C, and Srinivasan M. 2012. Asynchronous frameless event-based optical flow. *Neural Netw.* **27**, 32–37.
- Besl PJ and McKay HD. 1992. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(2), 239–256.
- Bolopion A, Ni Z, Agnus J, Benosman R, and Régnier S. 2012. Stable haptic feedback based on a dynamic vision sensor for microrobotics. *Proc. 2012 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 3203–3208.
- Conradt J, Berner R, Cook M, and Delbrück T. 2009a. An embedded AER dynamic vision sensor for low-latency pole balancing. *Proc. 12th IEEE Int. Conf. Computer Vision Workshops (ICCV)*, pp. 780–785.
- Conradt J, Cook M, Berner R, Lichtsteiner P, Douglas RJ, and Delbrück T. 2009b. A pencil balancing robot using a pair of AER dynamic vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 781–784.

- Delbrück T. 2008. Frame-free dynamic digital vision. *Proc. Int. Symp. Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, University of Tokyo, March 6–7, pp. 21–26.
- Delbrück T and Lichtsteiner P. 2007. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 845–848.
- Drazen D, Lichtsteiner P, Häfliger P, Delbrück T, and Jensen A. 2011. Toward real-time particle tracking using an event-based dynamic vision sensor. *Exp. Fluids* **51**(55), 1465–1469.
- Finger H and Liu SC. 2011. Estimating the location of a sound source with a spike-timing localization algorithm. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 2461–2464.
- Fu Z, Delbrück T, Lichtsteiner P, and Culurciello E. 2008. An address-event fall detector for assisted living applications. *IEEE Trans. Biomed. Circuits Syst.* **2**(2), 88–96.
- Grenet E. 2007. Embedded high dynamic range vision system for real-time driving assistance. *Technische Akademie Heilbronn e. V., 2. Fachforum Kraftfahrzeugtechnik*, pp. 1–16.
- Grenet E, Gyger S, Heim P, Heitger F, Kaess F, Nussbaum P, and Ruedi PF. 2005. High dynamic range vision sensor for automotive applications. *European Workshop on Photonics in the Automobile*, pp. 246–253.
- Gritsch G, Donath N, Kohn B, and Litzenberger M. 2009. Night-time vehicle classification with an embedded vision system. *Proc. 12th IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, pp. 1–6.
- Hofstatter M, Schon P, and Posch C. 2009. An integrated 20-bit 33/5M events/s AER sensor interface with 10 ns time-stamping and hardware-accelerated event pre-processing. *Proc. IEEE Biomed. Circuits Syst. Conf. (BIOCAS)*, pp. 257–260.
- Hofstatter M, Litzenberger M, Matolin D, and Posch C. 2011. Hardware-accelerated address-event processing for high-speed visual object recognition. *Proc. 18th IEEE Int. Conf. Electr. Circuits Syst. (ICECS)*, pp. 89–92.
- Hubel DH and Wiesel TN. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* **160**(1), 106–154.
- jaER. 2007. jaER Open Source Project, <http://jaerproject.org> (accessed August 6, 2014).
- Lee J, Delbrück T, Park PKJ, Pfeiffer M, Shin CW, Ryu H, and Kang BC. 2012. Live demonstration: gesture-based remote control using stereo pair of dynamic vision sensors. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 741–745.
- Li CH, Delbrück T, and Liu SC. 2012. Real-time speaker identification using the AEREAR2 event-based silicon cochlea. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1159–1162.
- Litzenberger M, Kohn B, Belbachir AN, Donath N, Gritsch G, Garn H, Posch C, and Schraml S. 2006a. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. *Proc. 2006 IEEE Intell. Transp. Syst. Conf. (ITSC)*, pp. 653–658.
- Litzenberger M, Posch C, Bauer D, Belbachir AN, Schon P, Kohn B, and Garn H. 2006b. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor. *Proc. 2006 IEEE 12th Digital Signal Processing Workshop, and the 4th Signal Processing Education Workshop*, pp. 173–178.
- Litzenberger M, Kohn B, Gritsch G, Donath N, Posch C, Belbachir NA, and Garn H. 2007. Vehicle counting with an embedded traffic data system using an optical transient sensor. *Proc. 2007 IEEE Intell. Transp. Syst. Conf. (ITSC)*, pp. 36–40.
- Liu SC, van Schaik A, Minch B, and Delbrück T. 2013. Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output. *IEEE Trans. Biomed. Circuits Syst.*, pp. 1–12.
- Lucas BD and Kanade T. 1981. An iterative image registration technique with an application to stereo vision. *Proc. 7th Int. Joint Conf. Artificial Intell. (IJCAI)*, pp. 674–679.
- Ni Z, Pacoret C, Benosman R, Ieng S, and Regnier S. 2012. Asynchronous event-based high speed vision for microparticle tracking. *J. Microscopy* **245**(3), 236–244.
- Rogister P, Benosman R, Leng S, Lichtsteiner P, and Delbrück T. 2012. Asynchronous event-based binocular stereo matching. *IEEE Trans. Neural Netw. Learning Syst.* **23**(2), 347–353.
- Schraml S, Belbachir AN, Milosevic N, and Schön P. 2010. Dynamic stereo vision system for real-time tracking. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1409–1412.
- Serrano-Gotarredona R, Oster M, Lichtsteiner P, Linares-Barranco A, Paz-Vicente R, Gomez-Rodriguez F, Camunas-Mesa L, Berner R, Rivas M, Delbrück T, Liu SC, Douglas R, Häfliger P, Jimenez-Moreno G, Civit A, Serrano-Gotarredona T, Acosta-Jimenez A, and Linares-Barranco B. 2009. CAVIAR: a 45K-neuron, 5M-synapse, 12G-connects/sec AER hardware sensory-processing-learning-actuating system for high speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* **20**(9), 1417–1438.

16

Towards Large-Scale Neuromorphic Systems



This chapter describes four example neuromorphic systems: SpiNNaker, HiAER, Neurogrid and FACETS. These systems combine subsets of the principles outlined in previous chapters to build a large-scale hardware platform for neuromorphic system engineering. Each of them represents a different point in the design space, and so it is instructive to examine the choices made in each system.

16.1 Introduction

Multichip AER can be used to construct neuromorphic systems that can scale up to millions of neurons. Part of the reason for this scalability is the evidence from biology – since biological systems have billions of neurons, the biological computational principles and communication structures must inherently be such that they can scale to systems of that size. Since

neuromorphic systems emulate biological systems, we can assume that emulating biologically plausible networks is a task that will scale with the size of the network.

16.2 Large-Scale System Examples

This chapter discusses four case studies in implementing large-scale neuromorphic systems: SpiNNaker, a system from the University of Manchester; HiAER, a system from the University of California at San Diego; Neurogrid, a system designed at Stanford University; and FACETS, an EU project led by the University of Heidelberg. The four systems take very different approaches to the problem of implementing neurons, ranging from using generic digital microprocessors to wafer-scale integrated custom analog electronics. Each system uses a different approach to the design of the communication network for spikes, as well as the implementation of synapses.

16.2.1 Spiking Neural Network Architecture

Of current developments, the SpiNNaker project at the University of Manchester has taken the most ‘general-purpose’ approach to the design of large-scale neuromorphic systems.

System Architecture

The core hardware element for the neuromorphic system is a custom-designed ASIC called the SpiNNaker chip. This chip includes 18 ARM processor nodes (the ARM968 core available from ARM Ltd., one of the project’s industrial partners), and a specially designed router for communication between SpiNNaker chips. One of the ARM968 cores is designated as the *Monitor Processor*, and is responsible for system management tasks. Sixteen of the other cores are used for neuromorphic computation, while the extra core is a spare and is available to improve the manufacturing yield. Each SpiNNaker die is mated with a 128 MB SDRAM die, and the two dice together are packaged as a single chip (Furber et al. 2013).

Each chip can communicate with six nearest neighbors. Each chip participates in a horizontal ring (utilizing two links), a vertical ring (utilizing two links), and a diagonal ring (utilizing two links). This can also be viewed as a two-dimensional torus network (the horizontal and vertical links) with additional diagonal links. The overall network topology is shown in Figure 16.1. The overall system is envisioned to contain over a million ARM cores.

Neurons and Synapses

The low-power ARM cores are used to implement all the neuronal and synaptic computation in software. This provides maximum flexibility, since the precise details such as the neuron equation, governing dynamics, the amount of biological detail present in the neuron and synaptic model, and so on are all under user control since the core compute element in the system is a general-purpose microprocessor.

Each ARM core is fast enough to be able to model $\sim 10^3$ point neurons, where each neuron has $\sim 10^3$ synapses that are modeled as programmable weights. A million core system would

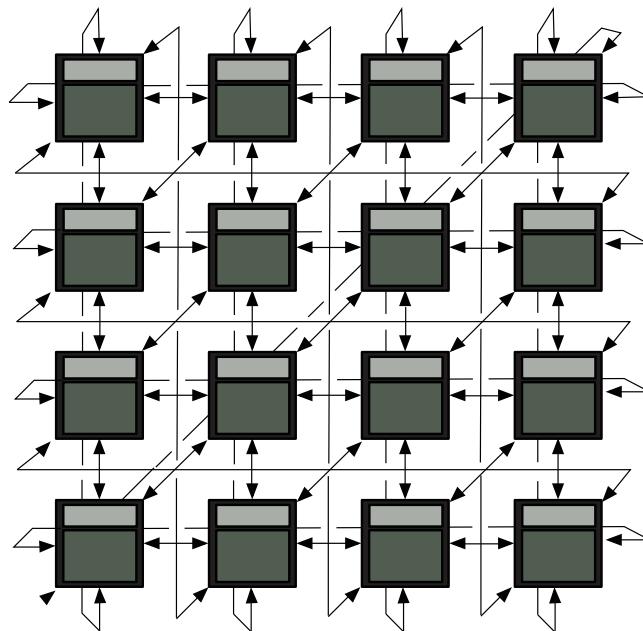


Figure 16.1 Chip-to-chip communication topology in the SpiNNaker system, showing a 16 node system. Each node in the system includes neuron resources (shown in dark gray) and routing resources (shown in light gray), and the arrows show communication links between SpiNNaker chips

be able to implement $\sim 10^9$ neurons and $\sim 10^{12}$ synapses, with the entire system operating in (biological) real time. Since the computation for a neuron is dominated by synaptic modeling, a more accurate way to view the computation power per core is to say that each core can model $\sim 10^6$ synapses, which corresponds to 10^3 neurons per core on average.

