THRESHOLD LOGIC

Threshold gates are based on the so-called majority or threshold decision principle, which means that the output value depends on whether the arithmetic sum of values of its inputs exceeds a *threshold.* The threshold principle is general itself and conventional simple logic gates, such as AND and OR gates, are special cases of threshold gates. Thus, threshold logic can treat conventional gates as well as threshold gates in general, in a unified manner.

For many years logic circuit design based on threshold gates has been considered an alternative to the traditional logic gate design procedure. The power of the threshold-gate design style lies in the intrinsic complex functions implemented by such gates, which allow system realizations that require fewer threshold gates or gate levels than a design with standard logic gates. More recently, there has been increasing interest in threshold logic because a number of theoretical results show that polynomial-size bounded level networks of threshold gates can implement functions that require unbounded level networks of standard logic gates. In particular, important functions such as multiple addition, multiplication, division, or sorting can be implemented by polynomial-size threshold circuits of small constant depth. Threshold-gate networks have been found to be also useful in modeling nerve networks and brain organization, and with variable threshold (or weights) values they have been used to model learning systems, adaptive systems, self-repairing systems, pattern-recognition systems, etc. Also, the study of algorithms for the synthesis of threshold-gate networks is important in areas such as artificial neural networks and machine learning.

The article has three well differentiated sections. First, a basic section deals with definitions, basic properties, identification, and complementary metal-oxide-semiconductor (CMOS) implementation of threshold gates. The second section is dedicated to the synthesis of threshold gate networks, from those for specific (and very well-studied) functions such as symmetric or arithmetic functions to general procedures for generic functions. Finally, the third section describes the specific application of threshold logic for the analysis and implementation of median and stack filters.

THRESHOLD AND MAJORITY GATES

A *threshold gate* (TG) is defined as a logic gate with n input variables, x_i ($i = 1, \ldots, n$), which can take values 0,1 and for which there is a set of $n + 1$ real numbers $w_1, w_2, \ldots, w_n$ and T, called weights and threshold, respectively, such that the output of the gate is

$$f = \begin{cases} 1 & \text{for} \quad \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{for} \quad \sum_{i=1}^{n} w_i x_i < T \end{cases} \tag{1}$$

A function represented by the output of a threshold gate, denoted by $f(x_1, x_2, \ldots, x_n)$, is called a threshold function. The set of weights and threshold can be denoted in a more compact vector notation by $[w_1, w_2, \ldots, w_n; T]$.

A *majority gate* is defined as a logic gate with n input variables, ξ_i ($i = 1, \ldots, n$), and a constant input ξ_0, which can take values $+1$, -1 and for which there is a set of real numbers $w_0, w_1, w_2, \ldots, w_n$, called weights, such that the output of the gate is:

$$f = \begin{cases} +1 & \text{for} \quad \sum_{i=0}^{n} w_i \xi_i \geq 0 \\ -1 & \text{for} \quad \sum_{i=0}^{n} w_i \xi_i < 0 \end{cases} \tag{2}$$

A function represented by the output of a majority gate, denoted by $f(\xi_1, \xi_2, \ldots, \xi_n)$, is called a majority function. Analogously to the threshold function, a majority function can be denoted in a vector notation as $[w_1, w_2, \ldots, w_n; w_0\xi_0]$.

Definitions for threshold and majority gates can be seen as different. However, if $+1$ and -1 in a majority gate are correlated to 1 and 0 in a threshold gate, respectively, then a majority gate with a structure $[w_1, w_2, \ldots, w_n; w_0\xi_0]$ and a threshold gate $[w_1, w_2, \ldots, w_n; T]$ have identical logical operations provided that the relation $\xi_i = 2x_i - 1$ is employed and that T is given by

$$T = \frac{1}{2}\left(\sum_{i=1}^{n} w_i - w_0\xi_0\right)$$

As a corollary to this statement, it can be easily shown that the class of all threshold functions is equivalent to the class of all majority functions. Henceforth, we will not differentiate between a majority function and a threshold function as far as logical operations are concerned.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

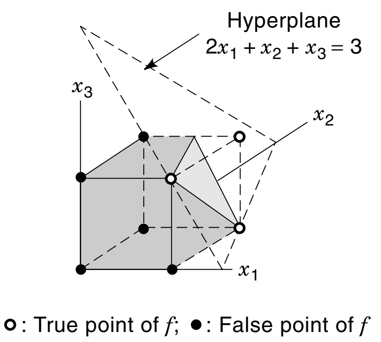


Figure 1. Separation of points of f by a hyperplane.

For example, the TG defined as [2, 1, 1; 3] represents the logical function $f(x_1, x_2, x_3) = x_1x_2 + x_1x_3$. This switching function can be equally represented as the majority gate [2, 1, 1; -2] provided that binary values $+1$ and -1 are correlated to 1 and 0, respectively.

However, it is important to note that nowadays the term *majority gate* is specifically employed for a subset of the gates defined by (1) or (2). They are $2n + 1$ input gates that generate a binary 1 when more than n inputs are at binary 1. Because the definition of a threshold function is in terms of linear inequalities, threshold functions are often called linearly separable functions. From a geometrical point of view, a TG with n inputs can be seen as a hyperplane cutting the Boolean n-cube. It evaluates a function f in the sense that $f^{-1}(1)$ lies on one side of the plane and $f^{-1}(0)$ on the other. An example is shown in Figure 1.

Figure 2(a) shows the IEEE standard symbol for a TG with all input weights equal to 1 and threshold in T. This standard does not have any symbol for a TG with weights other than 1. It is clear that a symbol for that gate can be built by tying together several inputs (a weight of w_i for the input x_i can be obtained by connecting x_i to w_i gate inputs) but it can result in a cumbersome symbol, so we will use the nonstandard symbol of Fig. 2(b) for TGs with generic weights.

Basic Useful Properties of Threshold Functions

There are a number of properties of threshold functions that are useful from the point of view of the viability and efficiency of implementing system using TGs as building blocks. These properties are not proven here but interested readers can find a complete treatment in Refs. 1–3. The emphasis here is put on their usefulness described previously. Some previous definitions are required.

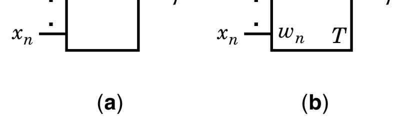


Figure 2. Threshold gate symbols: (a) all weights equal to 1; (b) weights not equal to 1.

A function $f(x_1, x_2, \ldots, x_n)$ is *positive* in x_i if and only if there is a sum-of-product expression for f in which $\bar{x}_i$ does not appear. It can be shown that if f is positive in x_i, then whatever the $\bar{x}_i$ residue of f, $f_{\bar{x}_i}(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$, is 1, the x_i residue of f, $f_{x_i}(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n)$, is also 1. This is, $f_{\bar{x}_i}(x_1, x_2, \ldots, x_n)$ implies $f_{x_i}(x_1, x_2, \ldots, x_n)$, or $f_{\bar{x}_i} \rightarrow f_{x_i}$, where $\rightarrow$ is the symbol for logical implication.

