Wiring Considerations in Analog VLSI Systems, with Application to Field-Programmable Networks

Thesis by

Massimo Antonio Sivilotti

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California USA

1991

(Defended July 17, 1990)

6.3 A Single-Phase Latch

An essential component of traditional sequential logic design is the synchronous latch, a storage element whose state changes only at clock *transitions*. Typically, such latches are placed between logic elements that are strictly combinational; clocking the latches with a single-phase clock provides a hazard-free design discipline.

Unfortunately, at the transistor level, devices are *level* sensitive, not edge sensitive. In conventional digital VLSI design, this constraint has given rise to design disciplines involving multiple clock phases. Typically, each clock phase enables resynchronization registers. In practice, these registers are often as simple as dynamic state-storage nodes gated by pass transistors. This explicit *pipelining* gives rise to constraints on each of the clock phases; this procedure often places severe demands on the design tools (since the clocking discipline becomes an integral part of the specification of the interface between circuit subsystems). Also, there are serious practical difficulties involved in distributing multiple phases (i.e., limiting clock skew), and additional circuit overhead involved in computing derived clocks.

Thus, although level-sensitive clock disciplines are very flexible, and ultimately hold the promise of maximum performance, they are needlessly complicated for many applications. In particular, novice designers would find extremely attractive the ability to apply familiar design techniques involving edge-triggered latches.

In Sections 6.3.1 to 6.3.2, we describe a new single-phase latch design. It is relatively small, comprising 14 transistors, yet can be designed to be risk-free. Although similar to master–slave designs, it is a true edge-sensitive implementation: The output becomes available immediately, and is not delayed until the complementary clock transition, as in traditional master–slave designs. This cell can be employed as either a D flip-flop or a T flip-flop. It has been used successfully in various systems, including scanning and self-timed retina design frames, and an asynchronous up–down

counter for neural integrators.

6.3.1 A Level-Sensitive Latch

The edge-sensitive latch is based on complementary set–reset logic (CSRL) (see Figure 6.10 and [MW85]). A CSRL stage consists of two cross-coupled inverters, with a power-up transistor Q_3, and two access devices Q_1 and Q_2. In the following discussion, complementary inputs are assumed. When the clock ϕ is high, the flip-flop is disconnected from the V_{dd} rail, and the internal state is imposed by the environment through the access devices. In this condition, that the inverter pair is able to restore *one* of its internal state nodes, because the Q_6 and Q_7 devices are always connected to ground, and hence can pull down *one* of nodes Q and $\overline{Q}$. This ability plays an important role in our discussion of the operation of the edge-triggered latch.

When the clock goes low, the access devices are disabled, and the power-up device Q_3 turns ON, making the latch fully static. The conventional application of CSRL [Waw86, Waw87] employs pipeline stages of CSRL latches, clocked by a two-phase nonoverlapping clock. The principal virtues of this approach are (1) only two clock phases are required (no locally generated complements are needed), (2) operation is fully restored and fully static, and (3) dense computational primitives are available that exploit the dual-rail signal inputs (and a general switching form described in [MW85]).

6.3.2 An Edge-Sensitive Latch

Instead of the use of two identical CSRL stages clocked by two nonoverlapping clock phases, we propose the use of two *complementary* stages clocked by the same clock (see Figure 6.11). This strategy is similar to the π-latch–μ-latch approach proposed by Denyer [MDM87]; the key difference is that when we employ CSRL stages we

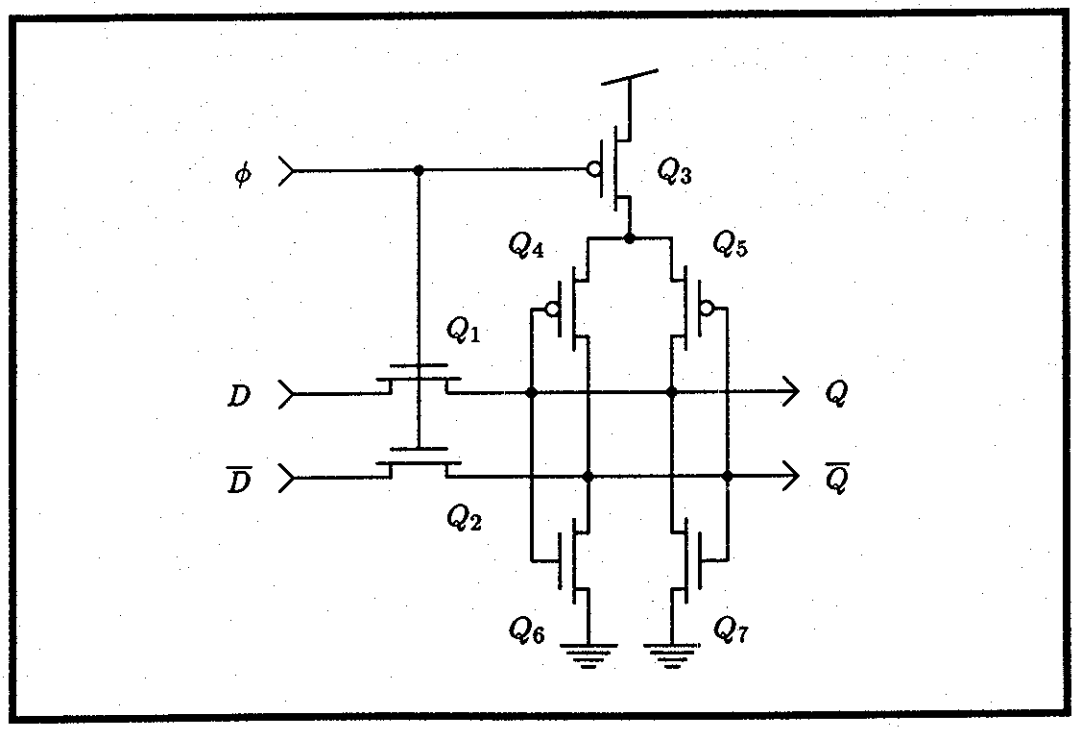


Figure 6.10: A single CSRL stage. When power is removed from the cross-coupled inverters Q_4/Q_6 and Q_5/Q_7, the access devices Q_1 and Q_2 are enabled, and can drive the external state $D/\overline{D}$ into the latch. When the clock goes low, the drive from the access devices is removed, and Q_3 turns ON, making the latch fully static.