Communication

The communication network is responsible for delivering spikes to their appropriate destinations. Since a neuron has $\sim 10^3$ synaptic inputs on average, the average output fan-out for a spike generated by a neuron is also $\sim 10^3$. SpiNNaker has a custom-designed communication architecture for spike delivery that handles both routing and fan-out.

When a neuron produces an output spike, it is given a source routing key and is delivered to the communication fabric for delivery to a set of destination neurons. Instead of a central routing table, each SpiNNaker router contains information that is sufficient to make local decisions per spike. When a spike arrives on one of the six incoming ports, the source key is matched against a 1024-entry ternary content-addressable memory (TCAM) with 32 bits (same as the source key) per entry. A TCAM entry specifies either a 0, 1, or X value for each bit, and a source key matches an entry if it agrees with the entry for every not-X location. If there is a match, then a 24-bit-long vector corresponding to the matched entry is retrieved. The bits of this vector represent each of the on-chip ARM cores (18 bits) and the six output ports

of the communication network (6 bits). Routing is straightforward: the spike is propagated to all the locations for which the corresponding bit is set. If there is no match in the TCAM, the default route is a ‘straight line’ – that is a packet arriving from the left would be propagated to the right, and so on for all six potential incoming directions.

The communication infrastructure also has support for a variety of packet types (point-to-point, nearest-neighbor), as well as support for fault detection, isolation, and recovery. Also, the design of the large-scale system is simplified by not requiring global clock synchronization – the communication network is entirely asynchronous, which makes SpiNNaker a ‘globally asynchronous, locally synchronous’ (or GALS) system.

Since the SpiNNaker system is supposed to operate in real time, the hardware does not provide precise guarantees about spike delivery. For example, the time taken for a spike to travel through a router may be impacted by other unrelated spikes using the shared router resource. Therefore, the exact spike arrival time is not a deterministic quantity. However, this is not viewed to be an item of concern because the biological systems being modeled are supposed to be robust to small variations in spike arrival times relative to the biological time scale.

Programming

The flexibility provided by the SpiNNaker system is both attractive and has a drawback – the positive aspect is that it is easy to change the details of neuron and synaptic modeling; the negative aspect is that all the details of the models have to be specified. A significant amount of work has been put into developing models that are easily accessible to the neuromorphic systems community.

The SpiNNaker team has developed mechanisms to map neural network models in PyNN to their hardware. This approach makes the system readily accessible to users of PyNN. Conceptually this can be viewed as ‘compiling’ PyNN descriptions into the appropriate software and router configuration information that implement the same model on the SpiNNaker hardware. Significant work is currently underway to expand the set of models that can be mapped to the hardware.

16.2.2 Hierarchical AER

Another proposal for the design of large-scale neuromorphic systems is being advocated by the University of California at San Diego (UCSD) as a way to extend their IFAT architecture described in Section 13.3.1 to systems with many more neurons. The HiAER combines custom neurons and synapse modeling hardware with programmable routing.

System Architecture

The architecture of the system consists of two types of components: the IFAT board and the routing chip. The IFAT system is the core computational element that implements neurons and synapses. Each IFAT system implements an array of neurons and their associated synapses. Each IFAT node has a local routing resource associated with it that is responsible for

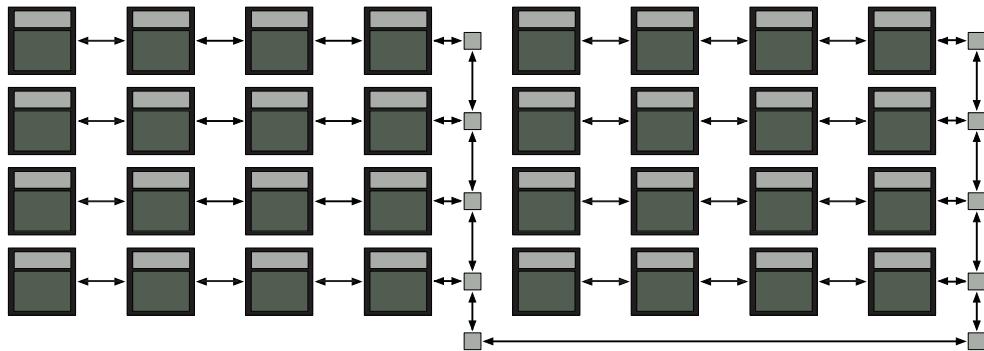


Figure 16.2 Chip-to-chip communication topology in the HiAER system, showing a 32 node system. Each node in the system includes neuron resources (shown in dark gray) and routing resources (shown in light gray), and the arrows show communication links between individual nodes. This example shows three levels of hierarchy, but the approach can be extended to an arbitrary number of levels

AER-based spike communication. Sets of IFAT/router nodes are connected in a linear array, and the edge of each array is responsible for communication to other arrays. These edge nodes are also organized in linear arrays, and so the entire system can be viewed as a hierarchy of arrays as shown in Figure 16.2.

Neurons and Synapses

The custom analog VLSI chips used in IFAT can model 2400 identical integrate-and-fire neurons each, and the IFAT board contains two of these chips along with support logic implemented using an FPGA. Switched-capacitor analog neurons on the chip implement a discrete time single-compartment model with multiple conductance-based synapses and a static leak. Neurons on the IFAT chip can be individually addressed, and only maintain an analog membrane voltage. Weights and reversal potentials are externally supplied and can therefore be configured on a per-neuron basis. An external FPGA and digital-to-analog converter (DAC) provide these parameters along with the neuron address to the IFAT chip, and this results in the specified neuron updating its membrane state (Vogelstein et al. 2004). Any spike generated by the neuron is also received by the FPGA that can then use connectivity information to transmit the spike to the appropriate set of destination neurons.

Communication

Spikes that are local to an individual IFAT array are looped back internally without having to travel through the routing network. Nonlocal routing in the HiAER system is tag-based. When a nonlocal spike is generated by an IFAT neuron, it is transmitted to the router that is immediately adjacent to the IFAT system. This *first level router* broadcasts (via individual nearest-neighbor communication) the spike to the local row, and each individual router in the

row matches the tag against a local routing table. If there is a local match, then the spike is communicated to the local IFAT array. The local IFAT array uses the tag as an index into a local routing table to identify the set of synapses for spike delivery. Since the IFAT system has a programmable FPGA and a local RAM for extra storage, the mechanism for local spike delivery can be changed as necessary.

One of the first-level routers does not have an associated IFAT array (see Figure 16.2). This router is responsible for communication beyond the local linear array. The router behaves in the same manner as the first-level router – when it receives a spike, it matches the tag against a local routing table to see if the spike should propagate to the next level of routing. If so, this spike is transmitted to the next level of routing, where a tag matching operation is performed in a manner similar to the first-level routing. The difference is that each second-level router is connected to a collection of IFAT systems and routers, not just one IFAT. This process can be repeated in a hierarchical manner to build a large-scale neuromorphic system (Joshi et al. 2010).

Each tag can be viewed as a ‘destination pattern’ – two spikes with the same tag are delivered to the same set of destination synapses. However, because there is locality in communication, parts of the global network that have nonoverlapping spike communication can use the same tags to refer different destination synapses. Therefore, the number of bits used to represent a tag is not a direct limitation on the number of total neurons/destination patterns in the system.

Programming

While the neuron model and synapse model itself cannot be changed, since it has been implemented with dedicated hardware, the parameters for each neuron and synapse are externally supplied. Therefore, programming the system corresponds to selecting the model parameters for each neuron, selecting tag values for spikes, and providing configuration information for all the routing tables in the communication network.

16.2.3 Neurogrid

The Neurogrid project at Stanford University consists of a system that comprises almost entirely of custom hardware for modeling biological neurons and synapses.

System Architecture

The core hardware element in Neurogrid is the Neurocore chip, which is a custom ASIC that uses analog VLSI to implement neurons and synapses, and digital asynchronous VLSI to implement spike-based communication (Merolla et al. 2014a). The chip was fabricated in a 180 nm process technology, and contains a 256×256 array of neurons, whose core functionality is implemented with analog circuits.

Each Neurocore chip can communicate with three other chips, with the routing network organized in a tree fashion (see Figure 16.3). The Neurogrid system is able to model one million neurons by assembling a tree of 16 Neurocore chips. Finally, it is possible to enhance

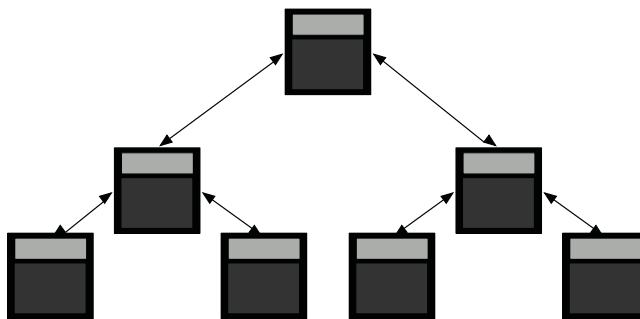


Figure 16.3 The Neurogrid system showing its tree topology between individual Neurocore chips. Each Neurocore chip can model neurons and has support for a multicast-based tree router. The third connection from the root of the tree is not shown

the flexibility of the Neurogrid system by connecting the root of the tree to an external device (e.g., an FPGA) to perform arbitrary spike processing or routing operations.

Neurons and Synapses

Each Neurocore chip contains 256×256 neurons. Each neuron is implemented using custom analog circuitry that directly implements a continuous-time differential equation model for neuron behavior. The neurons implement a quadratic integrate-and-fire (QIF) model, and is combined with four types of synapse circuits (Benjamin et al. 2012). The synapse circuits themselves implement a superposable synapse circuit, allowing a single circuit to model an arbitrary number of synapses of a given type. This approach enables the Neurogrid system to model a very large number of synapses with minimal overhead – the state required corresponds to representing connectivity between neurons rather than the connectivity to specific individual synapses.

Communication

The Neurogrid architecture uses three separate modes for communication: (i) point-to-point spike delivery; (ii) multicast tree routing; and (iii) analog fan-out. The Neurocore chips are organized in a tree, and spikes traverse up the tree to an intermediate node, and then traverse down the tree to the destination Neurocore chip. Spikes are source-routed: each spike contains the path taken by the packet through the network (Merolla et al. 2014a). For point-to-point routing, the packet simply specifies the routing information for each hop.

Multicast routing is supported by permitting packet flooding when the packet travels down the routing tree. In flooding mode, all Neurocore chips in the subtree receive a copy of the spike. Because this mode can only be used for packets traversing down the tree, the network is deadlock free, since there are no cycles in the routing graph.