A function $f(x_1, x_2, \ldots, x_n)$ is *negative* in x_i if and only if there is a sum-of-product expression for f in which x_i does not appear. It can also be shown that f is negative in x_i if and only if $f_{x_i} \rightarrow f_{\bar{x}_i}$.

A function is *unate* if and only if it is negative or positive in each of its variables. Now let us enunciate some properties of threshold functions:

Property 1. All threshold functions are unate. There are many unate functions that are not threshold functions.

Property 2. The weights associated with variables in which the function is positive (negative) are positive (negative).

Property 3. Any threshold function can be realized with integer weight and threshold values.

Property 4. Any threshold function can be realized with positive weight and threshold values if inversion is available.

The first two properties are important for the implementation of a procedure for identifying threshold functions, an essential task when a threshold-gate design style is adopted. Determining whether a function is unate or not is simpler than determining whether it can be realized by a TG. So first the function is checked for unateness. If it is not a unate function then it is not a threshold function either. Moreover, during the checking for unateness, variables are classified into positive or negative variables, which also contributes to the simplification of the identification procedure applying the second property.

The third and fourth properties are interesting from the point of view of the physical implementation of the TGs. For example, some of the currently available realizations that will be described later realize positive weights and thresholds. Property 4 guarantees that this does not limit the class of threshold functions they can implement.

Figure 3 shows the elementary relations of threshold functions. The meaning of the arrow labeled with 1 is that if $f(x_1, x_2, \ldots, x_n)$ is a threshold function defined by $[w_1, w_2, \ldots, w_n; T]$, then its complement $\bar{f}(x_1, x_2, \ldots, x_n)$ is also a threshold function defined by $[-w_1, -w_2, \ldots, -w_n; 1 - T]$. If a function can be realized as a threshold element, then by selectively complementing the inputs it is possible to obtain a realization by an element with only positive weights.

$$\begin{tikzcd} \begin{matrix} f(x) \\ [\boldsymbol{w};T] \end{matrix} \arrow[r, "\bar{f}", "(1)"'] \arrow[d, "\bar{x}"] & \begin{matrix} \bar{f}(x) \\ [-\boldsymbol{w};1-T] \end{matrix} \arrow[d, "\bar{f}(\bar{x})"] \\ \begin{matrix} [-\boldsymbol{w};T-\Sigma w_i] \\ f(\bar{x}) \end{matrix} \arrow[r, "\bar{f}"] & \begin{matrix} [\boldsymbol{w};1-T+\Sigma w_i] \\ f^d(x) \end{matrix} \end{tikzcd}$$

Figure 3. Elementary properties of threshold functions.

x_1	x_2	x_3	$h(x_1, x_2, x_3)$	
0	0	0	1	$0 \geq T$
0	0	1	0	$w_3 < T$
0	1	0	0	$w_2 < T$
0	1	1	1	$w_2 + w_3 \geq T$
1	0	0	0	$w_1 < T$
1	0	1	1	$w_1 + w_3 \geq T$
1	1	0	1	$w_1 + w_2 \geq T$
1	1	1	0	$w_1 + w_2 + w_3 < T$

No solution
h is not a threshold function

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	
0	0	0	1	$0 \geq T$
0	0	1	0	$w_3 < T$
0	1	0	1	$w_2 \geq T$
0	1	1	0	$w_2 + w_3 < T$
1	0	0	1	$w_1 \geq T$
1	0	1	1	$w_1 + w_3 \geq T$
1	1	0	1	$w_1 + w_2 \geq T$
1	1	1	1	$w_1 + w_2 + w_3 \geq T$

$w_1 = 2,\ w_2 = 1,\ w_3 = -2,\ T = 0$
f is a threshold function represented by the vector $[2, 1, -2; 0]$

Figure 4. Examples of a straightforward procedure for threshold function identification.

Threshold-Function Identification

A straightforward approach for solving the threshold-function identification problem consists in writing a set of inequalities from the truth table and solving it. If any solution exists, the function is a threshold function with weights and threshold given by the solution. If there is no solution, the function is not a threshold function. In Fig. 4 a pair of examples of this procedure is shown. This procedure is not very efficient because 2^n inequalities are required for a function with n variables. The problem can be solved in a more practical manner using some of the properties listed before. In order to describe this alternative procedure some definitions are needed.

There are 2^n *assignments* of values to n Boolean variables. An assignment $A = (a_1, a_2, \ldots, a_n)$ is smaller than or equal to an assignment $B = (b_1, b_2, \ldots, b_n)$, denoted as $A \leq B$, if and only if $a_i \leq b_i$ $(i = 1, 2, \ldots, n)$. Given a set of assignments $\{A_1, A_2, \ldots, A_k\}$, those A_i for which there is no A_j such that $A_j \geq A_i$, $1 \leq j \leq k$, $j \neq i$, are the *maximal assignments.* The *minimal assignments* are those A_i for which there is no A_j such that $A_j \leq A_i$, $1 \leq j \leq k$, $j \neq i$. Given a function depending on n variables, each assignment for which the function evaluated to 1 is called a *true assignment* and each one for which the function is 0 is called a *false assignment.*

The procedure has the following steps:

1. Determine whether the function f is unate. If not, the function is not a threshold function and the procedure finishes.
2. Convert the function f into another one g, positive in all its variables by complementing every variable for which f is negative.

x_1	x_2	x_3	$h(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(1) Unateness checking not passed

x_2	x_3	$h_{\bar{x}_1}(x_2, x_3)$
0	0	1
0	1	0
1	0	0
1	1	1

x_2	x_3	$h_{x_1}(x_2, x_3)$
0	0	0
0	1	1
1	0	1
1	1	0

As neither $h_{\bar{x}_i} \rightarrow h_{x_i}$ nor $h_{x_i} \rightarrow h_{\bar{x}_i}$ are verified, then h is not a threshold function

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(1) Unateness checking: f positive in x_1 and x_2; f negative in x_3
(2) Function g is positive in all variables

	x_1	x_2	x_3	$g(x_1, x_2, x_3) = f(x_1, x_2, \bar{x}_3)$
A_0	0	0	0	0
A_1	0	0	1	1
A_2	0	1	0	0
A_3	0	1	1	1
A_4	1	0	0	1
A_5	1	0	1	1
A_6	1	1	0	1
A_7	1	1	1	1

(3) Minimal true assignments $\{A_1, A_4\}$; maximal false assignments $\{A_2\}$
(4) Reduced set of inequalities: $w_3 \geq T$, $w_2 < T$, and $w_1 \geq T$, solution for $g(x_1, x_2, x_3)$: [2, 1, 2; 2]
(5) Solution for $f(x_1, x_2, x_3)$: [2, 1, −2; 0]

Figure 5. Examples of the second procedure for threshold function identification.