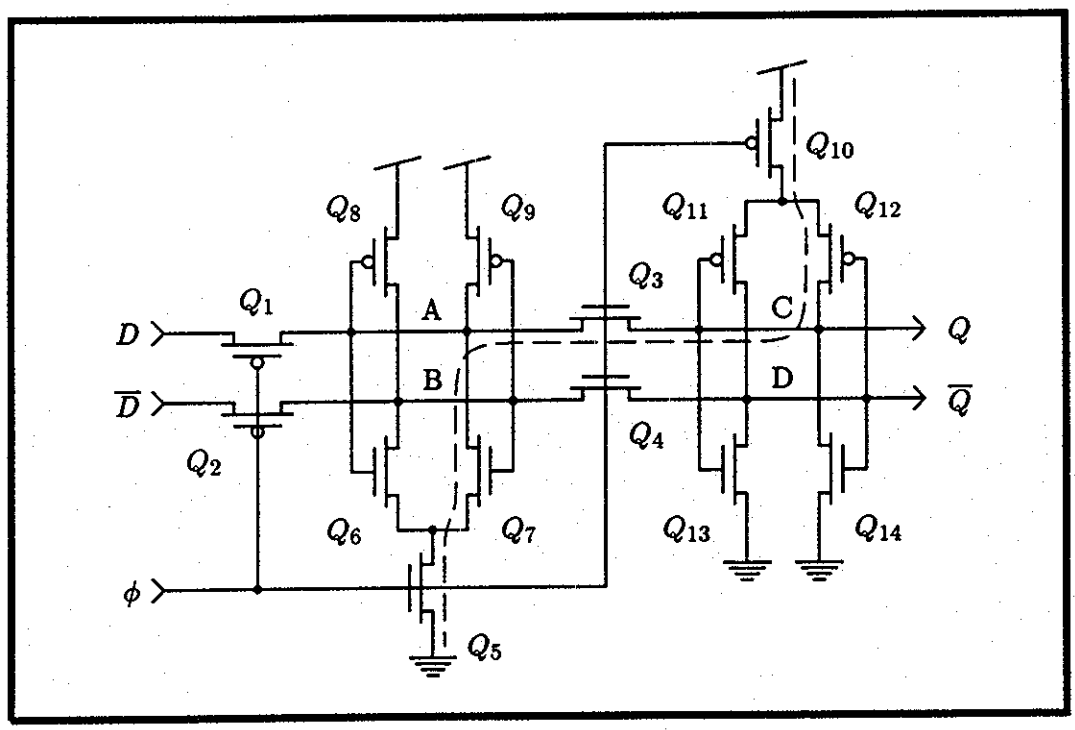


Figure 6.11: An edge-sensitive CSRL latch.

require only 14 transistors for fully static operation, instead of the 32 Denyer's design uses. In fact, our transistor count is quite competitive with the 10 transistors used in their *dynamic* design, especially considering the concomitant advantages of dual-rail signal representations.

The operation of the latch is conceptually simple: When the clock is low, the right-hand stage is powered, and is isolated from the left-hand stage. The left-hand stage, on the other hand, is driven by its inputs. When the clock goes high, the influence of the inputs is removed, and the left-hand stage latches its state. Furthermore, pass transistors Q_3 and Q_4 permit the new state to be propagated to the right-hand latch (and to the outputs); when the clock goes low, the outputs are restored to the rails, and the left-hand stage becomes sensitive to its inputs again.

6.3.3 Analysis of the Edge-Sensitive Latch

A more careful analysis is required to guarantee the direction of data transfer described in Section 6.3.2. A potential difficulty arises because pass transistors are inherently bidirectional elements, and the current state latched in the right-hand latch could corrupt the "next-state" data to be latched from the left, during the conduction overlap period as the clock rises with a finite rise time. The crucial observation that facilitates the analysis of the system is that, although CSRL stages can be thought of being powered down, they are, in fact, always capable of full current drive toward *one* of the power rails.

Referring again to Figure 6.11, we observe that, as ϕ rises, Q_3 and Q_4 begin to turn ON. There are four state variables we must consider; without loss of generality, suppose the initial conditions are $V_C = 5$ V, $V_D = 0$ V, $V_A \approx V_T$, and $V_B = 5$ V. Consequently, Q_8 drives node V_B high, just as Q_{13} pulls node V_D low. Thus, we can limit our consideration to nodes V_A and V_C, whose drives are a function of the clock voltage. Initially, V_C is driven high by Q_{10}–Q_{12}. Node V_A is the critical node: It is pulled up by Q_3, and pulled down by Q_7–Q_5 (it is also pulled down by Q_1, but, to be conservative, we ignore this current).

Q_3 is always saturated, so above threshold we have [Vit89]

$$I_{Q_3} = \left(\frac{W}{L}\right)_{Q_3} k_n(\phi - V_{T_n} - nV_A)^2$$

Assuming Q_7 is ON and is ohmic, the current out of node V_A is limited by the current through Q_5:

$$I_{Q_5} = \left(\frac{W}{L}\right)_{Q_5} k_n \left(2(\phi - V_{T_n})V_A - V_A^2\right)$$

Clearly, a conservative *sufficient* condition for the correct operation of the latch is

$$\begin{aligned} I_{Q_5} &> I_{Q_3} \\ \Rightarrow \quad \frac{(W/L)_{Q_5}}{(W/L)_{Q_3}} &> \frac{(\phi - V_{T_n} - nV_A)^2}{2(\phi - V_{T_n})V_A - V_A^2} \end{aligned} \tag{6.1}$$

We select a noise margin for safe operation (e.g., we allow V_A to rise to 1.5 V), and we consider the range of possible voltages for ϕ. For a typical 2μ CMOS process, Figure 6.12 indicates safe values for the access-transistor geometry ratio.

Clearly, this analysis is conservative in several respects. First, no provision is made for decreased current drive from the right-hand stage as the clock rises (and the drive of Q_{10} decreases). Second, a conservative noise margin of approximately 1 V has been chosen for V_A (in practice, V_A could be allowed to rise to about 2.5 V before failure; only when V_A and V_B cross over does actual failure occur). Third, no account was taken of the additional current drive through access device Q_1, which acts to decrease the likelihood of failure.

This analysis also presents an attractive *intuitive* explanation of why the single-phase latch works: The conflict is between transistors *of the same type*. Thus, we can guarantee safe operation, effectively independent of fine details of process parameters and variations, simply by elongating the access pass transistors (a ratio of $(W/L)_{Q_5}/(W/L)_{Q_3} = 4$ is typically chosen, and has been observed experimentally to be safe over the current range of MOSIS processes). Finally, because only transistors of the same type are ever in conflict, it is safe to alternate n-CSRL with p-CSRL stages (e.g., to construct shift registers).

6.3.4 Single Phase Toggle Flip-Flops

We can exploit this closure property, which allows composition of alternate n- and p-CSRL stages, in a more direct way: By connecting the D and Q (and $\overline{D}$ and $\overline{Q}$)