A final mechanism for spike delivery supported in Neurogrid is the notion of arbors. When a spike arrives at a destination neuron, it is delivered to a programmable neighborhood of the

neuron using an analog diffusor network. This means that a single spike effectively delivers input to a population of neurons, further increasing the number of neurons receiving the spike.

Programming

While the neuron model and synapse model itself cannot be changed, since it has been implemented with dedicated hardware, the parameters for both models are externally supplied. Therefore, programming the system corresponds to selecting the model parameters for neurons and synapse types in a chip and providing configuration information for all the routing tables in the communication network.

16.2.4 High Input Count Analog Neural Network System

The High Input Count Analog Neural Network (HICANN) system in the FACETS project has very different goals compared to the previously described designs. While the previous systems were designed with real-time operation at biological time scales in mind, the HICANN system takes the approach of providing a platform that enables accelerated modeling of neural network dynamics, while supporting neurons with 256–16,000 inputs. Supporting acceleration factors of up to 10^5 was a design consideration in this project.

System Architecture

Instead of building individual chips with a communication network, the FACETS project uses wafer-scale integration. A wafer contains repeated instances of the same reticle, so the system is a repeated array of an individual HICANN chip, where each reticle contains eight HICANN chips. The choice of wafer scale integration makes fault tolerance a requirement in the FACETS design.

Neurons and Synapses

Each HICANN chip contains two *Analog Neural Network Core* (ANNCORE) arrays that contain analog circuits for modeling neurons and synapses. The analog circuits can implement two neuron models: a conductance-based integrate-and-fire model, and an adaptive exponential integrate-and-fire model.

Each ANNCore contains 128 K synapse circuits and 512 neuron membrane equation circuits organized in two groups. Each group has a 256×256 synapse array and 256 membrane circuits, together with programmable connectivity to enable a set of synapses to be grouped with each membrane circuit. If all the neuron circuits are used, each neuron can have 256 synapses. However, other configurations with fewer neurons per ANNCore and more synapses per neuron are possible, up to a maximum of 16 K synapses per neuron (Schemmel et al. 2008).

Neuron and synapse parameters can be set via a digital interface, and on-chip DACs convert them into the appropriate analog input for the ANNCore circuits.

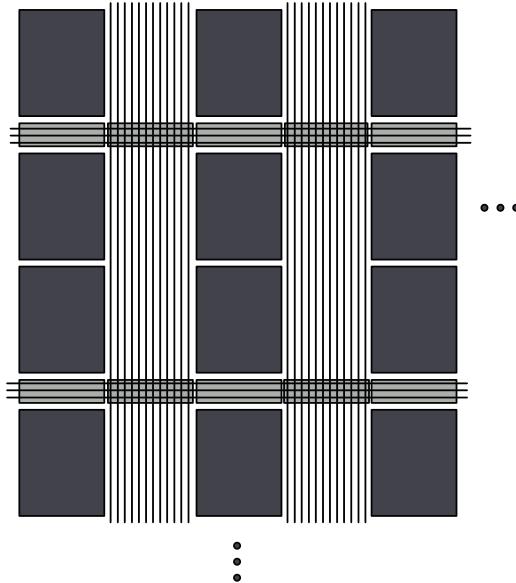


Figure 16.4 A wafer-scale FACETS system, with neuron and synapse resources (shown in dark gray), and communication resources (shown in light gray). The communication resources are statically set using programmable switches, rather than packet-based routing. Bus lanes are shown running horizontally and vertically, and the connectivity between the lanes is controlled by programmable static switches

Communication

The goal of highly accelerated modeling imposes significant stress on communication resources. By adopting a wafer-scale approach, the FACETS project leverages the higher on-chip bandwidth available in modern CMOS processes. If we view the chips on a wafer as a grid, the FACETS routing architecture uses multiple parallel lanes both in the horizontal (64 lanes) and vertical (256 lanes) directions between individual ANNcore arrays (Figure 16.4). Groups of 64 neurons time-multiplex their spikes onto a single lane. The lanes use low-voltage differential serial signaling to reduce power consumption, and address bits are transmitted in a bit-serial fashion.

Rather than using a packet-based routing approach, routing resources are statically allocated using programmable switches similar to the way an FPGA routing fabric is designed. Connections between lanes at chip boundaries can be shifted by one position, to increase the flexibility of the routing network.

Each bus can be ‘tapped’ by a HICANN chip that internally contains address matching logic. When a neuron address is transmitted on a lane that is tapped by a HICANN chip, the lane address is compared to the locally stored addresses. On a match, a spike is delivered to the appropriate synapse (Fieres et al. 2008).

There are many restrictions on connectivity and the final choices made by the FACETS design balance flexibility in routing against area and power constraints. A hardware efficiency of ~40% is shown in some of the experiments conducted by the FACETS team (Fieres et al. 2008).

Programming

Programming the system corresponds to selecting neuron and synapse groupings, parameters, and mapping the connectivity to the configurable routing resources in a manner that is analogous to the place-and-route flow in an FPGA.

16.3 Discussion

Each large-scale neuromorphic system has made very different choices when it comes to key design decisions: neuron model, synapse model, and communication architecture. A summary of the key differences is shown in Table 16.1.

The different choices made lead to systems that have different strengths. SpiNNaker is clearly the most flexible, but therefore also has the highest overhead for modeling neural systems. HiAER and Neurogrid use low-power analog neurons and synapses with digital communication and are optimized for different types of networks. In particular, Neurogrid is architected to be particularly efficient at implementing the columnar structure found in cortex, whereas HiAER has been designed to support more general connectivity. The HICANN system has been optimized for speed, and therefore cannot use the same level of multiplexing of communication links as Neurogrid or HiAER.

There are other large-scale system design projects underway, probably the most notable being the ‘TrueNorth’ (TN) effort being led by IBM to develop a low-power cognitive computing platform (Imam et al. 2012; Merolla et al. 2014b). The building block for their architecture is a ‘neurosynaptic core’, consisting of an array of digital neurons combined with synapses organized in a cross-bar configuration with a limited number of synaptic weights. The approach is fully digital, with a combination of asynchronous circuits and a synchronization clock that governs the timing precision of the neurons. The TN architects also made the decision to ensure deterministic computation; the TN neurons faithfully replicate a simulation exactly, thereby allowing easier design of a system while giving up some low-level efficiencies used in biological computation, such as analog dendritic state and probabilistic synaptic activation. This approach represents a different design point compared to the other four platforms discussed in this chapter.

Table 16.1 Key differences in choices for neuron and synapse models, and communication architecture in large-scale neuromorphic systems

System	Neurons	Synapses	Communication
SpiNNaker	ARM core (software model)	ARM core (software model)	Custom torus with diagonal links
HiAER	Integrate-and-fire, analog	Switched capacitor, conductance-based	Hierarchical, linear arrays
Neurogrid	Quadratic integrate-and-fire, analog	Superposable, continuous time	Multicast tree with analog diffusor
HICANN	Integrate-and-fire or adaptive exponential integrate-and-fire	Plastic, with 4-bit weights	Statically configurable plus flexible FPGA

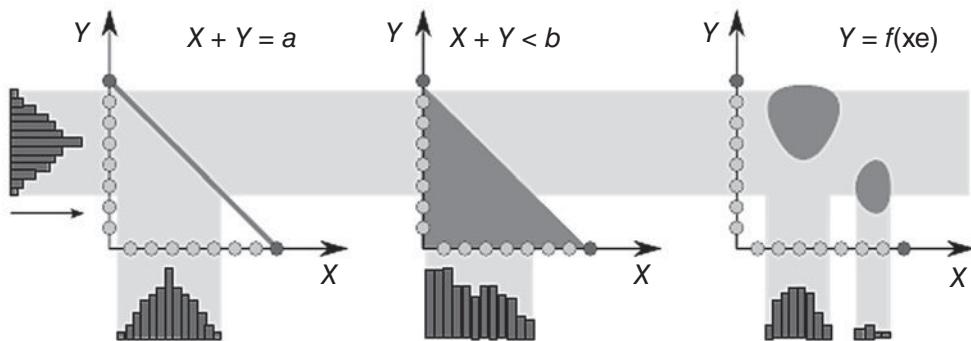
The architecture of large-scale neuromorphic systems will continue to evolve. New discoveries will be made and incorporated into architectural choices. Different researchers will take their own unique approach to the problem of modeling neuroscience. Ultimately these decisions will be validated by real-world application and market success, but it is too early to say which choices will prevail.

References

- Benjamin BV, Arthur JV, Gao P, Merolla P, and Boahen K. 2012. A superposable silicon synapse with programmable reversal potential. *Proc. 34th Annual Int. Conf. IEEE Eng. Med. Biol. Society (EMBC)*, pp. 771–774.
- Fieres J, Schemmel J, and Meier K. 2008. Realizing biological spiking network models in a configurable wafer-scale hardware system. *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, pp. 969–976.
- Furber S, Lester D, Plana L, Garside J, Painkras E, Temple S, and Brown A. 2013. Overview of the SpiNNaker system architecture. *IEEE Trans. Comput.* **62**(12), 2454–2467.
- Imam N, Akopyan F, Merolla P, Arthur J, Manohar R, and Modha D. 2012. A digital neurosynaptic core using event-driven QDI circuits. *Proc. 18th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, pp. 25–32.
- Joshi S, Deiss S, Arnold M, Park J, Yu T, and Cauwenberghs G. 2010. Scalable event routing in hierarchical neural array architecture with global synaptic connectivity. *12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, pp. 1–6.
- Merolla P, Arthur J, Alvarez R, Bussa t JM, and Boahen K. 2014a. A multicast tree router for multichip neuromorphic systems. *IEEE Trans. Circuits Syst. I* **61**(3), 820–833.
- Merolla PA, Arthur JV, Alvarez-Icaza R, Cassidy AS, Sawada J, Akopyan F, Jackson BL, Imam N, Guo C, Nakamura Y, Brezzo B, Vo I, Esser SK, Appuswamy R, Taba B, Amir A, Flickner MD, Risk WP, Manohar R, and Modha DS. 2014b. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **345**(6197), 668–673.
- Schemmel J, Fieres J and Meier K. 2008. Wafer-scale integration of analog neural networks *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, pp. 431–438.
- Vogelstein RJ, Mallik U, and Cauwenberghs G. 2004. Silicon spike-based synaptic array and address-event transceiver. *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)* **V**, 385–388.