3. Find minimal true assignments and maximal false assignments for g.
4. Generate inequalities for the assignments obtained in step 3. If there is no solution to such a set of inequalities, the function g is not a threshold function and the procedure finishes.
5. Derive weights and threshold vector for original function f applying the properties just stated. For every variable x_i that is complemented in the original function, its associated weight is changed to $-w_i$, and T to $T - w_i$.

Figure 5 illustrates the procedure for functions h and f from Fig. 4.

CMOS Threshold-Gate Implementations

The effectiveness of threshold logic as an alternative for modern very-large-scale integrated circuit (VLSI) design is determined by the availability, cost, and capabilities of the basic building blocks. In this sense, several interesting circuit concepts have been explored recently for developing standard CMOS-compatible threshold gates. The most promising are presented in this section. In order to denote their context of application, we distinguish between static and dynamic realizations.

Static Threshold-Logic Gates. There are two notable contributions to static threshold-gate CMOS implementations: one is based on the ganged technique (4–7), and the other uses the neuron MOS (νMOS) transistor principle (8–11). In Refs. 5 to 7 the ganged technique proposed in Ref. 4 was employed to build TGs with positive and integer weight and threshold values. Figure 6(a) shows the circuit structure for these ganged-based TGs. Each input x_i drives a ratioed CMOS inverter; all inverter outputs are hard-wired, producing a nonlinear voltage divider that drives a restoring inverter or chain of inverters whose purpose is to quantize the nonbinary sig-

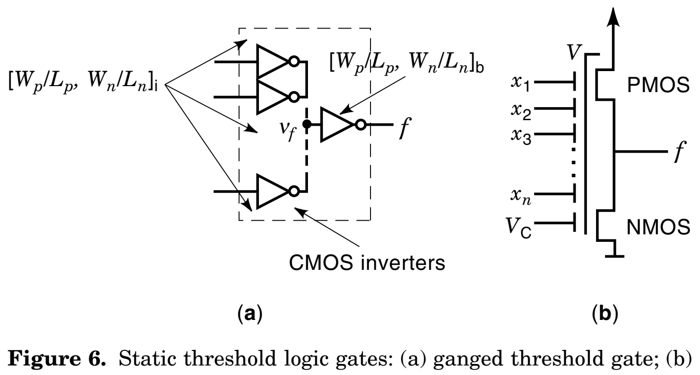


Figure 6. Static threshold logic gates: (a) ganged threshold gate; (b) νMOS threshold gate.

nal at the *ganged* node (ν_f). The design process for these gates involves sizing only two different inverters. Assuming the same length for all transistors, the transistor widths $[W_p, W_n]_{i,b}$ of each inverter are chosen taking into account the w_i and T values to be implemented. Weight values other than 1 can be realized by simply connecting in parallel the number of basic inverters (inverter with $w_i = 1$) indicated by the weight value; on the other hand, the value of T is determined by the value of the output inverter threshold voltage. Due to the sensitivity of this voltage and ν_f to process variations, the ganged-based TG has a limited number of inputs (fan-in). A good study of this limitation can be found in Ref. 12. However, the main drawback of this TG is the relative high power consumption.

Other interesting static TGs are based on the νMOS transistor. This transistor has a buried floating polysilicon gate and a number of input polysilicon gates that couple capacitively to the floating gate. The voltage of the floating gate becomes a weighted sum of the voltages in the input gates, and hence it is this sum that controls the current in the transistor channel. The simplest νMOS-based threshold gate is the complementary inverter using both p- and n-type νMOS devices. A schematic of this TG is shown in Fig. 6(b). There is a floating gate, which is common to both the p- and n-type (PMOS and NMOS) transistors, and a number of input gates corresponding to the threshold gate inputs, $x_1, x_2, \ldots, x_n$, plus some extra inputs (indicated by V_C in the figure) for threshold adjustment. Weights for every input are proportional to the ratio between the corresponding input capacitance C_i between the floating gate and each of the input gates, and the total capacitance, including the transistor channel capacitance between the floating gate and the substrate, C_{chan}. Without using the extra control inputs, the voltage in the floating gate is given by

$$V_F = \left(\sum_{i=1}^{n} C_i V_{x_i} \right) \Big/ C_{tot}$$

where

$$C_{tot} = C_{chan} + \sum_{i=1}^{n} C_i$$

As V_F becomes higher than the inverter threshold voltage, the output switches to logic 0. It is obvious that this νMOS threshold gate is simpler than the ganged threshold gate, however, its sensitivity to parasitic charges in the floating gate and to process variations could limit its effective fan-in unless adequate control is provided (15). In particular, ultraviolet light (UV) erasure is recommended for initialization.

Dynamic Threshold-Logic Gates. Two different principles have been exploited in dynamic TG implementations: the bistable operation of simple CMOS latches, and the capacitive synapse used in artificial neuron architectures. In both cases, compact gates with low power consumption, high speed, and high fan-in have been developed (13–15).

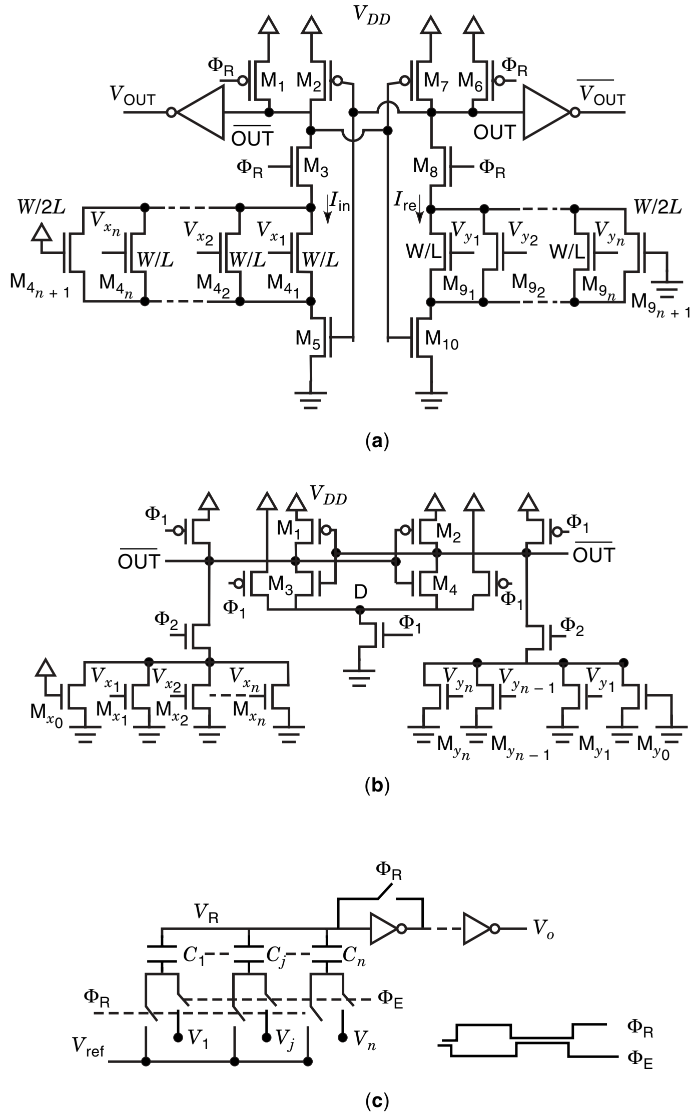


Figure 7. Dynamic threshold gates: (a) latch-type threshold gate; (b) alternative latch-type threshold gate; (c) capacitive-type threshold gate.