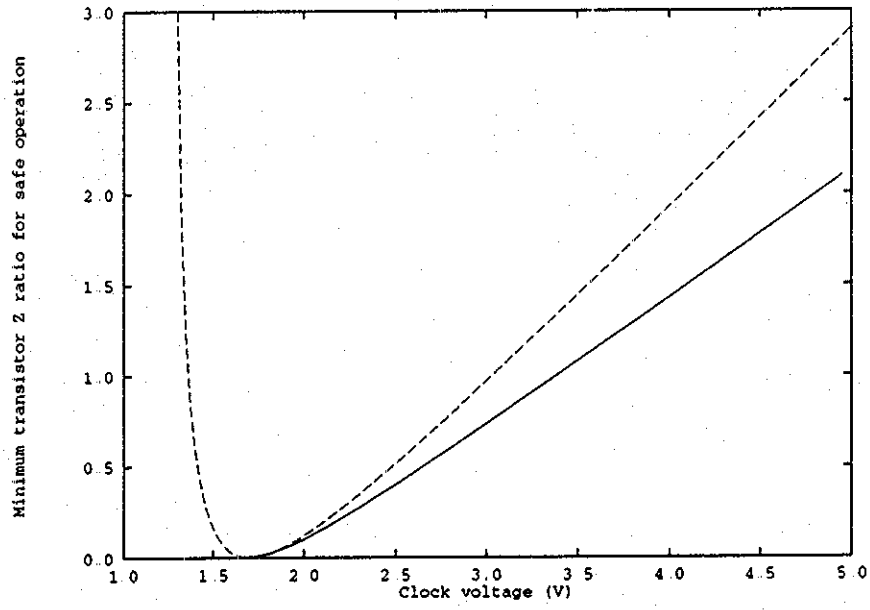


Figure 6.12: Minimum geometry factor for safe operation of the edge-sensitive latch. These values assume a typical CMOS process with $V_{T_n} = 1$ V, and $n = 1.4$. The dashed line is for Equation 6.1; the behavior at low voltages is an artifact, as the assumption of above-threshold saturation operation is violated. The solid line is obtained by applying a continuous model (above and below threshold, ohmic and saturation regimes) developed in [Vit89], and repeating the analysis of Section 6.3.3. No account is taken of capacitive coupling between the access devices' gate–source overlap capacitance with the diffusion capacitance of node V_A (typical geometries reveal less than 0.2 V coupling, however).

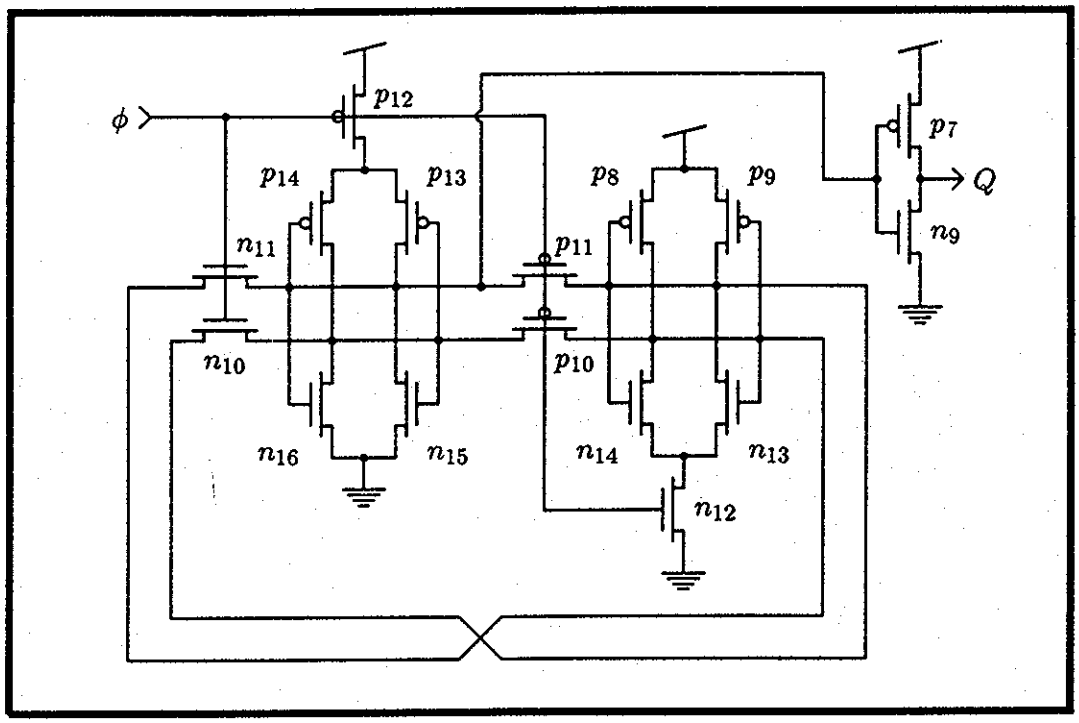


Figure 6.13: A toggle flip-flop cell using the single-clock CSRL discipline. The output inverter is included for increased drive, sharpening the output signal, and isolating the electrical environment within the cell. A typical T-flop would have two such buffered outputs (Q and $\overline{Q}$).

nodes of a *single* stage, we obtain a toggle flip-flop (see Figure 6.13). To maintain the electrical symmetry of the nodes within the T-flop, we buffer the outputs of the cell with inverters. The analysis for correct operation is essentially unchanged from that in the previous section. We have observed experimentally that, with the addition of the buffer inverters, a T-flop stage output can reliably drive the toggle-clock input of a subsequent stage (see Figure 6.14). In fact, the operation of the toggle cell is so robust that even very small clock signals can be used to toggle the cell (see Figure 6.15).

CH1 DC 2V 1ms AVG; CH2 DC 2V 1ms AVG;

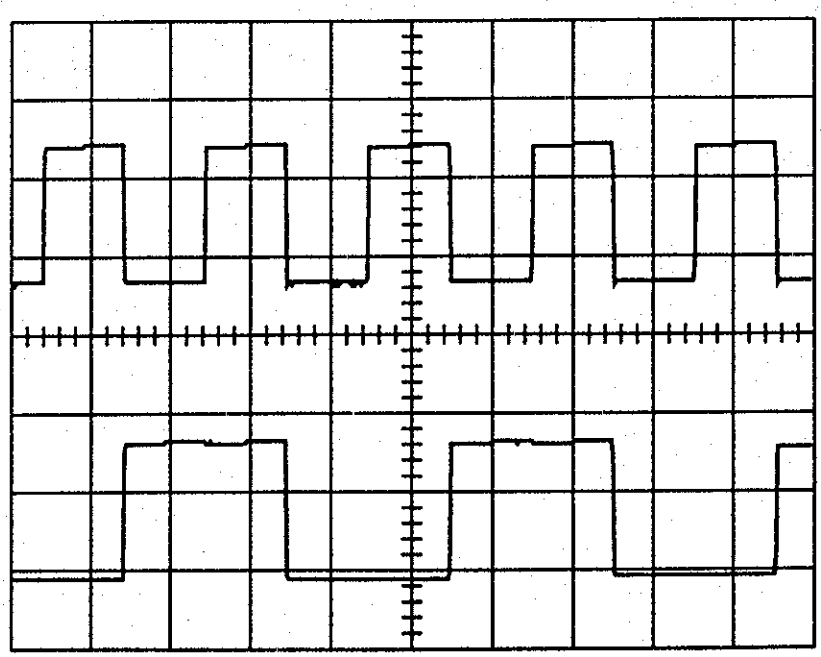


Figure 6.14: Two-bit binary counter implemented with toggle flip-flop cells. These experimental data are the outputs of the first two stages of a binary ripple counter.