17

The Brain as Potential Technology



This chapter turns from the concrete instantiations of event-based neuromorphic systems discussed in previous chapters to look toward a possible future. It discusses physical computation in the context of today's technology in contrast with our understanding of cortical computation, and in particular, in relation to cognition. It then proposes approaches to understanding computation in brains and how these could be rooted in the brain's self-construction. These considerations are made more concrete by an example of neural circuitry in cortex with powerful computational properties. The chapter concludes with a discussion of the eventual goal of inducing artificial cognition in neuromorphic electronic systems.

17.1 Introduction

The twentieth century brought huge advances in our understanding of information, and the methods and electronic technologies necessary for its processing. The central tasks of

The figure shows relational networks implemented by soft winner-take-all (WTA) circuits.

Event-Based Neuromorphic Systems, First Edition.

Edited by Shih-Chii Liu, Tobi Delbrück, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas.

© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

information processing are the storage, communication, and transformation of data; and the cornerstone of their implementation since the 1940s is the digital computer that was first built of electro-mechanical relays, then thermionic valves, and for the last 50 years, silicon. These machines instantly overtook human computers in speed and accuracy of arithmetic. The detailed methods for performing these tasks are of course provided by us – intelligent humans – and are not yet intrinsic to the information processors themselves. We have yet to incorporate this creative biological intelligence into technology. Indeed, the feature that distinguishes biological intelligence from more simple information processing is the biological agent's ability to extract and exploit appropriate knowledge from the world, and to apply that knowledge for its economic advantage – as in feeding itself, defending itself, breeding (sometimes with itself), and communicating its needs and intentions, mostly with others. The important consideration here is that 'knowledge' is not simply stored data. It is a flexible, meaningful, purposeful, and context-sensitive organization of immediately accessible data – a cross between an encyclopedia and a DIY manual, located in the biological brain.

Progress toward artificial autonomous intelligent systems capable of real-world interaction has been slow simply because this intrinsically goal-oriented form of data organization and processing that characterizes brains, remains poorly understood in engineering terms. It is humbling that the intelligent coordination of flying, navigation, foraging, and communication by a honey bee (which possesses a rather modest brain of only one million neurons) far exceeds the intelligent performance of the most advanced autonomous artificial systems we can currently build. In the present age of the global proliferation of the use of silicon-based technologies that are increasingly sophisticated and personalized, it is worth re-emphasizing that every significant expression of knowledge on this planet still has its source in biological nervous systems, and not in technology.

Few branches of neuroscience, however, are actually striving to understand, let alone resolve this fundamental disparity. The major advances in autonomous intelligence now come from developments in machine learning algorithms whose relation to brain architectures and processes are remote. However, neuromorphic engineering does have as its goal the development of artificial neural systems whose processing architecture and principles are based on those of biological nervous systems. As we have seen in this book, they are usually composed of hybrid analog and digital electronic circuits that emulate biological sensors and their processing neuronal networks. Initially, the big challenges for neuromorphic engineering were simply to understand how to exploit the properties of silicon devices to build circuits in a neuromorphic style, and to combine sensors and neuron-like computational elements into reactive systems, integrating them in such way that they expressed their processing and behavior in a 'neurally inspired' way. These goals have now been reached, at least academically if not industrially, and the scientific focus is shifting toward the conceptual and technical problems of how to induce more sophisticated behaviors in neuromorphic systems. The ultimate aim is to achieve a neuromorphic 'cognition' that resembles that of animals in being capable of creating, storing, and manipulating knowledge of its external and internal world, and of using this knowledge for economically advantageous behavior. The induction of effective brain-like cognitive behaviors in autonomous artificial agents offers enormous economic, technological, and social benefits; and so the abstraction of engineering principles from the biological brain is a major challenge for the twenty-first century. Harnessing the principles of biological intelligence can be expected to have a major impact on the technology market, as autonomous

intelligence becomes incorporated into equipment, vehicles, buildings, utilities, and clothing, which some see as the relentless march to the emergence of a truly ‘prosthetic human’.

17.2 The Nature of Neuronal Computation: Principles of Brain Technology

After a century of ‘modern’ neuroscience, we know a great deal about the nuts and bolts of nervous systems, yet as we have seen in this book, only little of this knowledge has been translated into technology. Why is it then that there are still very significant differences between current computers and nervous systems in the ways that computational processing is implemented? It is worth examining the chief differences between the two, which lie in basic processing components, their system architecture, information encoding, methods of configuration, and methods of construction and calibration.

As we have seen throughout this book, the computational elements of neuronal systems process data asynchronously, and on demand: there is no central clock that enforces global coherence. The high carrier mobility of electronics allows computers to use extremely fast components in order to perform their processing and at the same time allows the use of semi-global synchronization, which simplifies the design process for logic circuits. By contrast, all neuronal components operate in real physical time and so are inherently synchronized between themselves as well as with world events. The memories of synapses and their processing by neurons are intimately colocalized; unlike conventional computers, there is no organizational segregation of memory and processing. Unlike computers, there is no random access to the memories of neuronal systems. Instead, the loading of neuronal memory occurs principally by changing strengths of synaptic connections and neuronal growth. These strengths are not only changed on local performance criteria, but also as a consequence of the successful performance of entire neuronal subsystems. This performance-based plasticity means that the configuration of biological memory is a consequence of processing rather than the cause of it. Moreover, the overall assembly and configuration of neuronal systems occur by self-assembly, which means importantly that there is no external source of knowledge and skill that builds and programs them. These differences are a challenge for technology, and their understanding is likely to depend largely on research in the neurosciences.

Technological computation is the systematic (algorithmic) transformation of symbols, without any intrinsic regard for the meaning and significance of the symbolic data. Their meaning and significance are extrinsic to the computation and derive from the interpretation of human programmers who design the symbolic encodings of the data and the algorithms that manipulate them. Consequently, present methods effectively ensure that intelligence is not inherent to computation, and this is likely to remain the case as long as the behavior a computation entails is determined by its human programmers and not by the computer itself. If we see cognition, in particular, as purely thinking without action, it seems unlikely that any sessile hardware will ever be compelled to reflect on the consequences of its actions. On the other hand, there is no reason to expect that present methods of computation are unable to express intelligence. Instead, it is likely that intelligence is a particular style of computation in which the attribution of meaning, significance, and purpose arise out of the self-organization of encodings by the algorithms themselves, rather than the external programmers. The relevance of the meanings

that ground encoded data can be assessed in terms of the predictive power that these meanings bring to a computation.

Intelligent behavior depends on this predictive power. It permits the successful agent to interact efficiently and flexibly with its internal and external world (including other agents) and thereby gain advantage with respect to its friends and foes, even in situations where the environment itself may be changing. Intelligence requires an ability to quickly extract relevant information from the sea of irrelevant sensory input; to reason on these data by deduction, induction, and transduction; to test efficiently the hypotheses of reasoning by crucial experiment; and to evaluate plans of action against probable reward. Many of these concepts have been recognized during the twentieth century. That these lessons have been well-learned is shown by the remarkable and well-publicized successes in recent years in producing artificially intelligent behavior in some domains (e.g., chess [Deep Blue, (Hsu 2002)]; autonomous vehicles [Stanley, (Thrun et al. 2007)]; ‘Jeopardy’ [Watson, (Ferrucci et al. 2010)]), however, we have made only modest progress toward the implementation of such behaviors in a self-organizing medium analogous to that of biological neural networks. We still lack fundamentally an understanding of the relationship between the intelligent algorithms and the medium that must support them. And the little we have been able to achieve in these directions using machine learning ultimately must be computed on hardware that wastefully burns power enforcing internal precision everywhere at every instant, even when it is not necessary.

At root, all computation is a physical process that must be instantiated in a physical medium. Algorithms are expressed in the context of an abstract architecture, which is an organization composed of processors and channels able to communicate information between these processors. The physical implementation of the architectures necessary to support algorithms for general purpose digital computers are well understood. However, the relationship between algorithms and their physical implementation is much less clear for parallel digital computers and hybrid analog–digital computers; and remains nearly unknown for neuronal systems. This latter evaluation is quite staggering in view of our present, rather detailed knowledge of the physical and electrochemical properties of neurons; so why are we struggling so? The reason for the difficulty is that even if we assumed that the principles of brain computation were similar to those used in conventional computing systems, the implementation of neural computation is very different. In particular, biology has evolved by means of self-construction, self-repair, and self-programming that allow it to generate complex behaviors from simpler elements. These biological abilities are not well understood, but will surely have consequences for the design of future artificial information processing and behaving systems, probably even more so in some future ‘post silicon’ era.

17.3 Approaches to Understanding Brains

At first sight – and more so on the second – the complexity of biological processes is daunting, and extremely so in the case of the brain. The most frequently repeated cliché about our brain is that it is the ‘most complex organ we know of in the universe’. If biological processes are indeed too complex to be understood, they will inevitably have little practical impact on the design of artificial systems. However, the history of neuroscience shows that when we are able to abstract wisely from biological observation, strong technological insights can follow.

For example, in the early 1940s McCulloch and Pitts (1943) demonstrated how simple logic gates, which were abstractions of neurons, could be prescriptively assembled into sequential chains that compute effectively any required function. Through their eponymous ‘neuron’ they established a view of the principle and implementation of artificial and neural information processing that continues to dominate both computer science and neuroscience. And soon after, Hodgkin and Huxley (1952) demonstrated, with their powerful dynamical model of voltage-sensitive channel molecules, the general principle of neuronal electrophysiological signal processing, which became the cornerstone for understanding the nature and relevance of synaptic interactions between neurons.

These two epochal studies offered the promise that processing in neurons and their networks could be completely understood as (largely organizationally static) networks of exotic, but simple signal processors, in which only the electrical signals of membranes and their coupling between neurons are the relevant information processing domain. This paradigm is still very dominant in the twenty-first century, but of course its appealing simplification neglects the fact that biological cells, including their membranes, are essentially vast, living, metabolic machines that by their very existence express a complex organization of metabolic and molecular computation, of which membrane electrical phenomena are only one useful consequence. Nevertheless, the distributed signal processor view gave rise to decades of productive research into the computational properties and powers of artificial neural networks (ANNs). These results were focused on two major circuit architectures characterized by either feedforward or recurrent connections. The feedforward circuits were very attractive to psychologists and to computer scientists, who drove the field of ‘Connectionism’ and used feedforward networks to model a number of psychophysical and cognitive functions, such as Gestalt phenomena, depth perception, and speech acquisition. The feedback circuits appealed generally to physicists, who explored the dynamical, chaotic properties that inevitably arise in recurrent systems. This work on recurrent networks, most notably that of Hopfield and that of Wilson and Cowan, was, like the connectionists, enormously influential on a generation of theorists. However, in retrospect it had rather little influence on experimentalists or engineers trying to develop neuromorphic systems.