The first latch-type threshold gate was proposed in Ref. 13 and its schematic is shown in Fig. 7(a). Its main part consists in a CMOS current-controlled latch (transistor pairs M_2/M_5 and M_7/M_{10}) providing the gate's output and its complement, and two input arrays (M_{4_1} to M_{4_n} and M_{9_1} to M_{9_n}) constituted by an equal number of parallel transistors whose gates are inputs of the TG and their sizes are determined by the corresponding weight and threshold values. Transistor pairs M_1/M_3 and M_6/M_8 specify the precharge or evaluation situation, and the two extra transistors $M_{4_{n+1}}$ and $M_{9_{n+1}}$ ensure correct operation when the weighted sum of inputs is equal to the threshold value. Precharging occurs when the reset signal Φ_R is at logic 0. Transistors M_1 and M_6 are on, transistors M_3 and M_8 are off, and both OUT and $\overline{OUT}$ are at logic 1. Evaluation begins when Φ_R is at a logic 1, transistors M_1 and M_6 are turned off, M_3 and M_8 are turned on, and nodes OUT and $\overline{OUT}$ begin to be discharged. In this situation, depending on the logic values at the inputs of the two transistor arrays, one of the paths will sink more current than the other, making the decrease of its corresponding output node voltage faster (OUT or $\overline{\text{OUT}}$). When the output node of the path with the highest current value is below the threshold voltage of transistors M_5 or M_{10}, one of them is turned off, fixing the latch situation completely. Supply current only flows during transitions and, consequently this TG does not consume static power.

Input terminal connections and input transistor sizes in this TG must be established according to the threshold value T to be implemented, and to the fact that when all transistors M_{4_i} and M_{9_i} ($i = 1, 2, \ldots, n$) have the same dimension and the same voltage at their gate terminal, then $I_{in} > I_{ref}$ due to $M_{4_{n+1}}$. If a programmable TG is required, the best design choice is to use one of the input arrays for the TG inputs and the other array for control inputs, which must be put to logic 1 or 0 depending on the value of T. For illustration, the operation of a 20-input threshold gate [1, 1, . . ., 1; T] with programmable threshold T is shown in Fig. 8. The outputs depicted correspond to different values of T: (a) $T = 1$, this is a 20-input OR-gate; (b) $T = 10$; and (c) $T = 20$, a 20-input AND-gate. The results shown correspond to the following sequence of logic input patterns: $(x_1, x_2, \ldots, x_{19}, x_{20}) = \{(0, 0, \ldots, 0, 0), (0, 0, \ldots, 0, 1), (0, 0, \ldots, 1, 1), \ldots, (1, 1, \ldots, 1, 1)\}$. The i-th input combination is evaluated in the i-th reset pulse. So, we have the weighted sum of the inputs in the x scales.

The circuit in Fig. 7(b) is an alternative realization proposed in Ref. 14 for dynamic latch-type threshold gates. In this gate, the input transistor arrays (M_{x_i} and M_{y_i}, $i = 0, 1, \ldots, n$) are connected directly to the latch's output nodes, and precharging occurs when Φ_1 and Φ_2 are at logic 0, putting nodes D, OUT, and $\overline{\text{OUT}}$ at logic 1. For the evaluation phase both Φ_1 and Φ_2 are at logic 1 but Φ_2 must return to the low level before Φ_1 in order to allow latch switching. The performance of this TG is similar to that in Ref. 13 but it needs more transistors and two different control signals that have to be obtained from a general clock.

The principle of capacitive synapse has been exploited in the capacitive threshold-logic gate proposed in Ref. 15. Its conceptual circuit schematic is shown in Fig. 7(c) for an n-input gate. It consists of a row of capacitors C_i, $i = 1, 2, \ldots, n$, with capacitances proportional to the corresponding input weight, $C_i = w_i C_u$, and a chain of inverters that functions as a comparator to generate the output. This TG operates with two nonoverlapping clock phases Φ_R and Φ_E. During the reset phase, Φ_R is high and the row voltage V_R is reset to the first inverter threshold voltage while the capacitor bottom plates are precharged to a reference voltage V_{ref}. Evaluation begins when Φ_E is at a logic 1, setting gate inputs to the capacitor bottom plates. As a result, the change of voltage in the capacitor top plates is given by

$$\Delta V_R = \left(\sum_{i=1}^{n} C_i (V_i - V_{ref}) \right) \Big/ C_{tot}$$

where C_{tot} is the row total capacitance including parasitics. Choosing adequate definitions for V_{ref} and C_i as functions of the input weight and threshold values, this above relationship can be expressed as

$$\Delta V_R = \left(\sum_{i=1}^{n} (w_i x_i - T) C_u V_{DD} \right) \Big/ C_{tot}$$

which together with the comparison function of the chain of inverters give the TG operation:

$$V_o = V_{DD} \qquad \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq T$$

and

$$V_o = 0 \qquad \text{if} \quad \sum_{i=1}^{n} w_i x_i < T$$

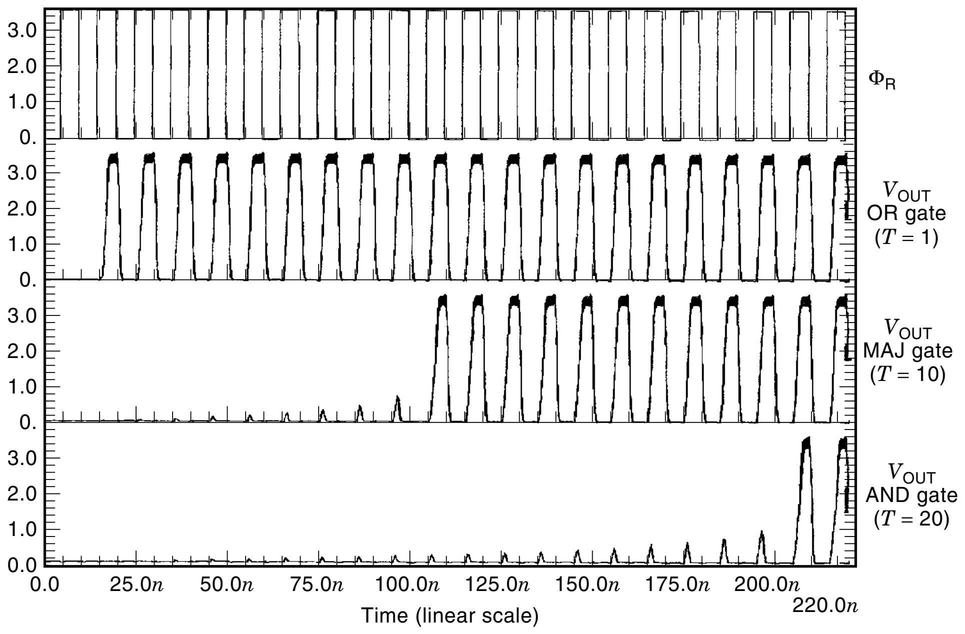


Figure 8. Simulation results for a programmable threshold gate implemented by the circuit of Fig. 7(a). The letter n stands for nanoseconds.