CH1 DC 2V 5us AVG; CH2 DC 2V 5us AVG;

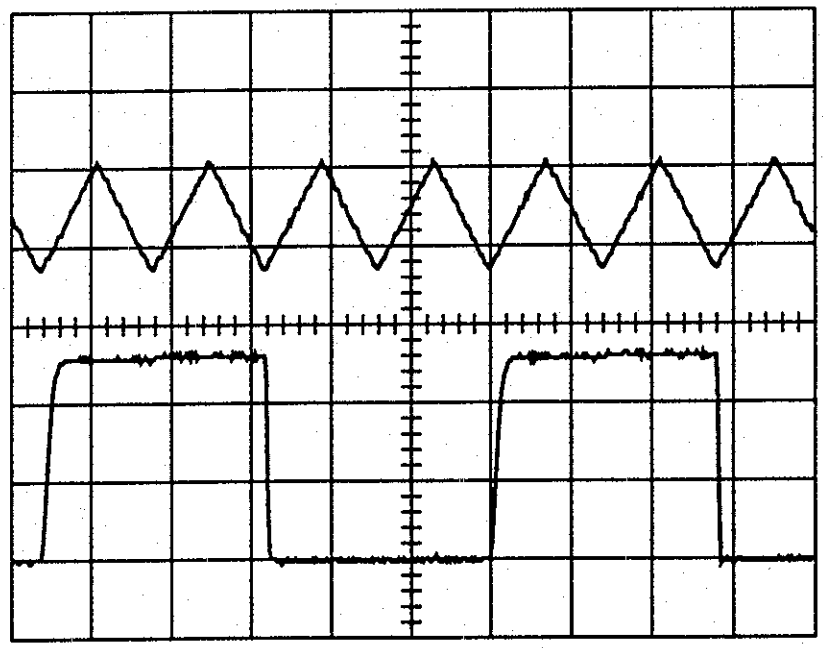


Figure 6.15: Nonrestored clock applied to toggle flip-flop cell. These experimental data illustrate the robustness of the toggle flip-flop cell against variations in clock rise- and fall-time, and in amplitude of clock waveform. The output is from the *second* stage of a binary ripple counter.

6.3.5 Asynchronous Up–Down Counters

A direct application of the edge-triggered toggle flip-flop cell is a binary ripple counter. Either an up-counter or a down-counter can be constructed, by selecting the $\overline{Q}$ or Q output (respectively) of a stage to drive the clock input of the next stage. In this section, we describe two interesting extensions to this procedure, to construct a universal cell for Gray code counters, and for combining two binary up-counters to provide an asynchronous up–down counter.

A Scalable Gray-Code Counter

A well-known disadvantage of binary ripple counters is their passage through spurious states as the carry signal propagates down the chain. For example, in going from the 0111 state to the 1000 state, a ripple counter will (probably) pass through the states 0110, 0100, and 0000. This behavior is unattractive for asynchronous systems (e.g., D/A converters) that are always sensitive to transitions in their inputs. Note that this behavior is independent of the counting *rate*, and is fundamentally different that the high count-rate dynamic instability of multiple carries propagating simultaneously down the chain.

Systems that depend on noise-free transitions between successive states, such as shaft-encoders, rely on Gray-code encodings, where adjacent states differ by exactly one bit. Figure 6.16(a) illustrates this code; Gray code can be converted to binary by exploiting the reflected-binary nature of the code (Figure 6.16(b)).

We can construct a Gray-code counter by observing the following property of the Gray-code sequence: At every clock transition, we flip the least-significant bit that takes us to a state that has not been already visited. Figure 6.17 shows an even more direct interpretation: The bit that changes is the one that describes the extent of carry propagation in the corresponding binary ripple counter. Figure 6.17b shows

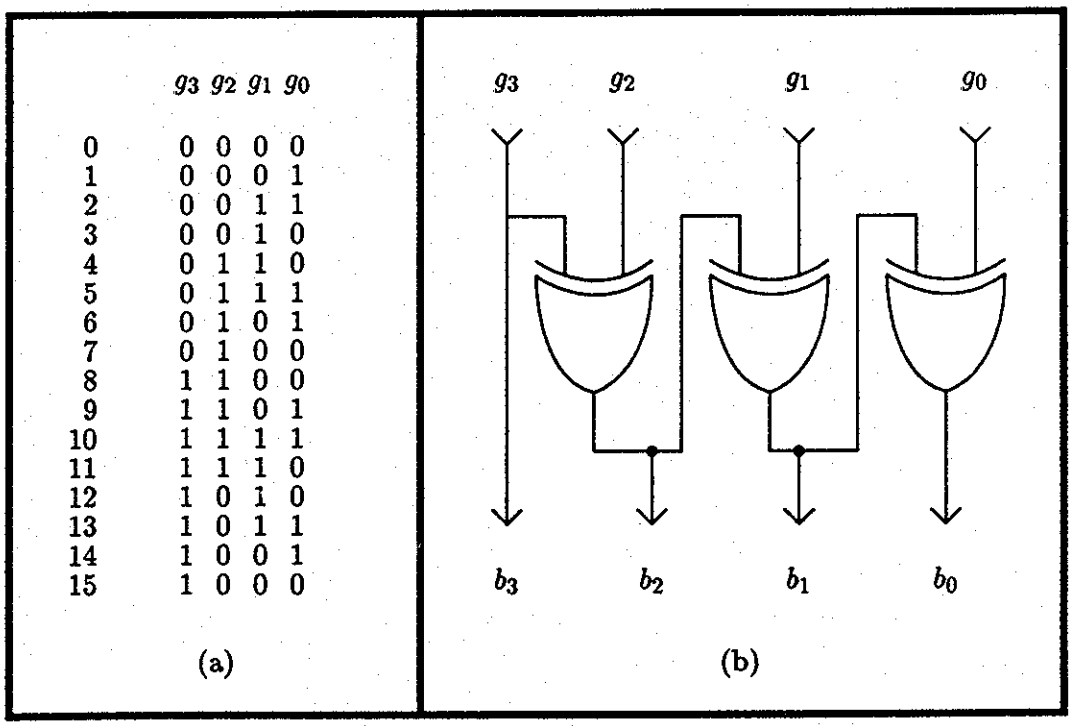


Figure 6.16: The Gray code. (a) Adjacent codewords differ by exactly one bit, thereby preventing spurious states between adjacent codewords. (b) Conversion of Gray code to binary, exploiting a recursive interpretation of the Gray code: If the most significant bit (MSB) of a subword is set, we must invert the remaining bits in the subword before interpreting the result.