While neural network theorists were having their day in the sun, experimental neuroscientists were painstakingly developing methods for observing and measuring the actual connectivity of individual neurons and their networks. One brain region that has always attracted a lot of attention in this regard is that very successful processor of autonomous intelligent behavior, the neocortex of the mammalian forebrain. The striking conclusion of the experimentalists’ labors is that the neuronal circuits of the neocortex cannot be easily characterized as being either feedforward or feedback in the ANN sense. Instead, neuroscientists are teasing apart what appears to be a rather complex, special-purpose processor composed of some hundred different neuronal types.

Although the neocortex contains a vast number of neurons (10^9), the results of two decades of sampling their detailed morphologies suggests that the overall architecture of the neocortex is rather regular. Ideally of course, we would like to have a detailed map of all connections between all neurons; a map that is now called the ‘connectome’ (Sporns et al. 2005). Indeed, one of the most successful scientific strategies for understanding unknown structures is to build such maps. Scientists have built maps of the universe, galaxy, the earth, plants and animals, and the genomes of living organisms. To date, the only brain that has been mapped out completely at synaptic resolution is that of *Caenorhabditis elegans*, a small nematode worm, whose nervous

system consists of only 302 neurons, in its most common hermaphrodite incarnation. Now this goal of comprehensive reconstruction is being extended to more complex brains, particularly the mammalian brain. This endeavor of ‘dense reconstruction’ many consider to be one of the main scientific challenges of the twenty first century (see e.g., Blu (n.d.) and ATL (n.d.)). The notion of ‘reverse engineering’ plays strongly in the arguments in favor of this program. However, as the pilot of the exact replica of the Wright brothers’ ‘Flyer’ found in their failed re-enactment of Orville’s first powered flight at Kitty Hawk on the 100th anniversary of the event, more knowledge than is contained in an exact physical replica is needed to get airborne.

But even before the first powered flight of a replica brain is attempted, there are some formidable technical obstacles to be overcome. One is the sheer size of the problem. Since dense reconstruction aims at a description of the circuit at synaptic resolution, this cannot be done using a light microscope, but needs the magnification of an electron microscope. Image acquisition using a transmission or scanning electron microscope is slow. With an estimate of roughly 10 seconds per $2 \mu\text{m} \times 2 \mu\text{m}$ electron micrograph and a total of over $100 \times 100 \times 3000$ micrographs required for the reconstruction of a brain volume of $200 \mu\text{m}^3$, we can estimate the total image acquisition time using a single electron microscope to be about 10 years (note that a volume of $200 \mu\text{m}^3$ corresponds to an entire brain area in a small songbird such as the zebra finch, but is smaller than a whisker ‘column’ in a rat, and only a fraction of a hypercolumn in monkey or human neocortex). The second obstacle is to make sense of all the hundreds of terabytes of images using automated image processing algorithms. This is why alternative strategies have made much more rapid progress, for example, those that exploit regularities in the structure and that aim for a statistical description of the connections, rather than a wire-by-wire, synapse-by-synapse description of the circuit. These strategies have led to the discovery of most of the important principles of cortical circuit structure and function known to date.

17.4 Some Principles of Brain Construction and Function

The investigation of neocortical structure and its development has entered an exciting phase in which detailed structural and functional organization and the molecular mechanisms that support them are becoming ever more rapidly accessible to experiment. We have reached the stage where a large-scale systematic research attack on cortical organization could harvest benefits analogous to those seen in molecular biology. Significantly, during the last decade there has been an explosion of methods and experiments that are able to explore the objective/subjective interface of neuronal processing. Methods such as fMRI and fluorescent-dye imaging, and electrophysiological recordings in awake, behaving, and communicating animals and humans permit us to link the objective operation of neurons and their networks to the subjective experience of the behaving animal. Here again, we see a fertile area for research and we can expect major new insights in the next few years.

One clear target for investigation is the architecture and operation of the neocortex in order to obtain insights into the ‘computational’ methods that nature has evolved, and how they support the intelligence expressed by cortex. In this context, the concept of a ‘canonical circuit’ for the neocortex has been invaluable in providing a biologically-based, but simplified description of a circuit that captures essential connectivity and function of a local patch of cortex. The main action of cortical processing takes place in the local circuit, so defining the features of this circuit is a major step in understanding cortical function. The ‘canonical’ appellation expresses

the hypothesis that this circuit generalizes to all cortical areas in all mammalian species. This hypothesis is widely assumed to be so, but has yet to be rigorously tested. Data arising from the few areas that have been studied in rodent, cat, and monkey offer encouragement. This concept of a canonical circuit is important at many levels, especially in providing a theoretical framework with which to link animal and human studies and in providing a direct means of realizing key features of cortical processing in neuromorphic circuits.

Current data show that the cortical circuits can be best understood in terms of the laminar distribution of relatively few (between say 10–100) types of excitatory and inhibitory neurons (Gilbert 1983; Gilbert and Wiesel 1983). The degree of connection between these types have been estimated for the static anatomy of cat visual cortex *in vivo* (Binzegger et al. 2004), and also for physiological connections between some neuronal types in rat somatic cortex *in vitro* (Thomson et al. 2002). These functional and other quantitative anatomical circuit data indicate a number of intriguing circuit properties that offer many challenging clues to fundamental properties of cortical processing.

One clue is the dominance of local cortical synapses over those provided by individual afferents. Overall, the vast majority of excitatory synapses, and almost all inhibitory synapses, originate from neurons within the cortex (Braitenberg and Schüz 1998). Even more than this, within a given cortical area the majority of synapses are derived from neurons within that area (Binzegger et al. 2004). By contrast, the afferent projections to cortex from either thalamus or other individual cortical areas, each contribute a surprisingly small percentage of all excitatory synapses in the target area. For example, in the visual cortex, synapses from the lateral geniculate nucleus form less than 10% of all the excitatory synapses on their main target neurons that lie in the middle layers of area 17 of cats and monkeys (Ahmed et al. 1994; Garey and Powell 1971; Latawiec et al. 2000; Winfield and Powell 1983; da Costa and Martin 2009). Yet these afferents clearly provide sufficient excitation to drive the cortex. Similarly, the interareal projections also form a few percent of the synapses in their target layers, yet both the ‘feedforward’ and ‘feedback’ interareal circuits are evidently functionally significant. This raises the interesting functional question of how the local cortical circuits reliably process their small input signals.

A second clue is the large fraction of excitatory connections that occur between pyramidal cells of the superficial cortical layers. Although excitatory neurons in other layers, such as the spiny stellate neurons of layer 4, also receive input from their neighbors, it is only in the superficial layers that the pyramidal cells make very extensive arborizations within their own layer. Indeed, nearly 70% of a superficial pyramid’s excitatory input is derived from other cells of its own type. Consequently, first-order recurrent connections between superficial pyramidal cells, whereby a target neuron projects back to its source neuron in a tight positive feedback loop, are more likely than in any other layer. Douglas et al. (1995), Hahnloser et al. (2000) and others (see for example, Ben-Yishai et al. 1995; Pouget et al. 2003) have proposed that positive feedback plays a crucial role in cortical computation by providing gain for active selection and recombination of the relatively small signals arriving from the sensory periphery.

The overall gain of any system is the ratio of its output and input signals. However, in feedback systems, two additional gain concepts enter into discussion. The first is the open loop gain, measured across the system with feedback disconnected. The second is the loop gain, which is measured around the feedback loop and so is the product of the forward and feedback elements. This loop gain may be negative or positive with respect to the input. In conventional analog electronic feedback circuits the forward gain is typically very large, but in neuronal

networks it is only modest. Negative feedback is inherently stable because it subtracts from the input. Electronic circuits frequently combine high forward gain with negative feedback across precisely-controlled passive elements (e.g., capacitors) to make amplifiers less sensitive to forward gain variation caused by the much less precisely controlled transistors; see for instance, the pixel circuit of the DVS in Chapter 3. And in the Axon-Hillock circuit of Section 7.2.2, positive feedback across precisely controlled capacitive feedback ensures a precisely controlled amount of input charge per spike. While individual biological neurons have high gain at the edge of threshold when they are about to spike, in neuronal networks that lack high forward gain, smaller negative feedback controls dynamic range and provides circuit stability.

Positive feedback is potentially unstable because it sums with the input. This positive feedback can provide very large overall signal amplification by the system. But, should the positive loop gain become too large, the output is unstable. In electronic circuits where the signal range is inherently limited, strong positive feedback is used to implement decisions and digital signal restoration. However, for nervous systems the situation is more complex. Neurons fire sparsely, and then only at a small fraction of their maximum discharge rate. For them, high positive feedback would appear to carry an ever present danger of divergence into epilepsy. Nevertheless, the high fraction of recurrent excitatory connections observed in at least the superficial layers of the cortex, suggests that some neural networks employ strong positive feedback to advantage.

The feedforward gain of individual neurons operating in rate mode is small. By *rate mode*, we mean the average rate of spikes over some time scale is high enough that over a relevant time scale the rate has a meaning. Typically, many input spike events must be applied to a neuron before it produces a single spike output. However, simulation studies and physiological evidence suggests that cortical circuits can generate significant system gain by the positive feedback excitation (Douglas and Martin 1991; Ferster et al. 1996; Nelson et al. 1994) mediated by recurrent intracortical axonal connections (Ahmed et al. 1994; Douglas et al. 1995; LeVay and Gilbert 1976; Peters and Payne 1993).

Positive feedback amplification may seem inherently dangerous, but neurons subject to positive feedback can be stable if the sum of their input and positive feedback excitatory currents are less than the total negative current dissipated through their neuronal leak, action potential, and inhibitory conductances. Such circuit-dependent stability must exist in cortex, because the steady discharge rate of active cortical neurons is usually much less than their maximum rate, and so stability does not depend on the neurons being driven into discharge saturation.

17.5 An Example Model of Neural Circuit Processing

Some insights into the properties of recurrently connected neuronal networks have come from the study of simplified artificial neural network models. Although these models are necessarily much simpler than networks of real cortical neurons and their connections, they do capture some of the canonical principles we find in cortical networks and they have the principal advantage that their modes of behavior can be clearly understood and used to interpret the experimentally observed organization and operation of the cortical networks. For example, Hopfield and others (Cohen and Grossberg 1983; Hopfield 1982, 1984; Hopfield and Tank

1985) showed that recurrent networks of ideal neurons are dynamical systems whose stable patterns of activation (or attractors) can be viewed as memories or the solutions to constraint satisfaction problems. More recently, there has been a growth in the use of networks of linear threshold neurons (LTNs) to understand cortical circuits (Ben-Yishai et al. 1995; Douglas et al. 1995; Hahnloser et al. 2000; Hansel and Sompolinsky 1998; Salinas and Abbott 1996). LTNs have analog (nonspiking) positive outputs that are directly proportional to the positive difference between the excitation and inhibition that they receive. If this difference is zero or negative, they remain silent. LTN neurons are interesting because their threshold behavior and linear response properties capture some properties of cortical neurons.