Experimental results from different capacitive threshold-logic gates fabricated in a standard CMOS technology (15) have shown the proper functionality of this type of threshold gates and its large fan-in capability.

THRESHOLD-GATE NETWORKS

There are functions that cannot be implemented by a single threshold element. However, as TGs realize more complex functions than the conventional gates, this section studies the capabilities of interconnecting TGs so that the potential advantages of realizing digital systems using TGs as building blocks are pointed out. First, the question of whether any Boolean function can be realized interconnecting TGs is addressed. Then the computation power of such a network is analyzed.

Figure 9 shows the model for a feed-forward network of functional elements. It is the most general type of network without feedback because inputs can be connected to any of the functional blocks in the network, and the only restriction affecting the functional blocks to which the output of one of them can be connected is that loops are not allowed. The *depth* of a network is the maximum number of functional elements in a path from any input to any output. The *size* of a network is defined as the total number of functional elements it contains. In the following a feed-forward network in which the functional elements are conventional digital gates (AND, OR, NOT) will be referenced as a logic circuit or logic network. It is well known that any Boolean function can be implemented by a logic circuit, and so, any Boolean function can also be implemented by a feedforward network of TGs as AND, OR and NOT gates are TGs. However, from a practical point of view, these results are not enough. The existence of a determinate network implementing a given function can be irrelevant. This is illustrated by a network that computes the function too slowly to fulfill speed specifications or a network with too large a hardware cost. That is, the depth and the size of the network are critical because these parameters are related to the speed and to the required amount of hardware, respectively. For example, it is well known that any function can be implemented with a depth-2 logic network of AND and OR gates (depth-3 network of NOT–AND–OR gates if input variables are in single rail). However, it is also well known that there are common functions for which the number of required gates in such implementations increases exponentially with the number of variables, and so they are not realized in two levels. From a different point of view, there are functions that cannot be implemented by a polynomial-sized network with constant depth (independent of the number of variables), resulting in too large a computation time for large n.

It follows from the previous arguments that a more realistic problem is determining the existence of a constant-depth network with size bounded by a polynomial in n, $O(n)$, for a Boolean function of n variables. There are a number of functions for which the answer to this question is negative for logic networks and positive for threshold networks.

Consider, for example, the parity function, $f_{\text{parity}}(x_1, x_2, \ldots, x_n)$, which is 1 if and only if an even number of its inputs are logical 1. No logic circuit with a polynomial (in n) number of unbounded fan-in AND–OR–NOT gates can compute it with constant depth (16). In Fig. 10(a) a depth-2 logic network implementing parity for $n = 4$ is depicted. For arbitrary n, its size is $2^{n-1} + 1$. A depth-2 threshold network for f_{parity} of four variables is shown in Fig. 10(b). For an arbitrary n only $n + 1$ gates are required.

The parity function belongs to the more general class of symmetric functions that can be efficiently implemented by threshold networks and that have received much attention. Symmetric functions are the subject of the next subsection.

Threshold Networks for Symmetric Functions

Symmetric functions are not particularly easy to realize using traditional logic gates. However, an important feature of threshold logic gates is their capability for obtaining implementations of symmetric functions by simple networks.

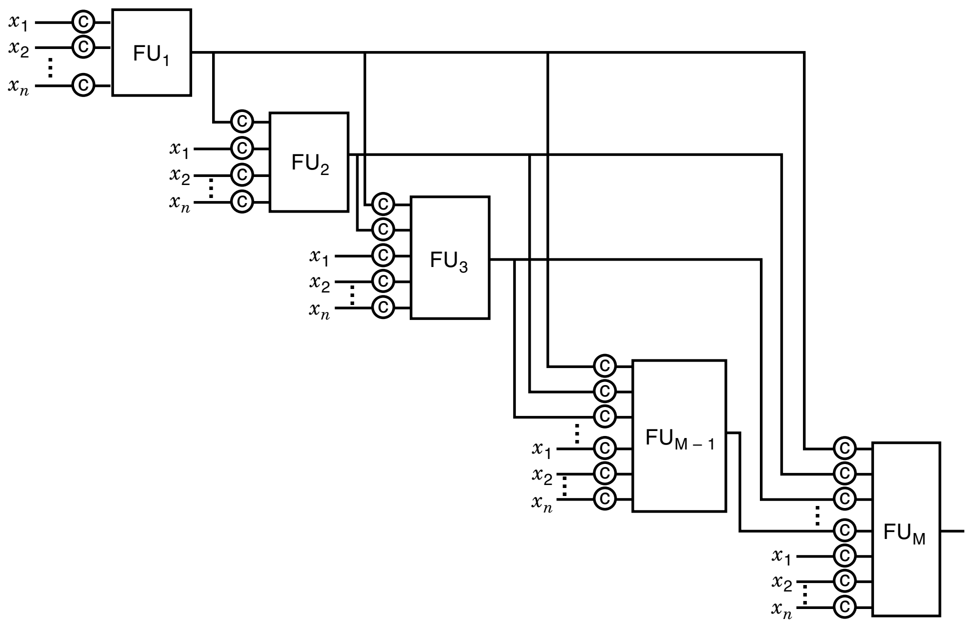


Figure 9. Feed-forward network of functional elements (FU). The letter c denotes where connections can exist.

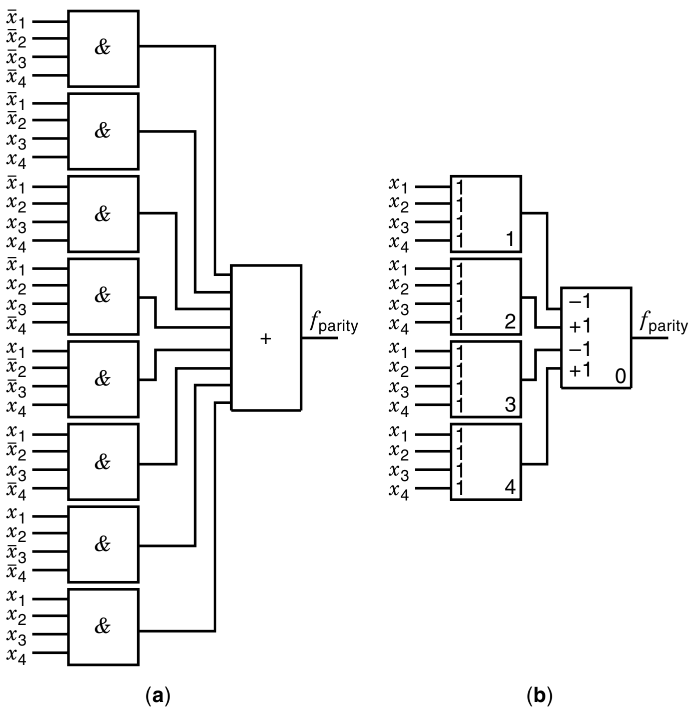
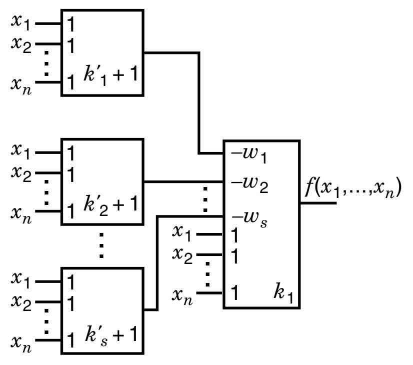