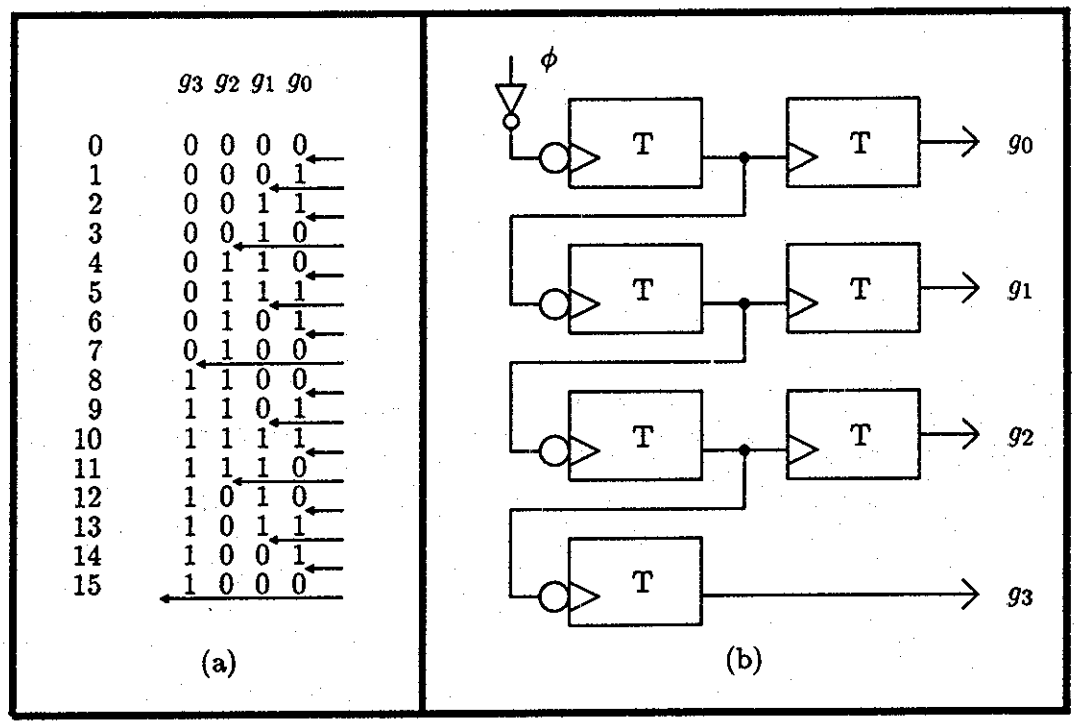


Figure 6.17: A scalable Gray-code counter. (a) A general algorithm for selecting which bit to toggle, when counting in a Gray code. (b)Use of a binary ripple up-counter to generate the toggle signal for that particular bit.

a direct implementation of this architecture; the result is an arbitrarily scalable Gray code counter, in which each stage is identical to the preceding stage (except for the last stage, which generates the most significant bit (MSB)). Sample output from a fabricated test structure is shown in Figure 6.18.

An Asynchronous Up–Down Binary Counter

We can synthesize an asynchronous $N-1$-bit up–down counter from two $(N-1)$-bit up-counters. Each $(N-1)$-bit counter can count in the range $[-M-1, M]$, where $M = 2^{N-2} - 1$. The problem we must solve is how to handle the wrap-around that occurs when each of these counters overflows.

Let U be the state of the up-counter, and $u = U_{N-2}$ be the value of the MSB of

CH1 DC 2V 2ms NORMAL; CH2 DC 2V 2ms NORMAL;

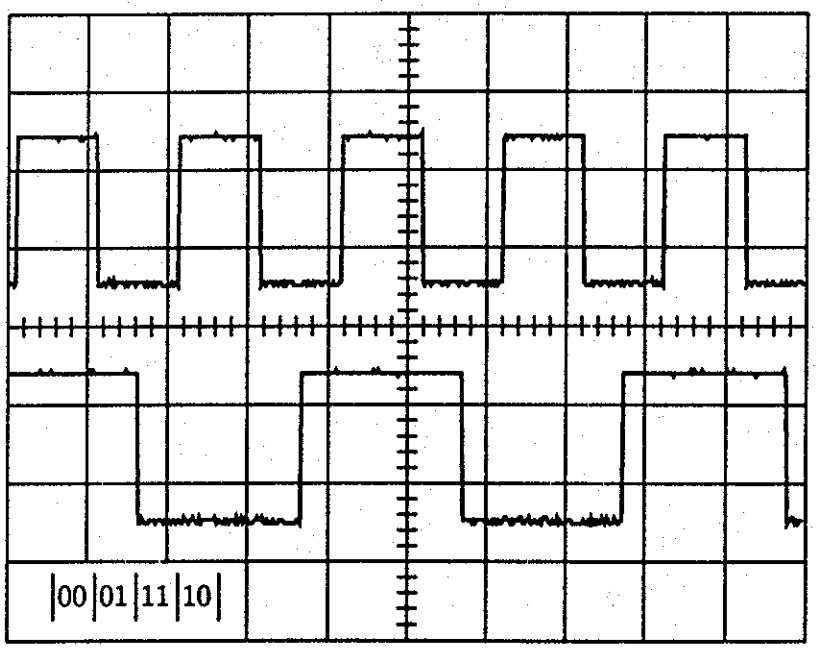


Figure 6.18: Sample output from a Gray-code counter. Both signals are logically inverted, and g_0 is displayed above g_1.

the up-counter. Similarly, D and d refer to the down-counter's state and MSB value. Let X be the difference between U and D, computed by a standard two's-complement $(N-1)$-bit subtraction (defined by $X = U + \tilde{D} + 1$, where "~" is the bit-wise logical negation operator).

We use an $(N-1)$-bit subtracter, and synthesized the MSB by a finite-state machine with u and d as inputs. For any given value in the up-counter, we can identify uniquely the correct difference if the down-counter's value lies in the range illustrated in Figure 6.19. If the down-counter's value lies outside this range, additional bits of state information are required—it would be simpler to extend the counters. Figure 6.19 also describes the interpretation that must be assigned to the $(N-1)$-bit difference X (i.e., the value of the MSB ($x = X_{N-1}$) required for the correct interpretation of the difference, in two's-complement notation).

We can reduce the state-transition diagram (Figure 6.20) to a Huffman flow table (Figure 6.21). If we ignore the possibility of overflow errors, and assume that only fundamental-mode operation is allowed (i.e., only single transitions of u, d, or x are permitted), this flow table collapses to that shown in Figure 6.22, where A is the union of states 1, 2, and 3, and B is the union of states 3, 4, and 5.

At this point, state encodings can be assigned, and Boolean logic minimization can be applied to obtain the following finite-state machine. For $S = \{A, B\} = \{0, 1\}$,

$$
\begin{aligned}
s^{\text{next}} &= u \cdot d + u \cdot s^{\text{present}} + d \cdot s^{\text{present}} \\
x^{\text{next}} &= \overline{u} \cdot \overline{d} \cdot x^{\text{present}} + \overline{u} \cdot d \cdot \overline{s} + u \cdot d \cdot x^{\text{present}} + u \cdot \overline{d} \cdot s
\end{aligned}
$$

If we want to detect overflow, the state flow table can be collapsed only to four states, and an additional state variable (w) is required:

$$
s^{\text{next}} = u \cdot d + u \cdot s^{\text{present}} + d \cdot s^{\text{present}}
$$