A frequently studied LTN network consists of two populations of neurons: one of excitatory neurons, and a smaller population of inhibitory neurons (even one inhibitory neuron will suffice). For simplicity, the patterns of connection within each population are homogeneous. The excitatory neurons receive feedforward excitatory connections that carry the input signal, feedback excitatory connections from other members of their population, and feedback inhibitory connections from the inhibitory neuron(s). Often the population of excitatory neurons are arranged as a one dimensional spatial map and the pattern of their recurrent connection strengths is regular, typically a hill-shaped function of distance of a source neuron from its target.

Even such simple recurrent networks have interesting properties that have contributed directly to our understanding of signal processing by cortical circuits. A key finding is that these properties arise out of the interaction between the feedback excitation that amplifies the weak external inputs to the network, and the nonlinearity introduced by the inhibitory threshold that itself depends on the overall network activity. The positive feedback enhances the features of the input that match patterns embedded in the weights of the excitatory feedback connections, while the overall strength of the excitatory response is used to suppress outliers via the dynamical inhibitory threshold imposed by the global inhibitory neuron. In this sense, the network can actively impose an interpretation on an incomplete or noisy input signal by restoring it toward some fundamental activity distribution embedded in its excitatory connections.

The explanation for this remarkable emergent property is as follows (for details see Hahnloser et al. 2000). Consider the network just described. The synaptic interactions between the various neurons of the network can be described by a weight matrix. Notice, however, that if a neuron is not active, it does not express its interactions. In this case, the full weight matrix of the network can be replaced by a reduced matrix, the ‘effective weight matrix’ that is similar to the full matrix, but with all entries of the silent neurons zeroed out. Consequently, as various neurons rise and fall across their discharge threshold, the effective weight matrix will change. For those familiar with electronic circuit simulation, it is as though the network dynamically changes to a different small-signal operating point whose behavior is determined by the large signal response to the input. Some of these matrices may be stable, but others may not be. We will consider how the network can converge to its steady-state output by passing through various combinations of active neurons and so various effective weight matrices.

Imagine that a constant input pattern is applied to the input neurons. Now, in the unlikely case that the outputs do not change at all, then the job is done; the network has already converged. More likely, some or all of the neurons will change their activity due to a combination of feedforward and feedback activation. One possibility is that all the neurons could increase their activity as a result of unstable positive feedback. But this instability is forbidden by the

common inhibition applied to all excitatory neurons, which, provided the inhibition is strong enough, precludes a regenerative common increase in activation. The remaining possibility is that although the feedback is unstable, only some neurons are able to increase their activation, while others must decrease theirs. Then, eventually one of the decreasingly active neurons will fall beneath threshold, at which stage it no longer contributes to the active circuit and so the effective weight matrix must change, removing the interactions of this newly silent neuron. This pruning process continues until finally the network selects a combination of neurons (a ‘permitted set’, see Hahnloser et al. 2000, 2003) whose effective weight matrix is stable, and allows the network to converge to a steady state. The important observation here, is that positive feedback can be unstable (the feedback gain is greater than one) during the transient behavior of the network, and that the network can use this instability to explore new partitions of active neurons until a suitable (stable) partition, consistent with the input pattern, is found. The computationally interesting properties of the recurrent cortical circuits rest in this modulation of the strength of the positive feedback.

17.6 Toward Neuromorphic Cognition

The steps from understanding computation, to evoking sophisticated functionality are not simple. It took nearly half a century, trillions of dollars, and a huge industrial and commercial incentive to advance from assembler language to object-oriented coding. So too, will be the steps from neuromorphic computation to neuromorphic cognition.

A system exhibits cognition when it is capable of creating, storing, and manipulating knowledge of the world and of itself, and of using this knowledge for economically advantageous behavior. To the extent that science has been able to evoke artificial cognition at all, it has been based on symbolic encodings of the world processed on conventional digital computers that use predominantly nonreal-time, serial, synchronized electronic processing. As described in great detail in this book, neuromorphic engineers have succeeded in establishing methods for constructing a different style of processing system, which is predominantly real-time, highly distributed, and uses asynchronous events. Although this technology is more similar to neuronal systems, even the most sophisticated VLSI neuromorphic systems created thus far tend to be reactive in quality, and fall short of expressing cognitive abilities. A common remark from outsiders is that equivalent functionality can be achieved in a few lines of conventional software code, which is not the point, at least not if we take a long-range view. Of course, the things we can do now with conventional hardware are best done using conventional software because these technologies were developed together, so that possible behaviors of the hardware can be expressed concisely. Rather, one should ask, what are the causes of the shortfalls in neuromorphic technology?

These shortfalls are partly due to the stage of overall development of the field. The first order of business has been the subject of this book and has consisted of the bottom-up engineering of neuron-like hardware, a project which has necessarily taken many person-decades of effort from a small research community. And at least some of the technological developments, for example, the sensors described in Chapters 3 and 4; the communication architectures and circuits described in Chapters 2 and 12; and the systems built using these developments described in Chapter 15; can have short-term industrial impact, while the theoretical developments in learning from Chapter 6 and the related circuits from Chapters 10 and 8 could form key

parts of future hardware for learning. But a large challenge for neuromorphic engineers is to demonstrate whether neuromorphic architectures and computation confer any advantage over conventional digital methods for implementing cognition, not just more efficient computation. That the community recognizes this challenge and has responded to it, can be seen in the recent establishment in Europe of the lively CapoCaccia Cognitive Neuromorphic Engineering Workshop (Cap n.d.); and in the United States, the redirection of the NSF Telluride Neuromorphic Cognition Engineering Workshop (Ins n.d.) also toward that goal.

In what way could neuromorphic style circuits lend themselves to our understanding of cognition? It is already known that cortical networks can behave in ways that are quite different to conventional electronic circuits, and that they implement novel styles of computation. However, due to limited resources, many important research avenues are being neglected. For example, high-level HDLs have been developed that can map algorithms onto modular synchronous digital circuits, and so compose massively parallel processors. It might seem a natural step to apply these methods to the analyses of natural neural computing architectures such as cortex, but this step has not yet been taken. Perhaps one reason for this hesitation is that the computational problem in biology is much more complicated than in engineering, not least because both the algorithms and the architecture are poorly understood and so closely interlinked.

The Church–Turing thesis implies that, if cognition is algorithmic, then cognition must be implementable as a Turing machine. But what is not immediately apparent in this argument is that conventional algorithmic processing of information proceeds without any intrinsic regard for the meaning and economic significance of the data that are processed. The meaning and significance are extrinsic to the computation and currently these attributes are derived from the interpretation of the human programmers who designed the encodings of the data and the algorithms that manipulate them. That is, the cost-benefit assessment that is fundamental to intelligence is not inherent in the computation. By contrast, biological intelligence is the result of a particular style of computation in which the attribution of meaning, significance, purpose and so on arise out of the self-organization of encodings by the algorithms themselves, rather than being imposed by external programmers.

Experimental neuroscience offers approximate functional models, most of which are very tied to a specific piece of data and a specific region of cortex. Theoretical neuroscience proposes how such functions can be computed by abstract ANNs that have varying degrees of biological realism. As a result, there is much less understanding of how these functions are realized by the actual networks of neocortex, for which we still do not have a single detailed 3D circuit diagram, or any general rules to specify how the many different types of neurons interconnect to support generic functions. So, progress toward a means of systematically configuring artificial neural networks by methods other than laborious training methods has been very slow. For example, as yet we are unable to program neural networks to reason over past, present and predicted information in the manner that is so characteristic of intelligent animal behavior. The ability to configure (as opposed to train) the behavior of neural networks for a necessary task is an indispensable ingredient for building large engineered applications of neuromorphic systems. In a step toward such systems, Rutishauser and Douglas (2009) have recently shown how neuronal networks with a nearly uniform architecture can be configured to provide conditional branching between neuronal states. They show that a multistable neuronal network containing a number of states can be created by coupling two recurrent networks whose synaptic weights have been configured for soft winner-take-all (sWTA) performance.

The two sWTAs have homogeneous, locally recurrent connectivity except for a small fraction of recurrent cross-connections between them, which are used to embed the required states. The coupling between the maps allows the network to continue to express the current state even after the input that evoked that state is withdrawn. In addition, a small number of ‘transition neurons’ implement the necessary input-driven transitions between the embedded states. The large numbers of neurons required makes it premature to follow this approach to engineer a short-term practical application. However, the significance of this finding is that it offers a method whereby cortex-like ensembles of neurons could be configured for sophisticated processing by applying only small specializations to the same generic neuronal circuit. These machines could extend well beyond the brittleness of conventional state machine implementations.

Another interesting approach to configurable neural circuits is the Neural Engineering Framework (NEF) of Eliasmith et al. (2012), which allows the construction of arbitrary dynamical systems from spiking neurons. In general, such configurable neural models are rare, but also more than ever relevant to the biologists, who presently perform shotgun methods to data-gathering, rather than asking theory-directed questions to guide their investigations. In the direction of engineering, the use of biologically driven ideas will be a key ingredient by which neuromorphic engineering can achieve the goal of achieving neuromorphic cognition.