Figure 10. Networks realizing f_{parity} for $n = 4$: (a) logic network; (b) threshold network. Symbol & denotes AND gates and symbol + denotes OR gates.



$$w_j = k_{j+1} - k_j \qquad (j = 1, 2, \ldots, s-1)$$

$$w_s = \begin{cases} n + 1 - k_s & \text{if } k'_s < n \\ 0 & \text{if } k'_s = n \end{cases}$$

Figure 12. Minnick solution to the threshold-gate network implementation of a symmetric function.

Classically, two main solutions have been considered depending on the availability of the input variables of the network. Muroga (1) proposed a solution suitable when the input variables are available only at the first-level gates and the network has feed-next interconnections. Then any symmetric function of n variables can be implemented by a network that has at most $n + 1$ threshold gates in two levels. To show that, let us suppose that the symmetric function is 1 if and only if the number of 1's in the n variables, given by k, is in one of the ranges $k_1 \leq k \leq k'_1, k_2 \leq k \leq k'_2, \ldots, k_s \leq k \leq k'_s$. Figure 11 shows the threshold-gate network used to implement this function. If the number of 1's in the input variables, k, is $k_i \leq k \leq k'_i$, $1 \leq i \leq s$, then all the TGs whose thresholds are equal to or smaller than k have outputs of 1, and the other TGs have outputs of 0. Then, there are $2i - 1$ gates with output 1. Among them, i gates are connected to the output gate with weight $+1$, and $i - 1$ gates with weight -1. Thus the weighted sum of these inputs is $+1$, and the output gate will give an output of 1. When $k'_i + 1 \leq k \leq k_{i+1} - 1$, there are $2i$ gates with output 1, but the weighted sum of these inputs to the second TG is 0 and, consequently the output of that output gate is 0.

Minnick (in Ref. 1) authored a solution for which the input variables are available at gates of any level. A symmetric function of n variables can be implemented by a network that has at most $1 + \lfloor n/2 \rfloor$ TGs in at most two levels. Here $\lfloor x \rfloor$ denotes the integral part of x. To show that, let us suppose that the symmetric function is 1 if and only if the number of 1's in the n variables, given by k, is in one of the ranges $k_1 \leq k \leq k'_1$, $k_2 \leq k \leq k'_2, \ldots k_s \leq k \leq k'_s$. Figure 12 shows the threshold-gate network used to implement the symmetric function. If the number of 1's in the input variables k is $k_i \leq k \leq k'_i$, $1 \leq i \leq s$, then all the TGs with threshold equal to or smaller than k have an output of 1. As $k'_{i-1} + 1 < k_i \leq k \leq k'_i$, then there are $i - 1$ TGs giving an output of 1, which add

$$-\sum_{j=1}^{i-1} w_j = -(k_i - k_1)$$

to the weighted sum of the output threshold gate. The weighted sum of that gate will be $-(k_i - k_1) + k$, which, when compared to k_1, results in a number that is equal to or greater than zero, and in consequence, the output of the output threshold gate is 1. The case for an output of 0 can be shown in a similar manner. Also an equivalent realization can be found by expressing the outputs of the first-level threshold gates as inverted instead of weighted negatively.

An interesting solution for the modulo-2 sum (parity) of n variables was proposed by Kautz in (1). It is realizable with at most $s = 1 + \lfloor \log_2 n \rfloor$ threshold gates. The feed-forward network proposed is shown in Fig. 13(a), and the synthesis problem can be easily extended to solve a general symmetric function. Let us consider the general feed-forward solution shown in Fig. 13(a) specific to $s = 4$. Figure 13(b) shows the output values of the gates of this network in terms of the number of 1's in the input variables. It can be easily seen that the number of 1's in the input variables at which transitions

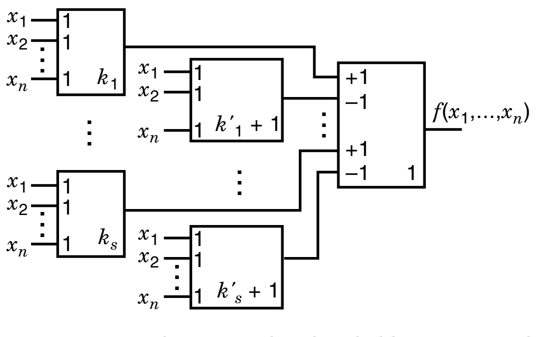
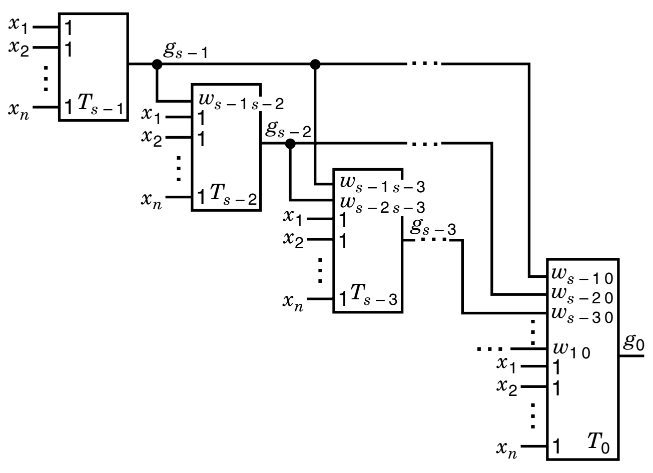
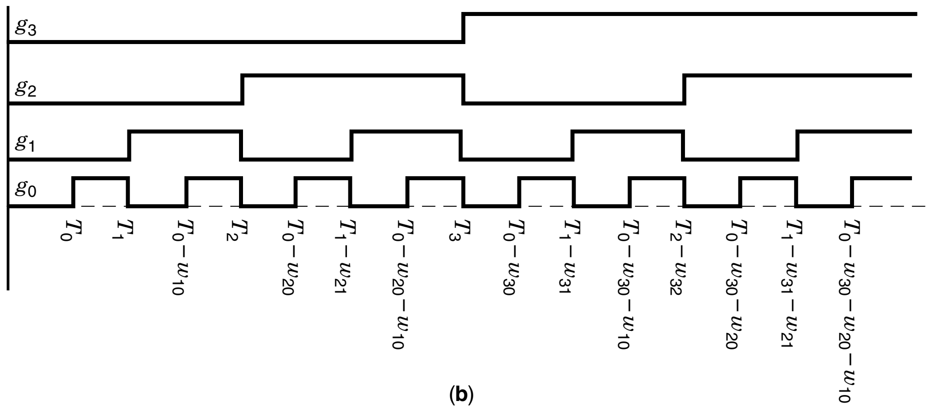


Figure 11. Muroga solution to the threshold-gate network implementation of a symmetric function.