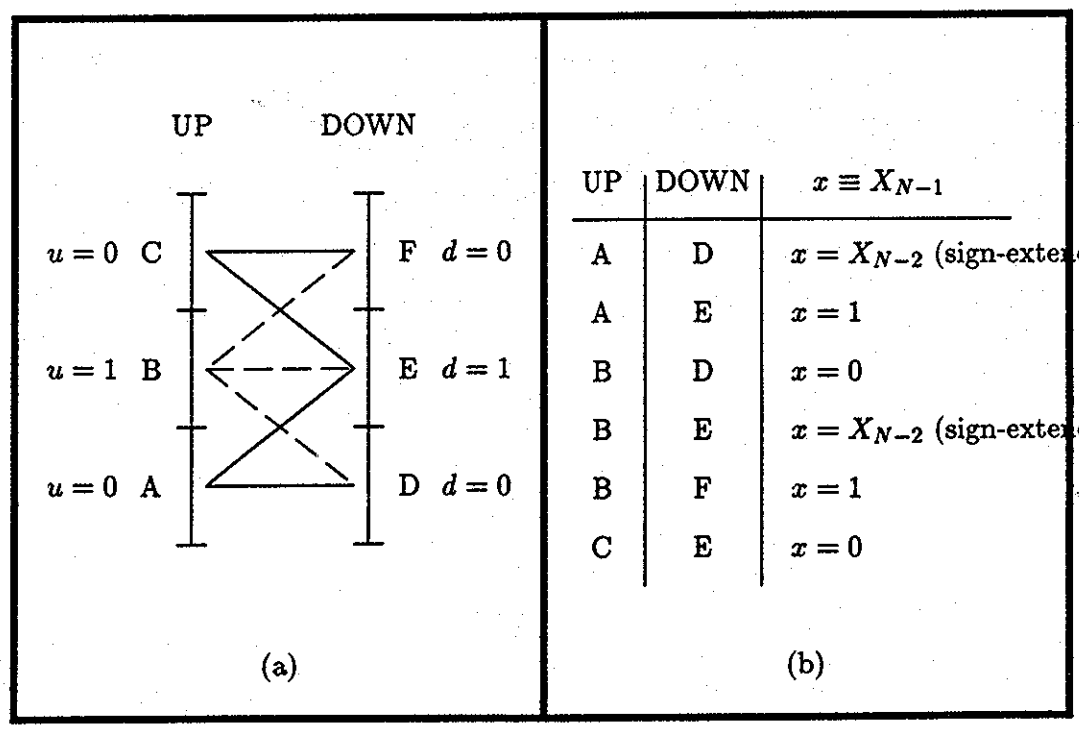


Figure 6.19: Synthesis of an up–down counter from two up-counters. We can interpret the difference between two up-counters uniquely, if we also keep track of in which subrange ($u = \{0,1\}, d = \{0,1\}$) the two up-counters are operating. In the worst case, the dynamic range of the synthesized difference output is $[-M, M]$, corresponding to $(N-1)$ bits of precision.

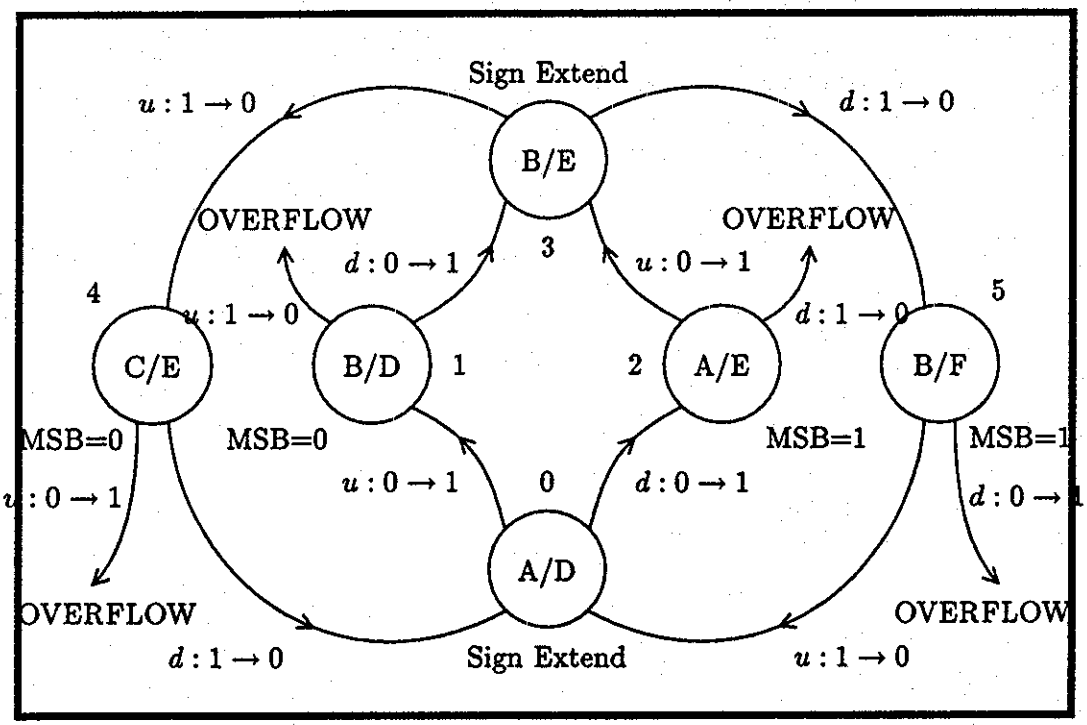
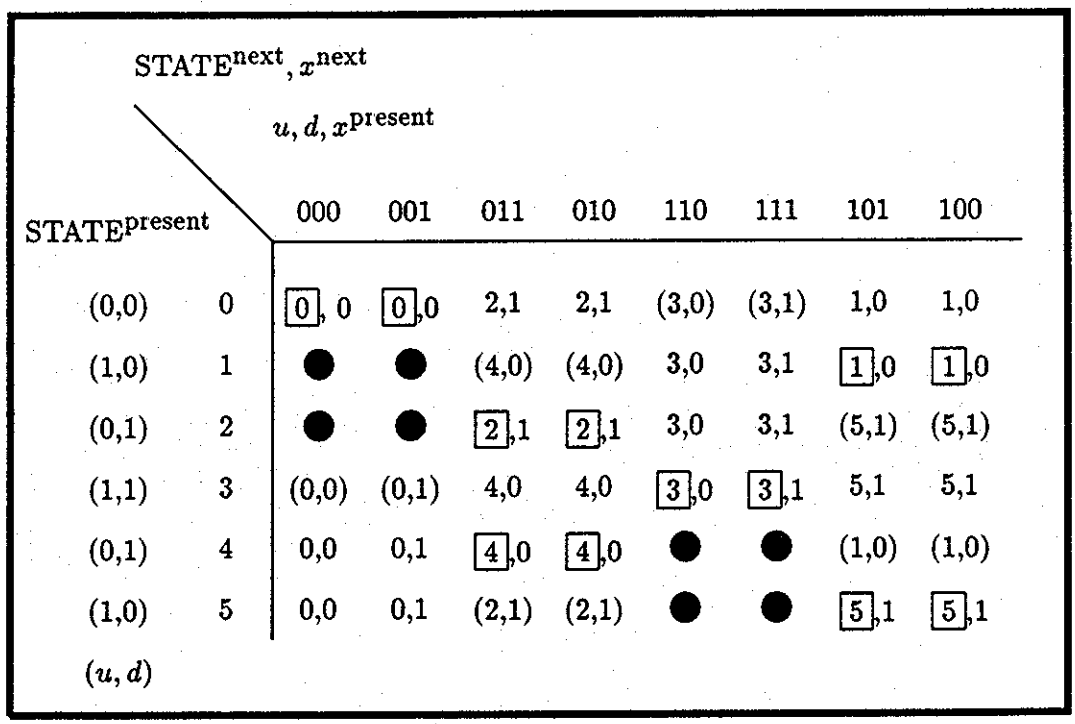
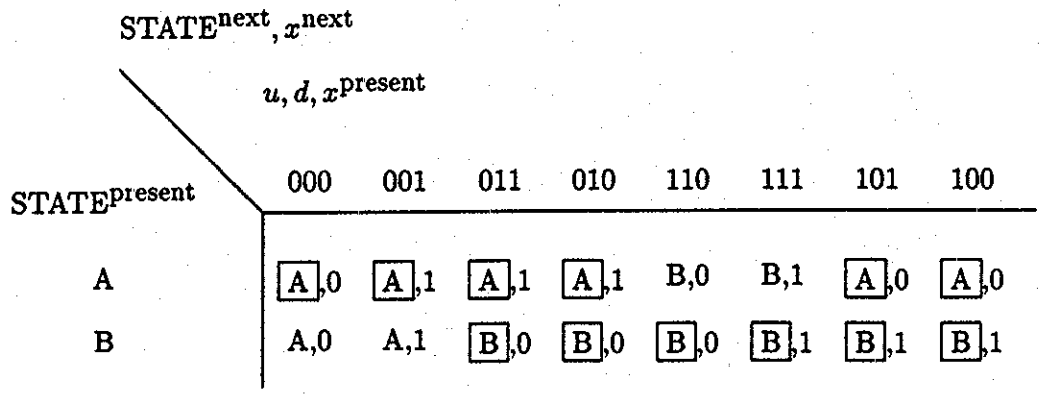