References

- Ahmed B, Anderson JC, Douglas RJ, Martin KA, and Nelson JC. 1994. Polyneuronal innervation of spiny stellate neurons in cat visual cortex. *J. Comp. Neurol.* **341**(1), 39–49.
- ATL n.d. ATLUM, <http://cbs.fas.harvard.edu/science/connectome-project/atlum> (accessed August 6, 2014).
- Ben-Yishai R, Bar-Or RL, and Sompolinsky H. 1995. Theory of orientation tuning in visual cortex. *Proc. Natl. Acad. Sci. USA* **92**(9), 3844–3848.
- Binzegger T, Douglas RJ, and Martin KAC. 2004. A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci.* **24**(39), 8441–8453.
- Blu. n.d. The blue brain project EPFL, <http://bluebrain.epfl.ch> (accessed August 6, 2014).
- Braitenberg V and Schüz A. 1998. *Cortex: Statistics and Geometry of Neuronal Connections*, 2nd edn. Springer, Heidelberg.
- Cap. n.d. Capo Caccia Cognitive Neuromorphic Engineering Workshop, <http://capocaccia.ethz.ch/> (accessed August 6, 2014).
- Cohen MA and Grossberg S. 1983. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. Syst. Man Cybern. SMC-13*, 815–826.
- da Costa NM and Martin KAC. 2009. The proportion of synapses formed by the axons of the lateral geniculate nucleus in layer 4 of area 17 of the cat. *J. Comp. Neurology* **516**(4), 264–276.
- Douglas RJ and Martin KAC. 1991. A functional microcircuit for cat visual cortex. *J. Physiol.* **440**(1), 735–769.
- Douglas RJ and Martin KAC. 2004. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.* **27**, 419–451.
- Douglas RJ, Koch C, Mahowald M, Martin KAC, and Suarez HH. 1995. Recurrent excitation in neocortical circuits. *Science* **269**(5226), 981–985.
- Eliasmith C, Stewart TC, Choo X, Bekolay T, DeWolf T, Tang C, and Rasmussen D. 2012. A large-scale model of the functioning brain. *Science* **338**(6111), 1202–1205.
- Ferrucci D, Brown E, Chu-Carroll J, Fan J, Gondek D, Kalyanpur AA, Lally A, Murdock JW, Nyberg E, Prager J, Schlaefler N, and Welty C. 2010. Building Watson: an overview of the DeepQA project. *AI Magazine* **31**(3), 59–79.
- Ferster D, Chung S, and Wheat H. 1996. Orientation selectivity of thalamic input to simple cells of cat visual cortex. *Nature* **380**(6571), 249–252.
- Garey LJ and Powell TPS. 1971. An experimental study of the termination of the lateral geniculocal cortical pathway in the cat and monkey. *Proc. Roy. Soc. Lond. B* **179**(1054), 41–63.
- Gilbert CD. 1983. Microcircuitry of the visual cortex. *Annu. Rev. Neurosci.* **6**, 217–247.

- Gilbert CD and Wiesel TN. 1983. Functional organization of the visual cortex. *Prog. Brain Res.* **58**, 209–218.
- Hahnloser RH, Sarpeshkar R, Mahowald MA, Douglas RJ, and Seung HS. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **405**(6789), 947–951.
- Hahnloser RH, Seung HS, and Slotine JJ. 2003. Permitted and forbidden sets in symmetric threshold-linear networks. *Neural Comput.* **15**(3), 621–638.
- Hansel D and Sompolinsky H. 1998. Modeling feature selectivity in local cortical circuits [Chapter 13]. In: *Methods in Neuronal Modeling: From Synapse to Networks* (eds Koch C and Segev I), 2nd edn. MIT Press, Cambridge, MA. pp. 499–567.
- Hodgkin AL and Huxley AF. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* **117**, 500–544.
- Hopfield JJ. 1982. Neural networks and physical systems with emergent selective computational abilities. *Proc. Natl. Acad. Sci. USA* **79**(8), 2554–2558.
- Hopfield JJ. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA* **81**(10), 3088–3092.
- Hopfield JJ and Tank DW. 1985. “Neural” computation of decisions in optimization problems. *Biol. Cybern.* **52**(3), 141–152.
- Hsu FH. 2002. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.
- Ins, n.d. Institute of Neuromorphic Engineering, <http://www.ine-web.org/> (accessed August 6, 2014).
- Latawiec D, Martin KA, and Meskenaite V. 2000. Termination of the geniculocortical projection in the striate cortex of macaque monkey: a quantitative immunoelectron microscopic study. *J. Comp. Neurol.* **419**(3), 306–319.
- LeVay S and Gilbert CD. 1976. Laminar patterns of geniculocortical projection in the cat. *Brain Res.* **113**(1), 1–19.
- McCulloch W and Pitts W. 1943. A logical calculus of ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133.
- Nelson S, Toth L, Sheth B, and Sur M. 1994. Orientation selectivity of cortical neurons during intracellular blockade of inhibition. *Science* **265**(5173), 774–777.
- Peters A and Payne BR. 1993. Numerical relationships between geniculocortical afferents and pyramidal cell modules in cat primary visual cortex. *Cereb. Cortex* **3**(1), 69–78.
- Pouget A, Dayan P, and Zemel RS. 2003. Inference and computation with population codes. *Annu. Rev. Neurosci.* **26**, 381–410.
- Rutishauser U and Douglas RJ. 2009. State-dependent computation using coupled recurrent networks. *Neural Comput.* **21**(2), 478–509.
- Salinas E and Abbott LF. 1996. A model of multiplicative neural responses in parietal cortex. *Proc. Natl. Acad. Sci. USA* **93**(21), 11956–11961.
- Sporns O, Tononi G, and Kötter R. 2005. The human connectome: a structural description of the human brain. *PLoS Comput. Biol.* **1**(4), e42.
- Thomson AM, West DC, Wang Y, and Bannister AP. 2002. Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2-5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. *Cereb. Cortex* **12**(9), 936–53.
- Thrun S, Montemerlo M, Dahlkamp H, Stavens D, Aron A, Diebel J, Fong P, Gale J, Halpenny M, Hoffmann G, Lau K, Oakley C, Palatucci M, Pratt V, Stang P, Strohband S, Dupont C, Jendrossek LE, Koelen C, Markey C, Rummel C, Niekerk J, Jensen E, Alessandrini P, Bradski G, Davies B, Ettinger S, Kaehler A, Nefian A, and Mahoney P. 2007. Stanley: the robot that won the DARPA Grand Challenge. In: *The 2005 DARPA Grand Challenge* (eds Buehler M, Iagnemma K, and Singh S). Springer Tracts in Advanced Robotics, vol. 36. Springer Berlin, Heidelberg. pp. 1–43.
- Winfield DA and Powell TP. 1983. Laminar cell counts and geniculo-cortical boutons in area 17 of cat and monkey. *Brain Res.* **277**(2), 223–229.

Index

- action potential, 162, 164
- adaptive exponential I&F neuron, 174, 342
- Address Event Representation, *see* AER
- address translation, 306
- AER, 12
 - 0.02, 28–29
 - 2D winner-take-all, 338
 - 4-phase handshaking, 298
 - active pullup, 301
 - address space, 32, 306, 327
 - arbiter, 290
 - arbiter interface block, 290, 295
 - bandwidth, 20, 334, 352
 - burst-mode, 16, 33
 - connector standard, 29–32
 - convolution chip, 336
 - daisy-chain, 324
 - decoder, 12, 299
 - encoder, 12, 13, 17, 297
 - fair arbiter, 293
 - frame conversion, 311, 312, 319, 320
 - greedy arbiter, 293
 - infrastructure, Unix pipe analogy, 318
 - merge circuit, 324
 - pre-decoder, 300
 - protocol, 316, 317
 - receiver, 306
 - receiver pixel, 300
 - sender, 306
 - serial (SAER), 34, 324–329
- silicon retina, 40
- sniffer, 310
- split circuit, 324
- wired OR, 302
- wired OR pullup, 290
- word-serial, 32–33
- word-serial protocol, 332
- AER channel
 - arbitration, 23, 26, 27
 - capacity, 20–27, 335
 - collision probability, 23, 24, 26, 27
 - collisions, 23, 24, 26, 27
 - contention, 23, 27
 - dispersion, 20, 21, 23, 24, 26, 27
 - integrity, 20, 21
 - jitter, 354
 - latency, 20, 21, 24–27
 - load, 20, 23–25, 27
 - queuing, 23, 24
 - throughput, 20, 22, 25–27
 - timing error, 24, 25, 27
 - waiting time, 24
- AEREAR, *see* silicon cochlea
- AEX, 324–325, 326, 341
- ALAVLSI, 350, 358
- algorithmic event processing, 365–379
- algorithmic mapping, 315
- ALOHA, 27
- amacrine cell, 47
- amplifier, 239

- analog signal processing, 249, 253
 API, 355, 356, 360, 362
Application Programming Interface, see API
 APS, 51, 64
 arbitration, 13, 21
 greedy tree, 15
 multidimensional, 16
 none, 17
 ring, 15
 token, 14
 token-ring, 15
 tree, 14
 ARM, 382
 asymmetric C-element, 289
 ATIS, 46, 56–58
 audio processing, 378
 auto-associative memory, 132
 automatic gain control, 82
 automatic quality control, 82
 axon, 12, 13, 104, 158, 164
 Axon-Hillock circuit, 59, 60, 159, 161, 162, 170
 axonal delays, 11, 315

 back side illumination, 65
 backlight display flicker, 62
 bandwidth
 AER, *see AER, bandwidth*
 software, 353
 basilar membrane, 72, 76
 Bernoulli-Cell, 158
 bias decoupling, 275–278
 bias generator, 261–282, 351, 357
 biases, 350, 351, 357, 361, 362
 binary STDP circuit, 212
 binaural auditory processing, 378
 bistable spiking network, 143
 bistable synapse, 122, 123, 127, 133, 134, 208
 bootstrapped mirror, 263
 bounded synapse, 121, 122
 Brian, 360, 362
 buffer, bias, 278
 bundled-data, 28
 bus mastering, 307, 319
 bypass capacitor, 277, 278
 calibration, 352, 362
 capacitive turn-on, 300
 CapoCaccia workshop, 2, 350, 403
 CAVIAR, 32, 318, 323, 335–340, 341, 344, 350, 358, 359
 PCI-AER, 319–320
 SimpleMonitorUSBXPress, 323
 USB-AER, 320
 USBAERmini2, 320–323
 central pattern generator, 92
 circuits, 98
 characteristic frequency, 73, 76
 ChipDatabase, 357–359
 circuit switched network, 10
 cochlea
 binaural, 84
 cascade, 76, 77
 characteristic frequency, 80
 scala media, 79
 tectorial membrane, 73
 cognition, 402
 collisions, 21
 comedi, 358
 Communicating Hardware Processes, 286
 communications theory, 23
 compatible lateral bipolar transistor, 231, 232
 conductance-based, 188
 silicon neuron, 166
 silicon synapse, 198
 configurable delays, 316, 320
 content-addressable memory, 383
 Convolutional Networks, 342
 cortex, 9, 48, 185, 186, 318, 331, 333, 367, 393, 397–400, 403
 CPG, 93
 CSMA, 14, 27
 current reference, 262
 current splitter, 263, 267–274
 current-mirror integrator, 191
 current-mode, 158

 daisy-chain AER, 324
 DARPA grand challenge, 1
 data acquisition, 306, 354, 357, 358
 data-width, 307