(a)



(b)

Figure 13. Kautz solution to the threshold-gate network implementation of a symmectric function: (a) general structure, (b) transitions of output values for $s = 4$.

from the value 0 to 1 occur in g_k defines the oppositely direct transitions from 1 to 0 in $g_{k-1}, g_{k-2}, \ldots, g_0$. Also, it is important to consider that although T_3, T_2, $T_2 - w_{32}$, T_1, $T_1 - w_{21}$, $T_1 - w_{31}$, T_0, $T_0 - w_{10}$, $T_0 - w_{20}$, $T_0 - w_{30}$ for g_3, g_2, g_1, and g_0 are independent and arbitrarily determined, some of the relations for g_1 and g_0 ($T_1 - w_{31} - w_{21}$, $T_0 - w_{20} - w_{10}$, $T_0 - w_{30} - w_{10}$, $T_0 - w_{30} - w_{20}$, and $T_0 - w_{30} - w_{20} - w_{10}$) are consequently determinated. Thus, the synthesis of a general symmetric function can become very complex because of this mutual dependence of parameters.

Solutions proposed by Muroga and Minnick (1) implement symmetric functions in an $O(n)$ depth-2 threshold network. Reducing the size of the network significantly from $O(n)$ requires an increasing of the network depth beyond 2. Recently it has been shown (17) that any symmetric function of n variables can be implemented with a depth-3 threshold network with at most $2\sqrt{n} + O(1)$ threshold gates, that is, an increase of 1 in the depth allows an implementation with a gate count reduced by a factor of $O(\sqrt{n})$.

Threshold Networks for Arithmetic Functions

Usually when implementing arithmeticlike functions by threshold networks, the required weights can grow exponentially fast with the number of variables. This is undesirable because of the requirements of high accuracy it places on the actual implementations. An example is the comparison function $f_{\mathrm{comp}}(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$, which takes as input two n-bit binary numbers $x = x_n x_{n-1} \cdots x_1$ and $y = y_n y_{n-1} \cdots y_1$ and produces output equal to 1 if and only if $x \geq y$. Figure 14(a) shows a TG implementing the function. The depth-2 network in Fig. 14(b) would be preferable for moderately large values of n because of the smaller weights it requires. A subclass of the threshold gates, those for which the weights are polynomially bounded by the number of input variables, is more practical. Restricting the allowed weights does not limit too much the computational power of the network. It has been shown (18,19) that any depth-d polynomial-size threshold circuit can be implemented by a depth-$(d + 1)$ polynomial-size network of the restricted threshold elements.

Interesting results have been derived for functions such as multiple addition, multiplication, division, or sorting, which have been shown to be computable by small constant-depth polynomial-size threshold networks. Different proposed implementations exhibit different depth-size trade-offs. Some results concerning optimal depth threshold networks are given in Ref. 20. It is demonstrated that multiplication, division, and powering can be computed by depth-3 polynomial-size threshold circuits with polynomially bounded weights.

The efficient threshold networks derived for these functions rely in many cases on the underlying new computation algorithms. One example is the block save addition (BSA) principle for multiple addition. Siu and Bruck (21) showed that the sum of n numbers can be reduced to the sum of two numbers by using the BSA principle. The key point of this technique is the separation of the n numbers in columns of $\lceil \log n \rceil$ bits that are separately added. Each sum is at most $2\lceil \log n \rceil$ bits long, and hence it overlaps only with the sum of the next column. Here $\lceil x \rceil$ denotes the smallest integer equal to or greater than x. So a number is obtained by concatenating the partial sums from the columns placed in even positions and another number with the concatenation of the sum from the odd columns.

In general, the realizations introduced so far cannot be directly applied if the fan-in of the TGs is constrained to be not more than m, $m \ll n$, where n stands for the number of input variables. Thus another area receiving attention is that of deriving depth-size trade-off for threshold networks implementing arithmeticlike functions with a simultaneous bound on the maximum allowable fan-in. Let us resort again to the parity function in order to illustrate this statement. In Ref. 22 it is shown that the parity function of n inputs can be computed using a threshold circuit of size $O(nm^{-1/(2^d-1)})$, depth $O(d \log n/\log m)$, and fan-in bounded by m for every integer $d > 0$.

Threshold Network Synthesis

The significance of the preceding results is that assuming TGs can be built with a cost and delay comparable to that of logic gates, many basic functions can be computed much faster and/or much cheaper using TGs than using logic gates. This is one of the motivations for investigating devices able to implement TGs. However, the usefulness of threshold logic as a design alternative, in general, is determined not only by the availability, cost, and capabilities of the basic building blocks but also by the existence of synthesis procedures. The problem to be solved at this level can be stated as given a combinational logic function, described in the functional domain (by means of truth tables, logic expressions, etc.), derive a network of the available building blocks realizing f that is optimal according to some design criteria.

Many logic synthesis algorithms exist for targeting conventional logic gates but few have been developed for TGs, although the problem was addressed as early as the beginning of the 1970s by Muroga. The procedure described by this author (1) transforms the problem of deriving the smallest (lowest gate count) feed-forward network realizing a given function in a sequence of mixed integer linear programming (MILP) problems. The problem of determining whether a given function f can be realized by a feed-forward threshold network with M gates, and if it can, determining the weights and the threshold for each of the M elements can be formu-