Figure 6.20: State-transition diagram for up–down counter. Transitions in the most significant bits (u and d) of the two up-counters are monitored; in conjunction with the current MSB (x), they determine the next state.



$STATE^{present}$		$STATE^{next}, x^{next}$ / $u, d, x^{present}$ 000	001	011	010	110	111	101	100
(0,0)	0	[0], 0	[0],0	2,1	2,1	(3,0)	(3,1)	1,0	1,0
(1,0)	1	●	●	(4,0)	(4,0)	3,0	3,1	[1],0	[1],0
(0,1)	2	●	●	[2],1	[2],1	3,0	3,1	(5,1)	(5,1)
(1,1)	3	(0,0)	(0,1)	4,0	4,0	[3],0	[3],1	5,1	5,1
(0,1)	4	0,0	0,1	[4],0	[4],0	●	●	(1,0)	(1,0)
(1,0)	5	0,0	0,1	(2,1)	(2,1)	●	●	[5],1	[5],1
(u, d)									

Figure 6.21: Huffman flow table for up–down counter. This table captures all the transitions of Figure 6.20, as a function of the inputs u, d, and x. Transitions in boxes represent fixed points, whereas transitions in brackets constitute violations of the fundamental-mode assumption (i.e., only one input is permitted to change at a time). The dark circles correspond to the detection of an overflow condition (i.e., a state that cannot be represented with only a single bit of additional information (x)).



$STATE^{present}$	$STATE^{next}, x^{next}$ / $u, d, x^{present}$ 000	001	011	010	110	111	101	100
A	[A],0	[A],1	[A],1	[A],1	B,0	B,1	[A],0	[A],0
B	A,0	A,1	[B],0	[B],0	[B],0	[B],1	[B],1	[B],1

Figure 6.22: Simplified Huffman flow-table for up–down counter. We no longer attempt to detect the overflow condition.

$$
\begin{aligned}
w^{\text{next}} &= d \cdot \overline{s} + u \cdot d + u \cdot \overline{s} \\
x^{\text{next}} &= \overline{u} \cdot \overline{d} \cdot x^{\text{present}} + u \cdot d \cdot x^{\text{present}} + \overline{u} \cdot d \cdot \overline{s} + u \cdot \overline{d} \cdot s \\
\text{OVERFLOW} &= \overline{u} \cdot \overline{d} \cdot w + u \cdot d \cdot \overline{w}
\end{aligned}
$$

Thus, we have constructed an $N-1$ bit up–down counter with a guaranteed range of $[-M, M]$ by subtracting two $(N-1)$-bit counters. Because two independent up-counters are used, the resulting up–down counter is free to accept asynchronous UP and DOWN inputs. This property makes this up–down counter attractive for neural integrators of the sort used in bidirectional servomotor control [DM88].

References

[AH87] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*. Holt, Rinehart and Winston, 1987.

[ASS84] P. E. Allen and E. Sanches-Sinencio. *Switched Capacitor Circuits*. Van Nostrand Reinhold, 1984.

[BBB+89] D. Beatty, K. Brace, R. E. Bryant, K. Cho, and L. Huang. *User's Guide to COSMOS: A Compiled Simulator for MOS Circuits*. Carnegie-Mellon University, 1989.

[BL84] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

[BSW82] R. Bryant, M. Shuster, and D. Whiting. Mossim II: A switch-level simulator for MOS LSI, user's manual. Technical Report TR #5033, California Institute of Technology, August 1982.

[CG70] D.G. Corneil and C.C. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1):51–64, Jan 1970.

[CL81] D. Cohen and G. Lewicki. MOSIS—the ARPA silicon broker. In *Proceedings from the Second Caltech Conference on VLSI*, pages 29–44, Pasadena, CA, 1981. California Institute of Technology.

[Coo73] L. N. Cooper. In B. Lundqvist and S. Lundqvist, editors, *Proceedings of the Nobel Symposium on Collective Properties of Physical Systems*, pages 252–264, New York, 1973. Academic Press.

[Del89] T. Delbrück. A chip that focuses an image on itself. In C. Mead and M. Ismail, editors, *Analog VLSI Implementation of Neural Systems*, pages 171–188. Kluwer Academic Publishers, 1989.

[Del90] T. Delbrück. private communication, 1990.

[DK85] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design*, CAD-4(1):92–98, 1985.

[DM88] S. P. DeWeerth and C. A. Mead. A two-dimensional visual tracking array. In *1988 MIT Conference on Very Large Scale Integration*, pages 259–275, Cambridge, MA, 1988. MIT Press.

[EIA88] *Electronic Design Interchange Format Version 2 0 0, Recomended Standard EIA–548*, March 1988. Electronic Industries Association.

[FH85] D. Feinstein and J. J. Hopfield. private communication, 1985.

[FM82] C. Fiduccia and R. Mattheyses. A linear–time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[GF86] S. L. Garverick and B. Frankel. *LSH User Manual and Documentation*. M.I.T. Lincoln Laboratory, 1986.

[Gil84] David Gillespie. LOG, the Chipmunk logic simulator's user's guide. Technical Report 5130:TR:84, California Institute of Technology, 1984.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[GR80] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, 1980.

[Hay78] J. P. Hayes. *Computer Architecture and Organization*. McGraw-Hill, 1978. Sec. 5.3.3.

[HGSM81] K. C. Hsieh, P. R. Gray, D. Senderowicz, and D. G. Messerschmitt. A low-noise chopper-stabilized switched-capacitor filtering technique. *IEEE Journal of Solid-State Circuits*, SC-16(6):708–715, 1981.

[Hop82a] J. J. Hopfield. Collective processing and neural states. *Modelling and Analysis in Biomedicine*, 1982.

[Hop82b] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79:2554–2558, April 1982.

[Hop84] J. J. Hopfield. Neurons with graded response have collective properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, 81:3088–3092, May 1984.

[HSM82] William R. Heller, G. Sorkin, and Klim Maling. The planar package planner for system designers. In *19th Design Automation Conference*, pages 253–260, 1982.

[KGV83] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):2554–2558, 13 May 1983.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, pages 291–307, February 1970.

[KS78] S. Kirkpatrick and D. Sherrington. *Phys. Rev.*, 17:4384–4403, 1978.