- database, 351, 352, 357
DAVIS, 64
debugging, 306, 353
demultiplexing, 12
dendrite, 105, 158
Dick Merrill, 65
diff-pair integrator (DPI), 159, 196
 filter, 172
 neuron, 172
differential signaling, *see* serial differential signaling
digital signal processor, 369
digitally-adjustable size factor, 165
direction selective cell, 373
discounted delay, 312
DMA, 307, 319, 354
DPI, *see* diff-pair integrator (DPI)
DPRAM, 314
DVS, *see* dynamic vision sensor (DVS)
dynamic vision sensor (DVS), 41–46, 335, 370
 pixel, 54
early packet, 310
embedded implementation, 369–370
encapsulation, 358
event
 annotation, 367
 object, 367
 packet, 367
event-driven, 366
excitatory post-synaptic current (EPSC), 190
FACTS, 341, 350, 361, 388
fan-out, 10, 30, 306, 313, 350
feedback
 cortical, 399
 negative, 400
 positive, 399, 401
fictive locomotion, 94
field programmable analog array, *see* FPAAs
FIFO, 307, 310–313, 317, 318, 322, 325, 369
filter
 gain, 222
 log-domain, 227
phase, 222
small-signal stability, 223
stability, 223, 224
firmware, 272, 282
fixed pattern noise, 60
floating-gate
 central pattern generator chip, 104
charge, 242
circuit, 238
electron tunneling, 107, 242
hot-electron injection, 242, 244
synapse, 107
flow control, 325
follower-integrator, 158
FPAAs, 238, 251, 342
FPGA, 313–315, 320, 324–328, 336, 340–342, 353, 356
frame-based vs. frame-free, 40
framing errors, 308
functional biomimesis, 92
Gabor chip, 324, 332
Gabor convolution, 334
gain, 399
GALS, 384
Gene’s law, 247
generalized integrate-and-fire neuron circuits, 170
global shutter, 58
graphical user interface, *see* GUI
gray matter, 9
greedy arbiter, 15
GUI, 351, 357, 360, 362
half-center oscillator, 94
half-wave rectification, 233
handshake
 abandoned, 34
 four-phase, 28
heartbeat, 310
Hebb rule, 129, 205, 209
Hebbian learning chip, 336
HiAER, 341, 384–386
HICANN, 341, 388–390
HMAX, 331, 342
Hodgkin–Huxley model, 103, 158, 166, 397

- homeostatic plasticity, 159, 198
 hop-count, 20
 Hopfield network, 123, 139
 horizontal cell, 47
- IFAT system, 329–332, 334, 384
 impact ionization, 242
 information hiding, 358
 inhibitory neuron, 339
 inner hair cells, 81, 233
 integrate-and-fire neuron, 102, 103, 112, 140, 157, 159, 169, 336, 338, 385, 388, 390
 intelligent transportation systems, 378
 inter-spike interval, 308, 311
 interaural time difference (ITD), 86, 378
 interrupt, 307, 317, 319, 320
 ion channel, 157
- jAER, 353, 359, 367, 370–372, 374, 378
- kTC noise in pixel, 58
- latency
 AER channel, *see* AER channel, latency
 AER device driver, 354
 AER serial interface, 325
 AER software processing, 352
 AER stream handling, 353
 AER UDP and TCP channels, 356
 algorithmic event processing, 369
 USB interface, 369
- layout, bias generator, 279
 layout, DVS pixel, 54
 leakage current, *see* off current
- log-domain
 circuits, 158
 current-mode filter, 193
 filter, 227
 integrator synapse, 194
 low-pass filter, 158
 LPF neuron, 171
- logarithmic encoder, 17
 logic analyzer, 306, 310
- long-term
 depression (LTD), 209
 plasticity, 203
 potentiation (LTP), 208
- look-up table, 313, 326, 356
- mapper, 19, 306, 312–316, 317, 320, 325, 353, 356, 357
 algorithmic, 315
 configurable delays, 316, 320
 High-Fanout, 325–327
 probabilistic, 315, 320
 sentinel value, 314
- master bias circuit, 263–267
- maximum spike rate, 25
- memory
 capacity, 123
 lifetime, 124
 management, 357
 map, 367
 states, 142
 merging, 307, 319, 324
- mesh routing, 20
- metaplastic
 states, 128
 synapses, 127
- metastability, 292
- microcontroller, 369
- mismatch, 40, 43, 48, 50, 56, 58, 60–62, 66, 67, 351
 bias current, 264, 272, 277, 281
 modeling neural circuits in locomotor control, 95
- monitor, 306, 307–311, 317, 320, 322–324
 synchronization, 309, 322
- Moore’s law, 366
- motion artifact, 49, 58
 motion feature, 373
 Muller C-element, 289
 multicast routing, 387
 multichip systems, 30, 329–340
 multiple sender – multiple receiver, 30
 multiplexing, 12
 multistate synapses, 127
 mutual exclusive element, 291

- NEST, 360
network attractor, 137
 states, 140, 142
NeuFlow, 342
neural network simulators, 156
neural population
 active fraction, 21, 22, 25, 27
 sampling rate, 21, 22, 27
Neurogrid, 386–388
NeuroML, 351
NEURON, 360
neurotransmitter, 186
noise removal, 370, 371
nonideality, 272, 281
nonvolatile storage, 249

object tracking, 336
Octopus retina, 48, 59, 329
off current, 270, 274, 281
open-source design kit, 263, 281
optical flow, 373
optimization, 354
Organ of Corti, 73
orientation feature, 50, 372, 373
orientation-selective multichip system, 334
ORISYS, 324, 332–335
outer hair cells, 73, 75, 81

packet switched network, 11
packet, event, 367
parameter control of multineuron chip, 318
parameters, 350, 351, 361, 362
parasitic capacitance effect, 55, 58, 265,
 275–277
parasitic photocurrent effect, 65, 277, 279,
 280
Parvo-Magno silicon retina, 40, 46–48,
 332
PCI, 315, 317, 319, 320, 326, 327, 341
PCI-AER
 CAVIAR, 319–320
 Rome, 317–318, 319, 341, 343, 357
PCSIM, 360, 362
pencil balancer robot, 369
perceptron, 129, 130, 135
phase dependent response characteristics,
 100
photodiode dark current, 63
photoreceptor, 66
placement software, 350
point-to-point communication, 19, 28
Principles of Brain Technology, 395
probabilistic mapping, 320
profiler, 355
programmable analog technology, 249
projective field, 315
PTAT, 264
PVT, 262
pyNCS, 361–362
PyNN, 352, 360
Python, 360–363
pyTune, 361, 362

quadratic integrate-and-fire neuron, 174
quality factor, 74, 85, 222
queuing, 21, 27
queuing theory, 24

rate mode, 400
real-time, 156
receptive field, 331, 332, 372
 auditory spatiotemporal, 341
reconfigurability, 306
region of interest, 40, 62
resistive network, 79
resonator, 74, 81
retina, 331
 bipolar, 47
 horizontal cell, 40
 inner plexiform layer, 39
 outer plexiform layer, 38
retinal ganglion cells, 332
robot, 335, 369
routing, 19–20, 34, 318, 324, 328,
 383–390
software, 350
topologies, 19

SATA, 325, 327, 341
SCX, 30, 315, 316–317, 342

- second-order
 filter – log domain, 230
 filter – voltage domain, 220
 low-pass filter, 222
 section, 77
 sequencer, 306, 311–312, 317, 320, 322–324
SerDes, 325, 327, 329
 serial AER (SAER), 34, 324–329
 serial differential signaling, 33–34, 329
Serializer–Deserializer, *see SerDes*
 shape recognition, 378
 shifted-source biasing, 274–275
 short-term plasticity, 201
 signal processing, 248
 silicon central pattern generator, 108
 silicon cochlea
 1D, 76–78
 2D, 80
 AEREAR, 84, 85, 370
 silicon neuron, 104, 156, 160
 silicon photoreceptor, 54
 silicon retina, serial analog output, 40
 silicon synapses, 188
SimpleMonitorUSBXPress, 323
Simulink, 342
 sniffer, 310
 soft winner-take-all, *see sWTA*
 soma
 silicon, 157, 168
 sound localization, 86
 spatiotemporal integration, 157
 spike timing, 130
 spike timing dependent plasticity, 130, 131,
 133, 206, 342
 Spike Toolbox, 359
 spike-based
 learning circuits, 203
 learning rule, 134
 plasticity, 198
 spike-frequency adaptation, 105, 162, 163,
 172
 spiking threshold, 162
SpiNNaker, 328, 361, 382–384
 splitting, 307, 319, 324
SRAM, 314, 318, 320, 341
 stereo vision, 370
 stimulation, 306
 stochastic learning, 132
 stored pattern, 139
 substrate leakage current, 62
Switched-Capacitor Mihalas–Niebur neuron,
 175
 switched-capacitor synapse, *see synapse*
Switched-Capacitors, 174
sWTA, 403
 synapse
 biological, 185
 building blocks, 157
 cascade model, 128
 dynamic plastic, 201
 floating-gate, 107
 log-domain integrator, 194
 long-term plasticity, 203
 NMDA, 200
 short-term plasticity, 203
 silicon, 157
 summing exponential, 192
 switched-capacitor, 199
 weight update, 203, 212
 synaptic
 dynamics, 186
 efficacy, 159
 modification, 126
 tau cell, 228
Tau-Cell neuron, 170
TCDS, 58
 Telluride workshop, 2, 350, 403
 temperature, 261–266
 temporal contrast, 55
 temporal contrast sensor, 41
 tempotron, 130
 Thalamic relay neuron, 168
 time
 continuous, 11
 discretized, 11
 representation of, 11
time represents itself, 11
 time to first spike, *see TTFS*
 time-base, 309, 312
 time-domain true correlated double
 sampling, 58

- time-multiplexing, 11
time-stamp, 11, 308, 310, 317, 320, 322, 353, 360
timing error, 21
tracker, visual, 374–377
tradeoffs, 21–27
transconductance, 263
transconductance amplifier, 158, 220
translinear
 loop, 227, 228
 principle, 159, 193
tree routing, 20, 386, 387
TTFS, 50
two-tone suppression, 82
uncorrelated memories, 125
USB, 44, 319, 320, 322, 323, 341, 369
USB-AER
 CAVIAR, 315, 320
SimpleMonitorUSBXPress, 323
USBAERmini2, 320–323
vector-matrix multiplication, 250
virtual circuit, 10
ViSe, 50–53, 58
white matter, 9
winner-take-all, 336, 338
 soft, *see* sWTA
wrap-around, 309
X first routing, *see* mesh routing
Xilinx, 341
XML, 357
XY routing, *see* mesh routing
zero moment point, 114

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.