[KS81] Eric R. Kandel and James H. Schwartz. *Principles of Neural Science*. Elsevier/North Holland, Amsterdam, 1981.

[Lau79] Ulrich Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. In *16th Design Automation Conference*, pages 1–10, 1979.

[Laz86] John P. Lazzaro. anaLOG: A functional simulator for VLSI neural systems. Master's thesis, California Institute of Technology, Computer Science 5229:TR:86, 1986.

[Lin86a] R. Linsker. From basic network principles to neural architecture: Emergence of orientation-selective cells. *Proc. Natl. Acad. Sci. USA*, 83:8390–8394, 1986.

[Lin86b] R. Linsker. From basic network principles to neural architecture: Emergence of spatial-opponent cells. *Proc. Natl. Acad. Sci. USA*, 83:7508–7512, 1986.

[LR71] B. Landman and R. Russo. On a pin vs. block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, pages 1469–1479, December 1971.

[Mah89] M. A. Maher. *A Charge-Controlled Model for MOS Transistors*. PhD thesis, California Institute of Technology, 1989.

[Mah90] M. A. Mahowald. private communication, 1990.

[MC80] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[MDM87] M. S. McGregor, P. B. Denyer, and A. F. Murray. A single-phase clocking scheme for CMOS VLSI. In *1987 Stanford Conference on Very Large Scale Integration*, pages 257–271, Cambridge, MA, 1987. MIT Press.

[Mea89] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[Mea90] C. A. Mead. Transactions of IEEE, Special Issue on Neural Networks (in press)., 1990.

[MG75] J.L. McCreary and P.R. Gray. All-MOS charge redistribution analog-to-digital conversion techniques—part I. *IEEE Journal of Solid State Circuits*, SC-10:371–379, December 1975.

[MH78] R. H. McCharles and D. A. Hodges. Charge circuits for analog LSI. *IEEE Transactions on Circuits and Systems*, CAS-25:490–497, 1978.

[MM86] Carver Mead and Mary Ann Maher. A charge-controlled model for sub-micron MOS. In *Proceedings of the Colorado Microelectronics Conference*, May 1986.

[MP43] W. C. McCulloch and W. Pitts. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MW85] Carver Mead and John Wawrzynek. A new discipline for CMOS design: an architecture for sound synthesis. In *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 87–104, 1985.

[MWEY81] Norio Miyahara, Takashi Watanabe, Makoto Endo, and Shin-ichiro Yamada. A new CAD system for automatic logic interconnection verification. In *Proceedings of 1981 ISCAS*, pages 114–117, 1981.

[Nag75] Laurence W. Nagel. SPICE2: A computer program to simulate semiconductor circuits. Technical Report ERL-M520, University of California, Berkeley, May 1975.

[Oea85] J. K. Ousethout and et al. The Magic VLSI layout system. *IEEE Design and Test of Computers*, 2(1):19–30, 1985.

[Par73] B. Parhami. Associative memories and processors: An overview and selected bibliography. *Proc. IEEE*, 61:722–723, 1973.

[Riv82] Ronald L. Rivest. The "PI" (placement and interconnect) system. In *19th Design Automation Conference*, pages 475–481, 1982.

[Row80] J.A. Rowson. *Understanding Hierarchical Design*. PhD thesis, California Institute of Technology, 1980.

[Rub87] Steven M. Rubin. *Computer Aids for VLSI Design*. Addison-Wesley, 1987.

[SEM85] Massimo A. Sivilotti, Michael R. Emerling, and Carver A. Mead. A novel associative memory implemented using collective computation. In Henry

Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 329–342, Rockville, MD, 1985. Computer Science Press.

[SEM86] Massimo A. Sivilotti, Michael R. Emerling, and Carver A. Mead. VLSI architectures for implementation of neural networks. In J. Denker, editor, *Neural Networks for Computing (Snowbird 1986)*. American Institute of Physics, 1986.

[SGH75] R. E. Suárez, P. R. Gray, and D. A. Hodges. All-MOS charge redistribution analog-to-digital conversion techniques–part II. *IEEE Journal of Solid State Circuits*, SC-10:379–385, December 1975.

[Siv86a] Massimo A. Sivilotti. Computation and collective systems. *Canadian Research*, 19(7):76–80, 1986.

[Siv86b] Massimo A. Sivilotti. A user's guide to the WOL design tools. Technical Report 5237:TR:86, California Institute of Technology, 1986.

[Siv88] Massimo A. Sivilotti. A dynamically configurable architecture for prototyping analog circuits. In Allen and Leighton, editors, *Advanced Research in VLSI, Proceedings of the Fifth MIT Conference*, pages 237–258, 1988.

[SMM87] Massimo A. Sivilotti, Michelle A. Mahowald, and Carver A. Mead. Real-time visual computation using analog CMOS processing arrays. In *1987 Stanford Conference on Very Large Scale Integration*, Cambridge, MA, 1987. MIT Press.

[Spi83] Rick L. Spickelmier. Verification of circuit interconnectivity. Technical report, University of California, Berkeley, 1983. UCB/ERL M83/66.

[Tan86] John Edward Tanner. *Integrated Optical Motion Detection.* PhD thesis, California Institute of Technology, 1986. 5223:TR:86.

[Tho80] C. D. Thompson. *A Complexity Theory for VLSI*. PhD thesis, Carnegie-Mellon University, 1980. Technical Report CMU-CS-80-140.

[TM86] John E. Tanner and Carver Mead. An integrated analog optical motion sensor. In S.-Y. Kung *et al*, editor, *VLSI Signal Processing, II*, pages 59–76, New York, 1986. IEEE Press.

[TWS90] M. S. Tomlinson, D. J. Walker, and M. A. Sivilotti. A digital neural network architecture for VLSI. In *Proceedings of IJCNN-90*, 1990.

[UD88] C. B. Umminger and S. P. DeWeerth. Implementing gradient following in analog VLSI. In *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI, March 1989*, pages 195–208, Cambridge, MA, 1988. MIT Press.

[Ull84] J. D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, 1984.

[Vit85] Eric A. Vittoz. Micropower techniques. In Yannis Tsividis and Paolo Antognetti, editors, *Design of MOS VLSI Circuits for Telecommunications*, pages 104–144. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[Vit89] Eric Vittoz. CMOS VLSI design: Analog & digital. Intensive Summer Course, Course notes, EPFL, Lausanne, Switzerland, 1989.

[Waw86] John Wawrzynek. A reconfigurable concurrent VLSI architecture for sound synthesis. In S.-Y. Kung *et al*, editor, *VLSI Signal Processing, II*, pages 385–396, New York, 1986. IEEE Press.

[Waw87] J. R. Wawrzynek. *A VLSI Architecture for Sound Synthesis*. PhD thesis, California Institute of Technology, 1987.

[Wee82] C. Weems. TITANIC: A VLSI based content addressable parallel array processor. *Proc. IEEE ICCC*, pages 236–239, 1982.

[Whi85] Telle E. Whitney. *Hierarchical Composition of VLSI Circuits*. PhD thesis, California Institute of Technology, 1985. 5189:TR:85.

[WV89] G. Wegmann and E. Vittoz. Very accurate dynamic current mirror. *IEEE Electron Letters*, 25:644–646, 1989.