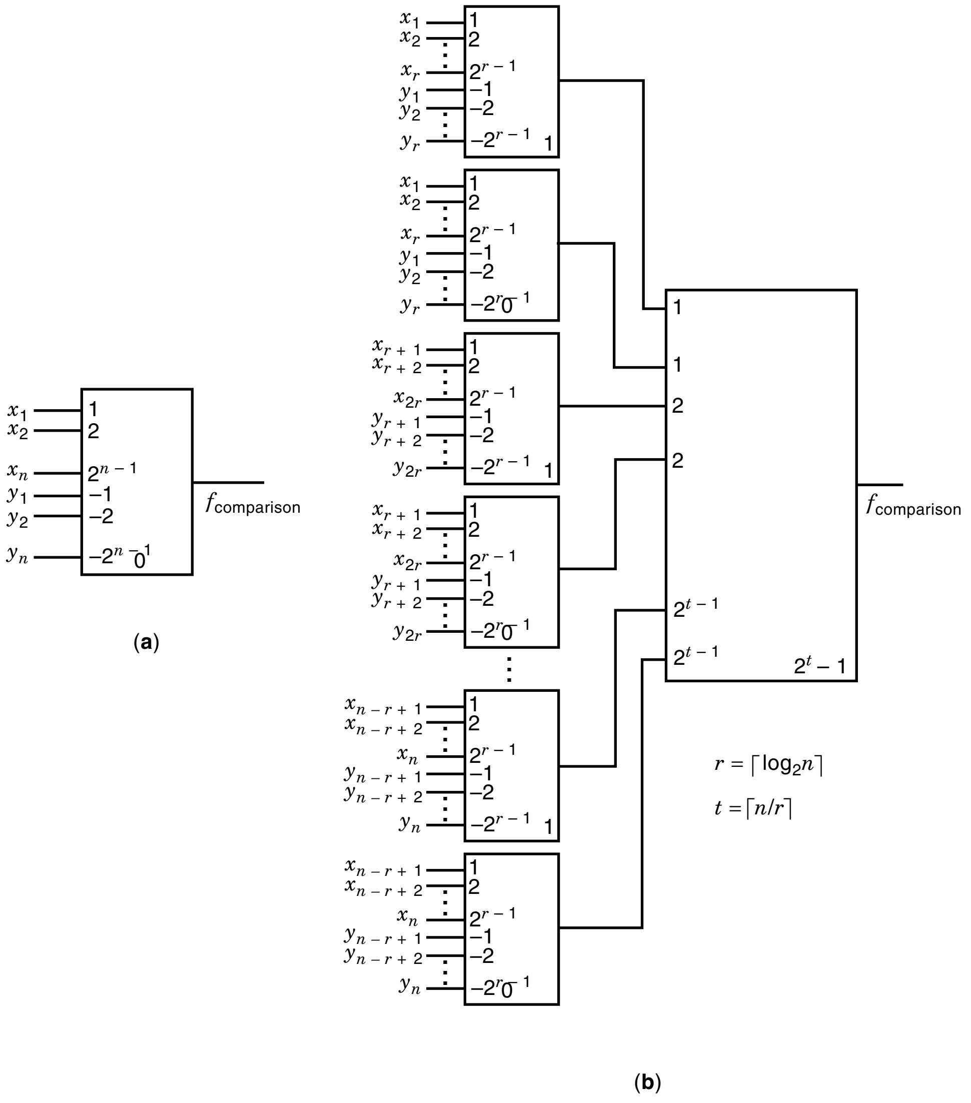


Figure 14. Networks realizing $f_{\text{comparison}}$: (a) depth-1 network; (b) depth-2 network.

lated as a MILP problem (the cost function is usually the total weight sum of the network). Starting with $M = 1$ and incrementing M until a feasible MILP problem is encountered, one derives the implementation with less gates. Clearly, exact approaches are practical only for small instances of the synthesis problem. The main limitation seems to be the number of variables that the function being synthesized depends on, because the number of inequalities and variables in the MILP problem to be solved increase exponentially with n. Thus, heuristic approaches are more relevant.

Concerning two-level (depth-2) threshold networks, an algorithm called LSAT (23), inspired in techniques used in classical two-level minimization of logic circuits, has been developed. The core of the algorithm performs as follows. Suppose we have a two-level threshold network satisfying the following conditions: (1) weights of first-level TGs are restricted to the range $[-z, +z]$ and (2) weights of the second-level gate are all equal to 1 and the threshold of this gate is S. Another threshold network that also satisfies previous conditions (1) and (2) with a minimal number of gates is obtained. This operation is repeated increasing S by 1 until a value of S is reached for which no solution is found. As a two-level AND–OR network is a threshold network of the type handled by the procedure with $z = 1$, $S = 1$, the algorithm is started with this network. Such a two-level circuit is easy to obtain and in fact is a standard input for other synthesis tools. LSAT has a run-time polynomial in the input size given by $n \times z$, where n stands for the number of variables and z defines the allowed range for the weights. This means central processing unit (CPU) time increases if large weights are required.

The practical use of synthesis procedures for TGs is not restricted to the design of integrated circuits but to areas such as artificial neural networks or matching learning. Different problems encountered in these fields are naturally formulated as threshold network synthesis problems

APPLICATION TO MEDIAN AND STACK FILTERS

For some time, linear filters have been widely used for signal processing mainly due to their easy design and good performance. However, linear filters are optimal among the class of all filtering operations only for additive Gaussian noise. Therefore problems such as reduction of high frequency and impulsive noise in digital images, smoothing of noisy pitch contours in speech signal, edge detection, image preprocessing in machine recognition, and other related problems with the suppression of noise that is non-Gaussian, nonadditive, or even not correlated with the signal can be difficult to solve (24).

These unsatisfactory results provided by linear filters in signal and image processing have been overcome by resorting to nonlinear filters. The more well known is perhaps the median filter, which has found widespread acceptance as the preferred technique to solve the signal restoration problem when the noise has an impulsive nature or when the signals have sharp edges that must be preserved. But the median filter has inherent problems because its output depends only on the values of the elements within its window. So, a median filter with a window width of $n = 2L + 1$ can only preserve details lasting more than $L + 1$ points. To preserve smaller details in the signal, a smaller window width must be used. But the smaller this window width, the poorer the filter noise-reduction capability (25).

This contradiction can be solved by incorporating in the filter output the index order of the sequence of elements. It is typically done by weighting filter input values according to their relative sequence index order. This idea leads in a natural way to the concept of the weighted-median (WM) filter (26), which has the same advantages as the median filter but is much more flexible in preserving desired signal structures due to the defining set of weights. Median and weighted-median filters are well-known examples of a larger class of nonlinear filters: the stack filters, which also include the maximum-median filters, the midrange estimators, and several more filters.

Stack filters are a class of sliding finite-width-window, nonlinear digital filters defined by two properties called the *threshold decomposition* (a superposition property) and the *stacking property* (an ordering property) (24). The threshold decomposition of an M-valued signal $\boldsymbol{X} = (X_1, X_2, \ldots, X_N)$, where $X_i \in \{0, 1, 2, \ldots, M - 1\}$, $i = 1, \ldots, N$ is the set of $(M - 1)$ binary signals $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{M-1}$, called threshold signals, defined as:

$$x_i^j = \begin{cases} 1 & \text{if} \quad X_i \geq j \\ 0 & \text{else} \end{cases} \qquad j = 1, \ldots, M-1 \tag{3}$$

From this definition, it is clear that

$$\sum_{j=1}^{M-1} x_i^j = X_i \qquad \forall \quad i \in \{1, \ldots, N\}$$

and also that the x_i^j are ordered, that is, $x_i^1 \geq x_i^2 \geq \cdots \geq x_i^{M-1}$ $\forall$ $i \in \{1, \ldots, N\}$. This ordering property is called the stacking property of sequences.

Two binary signals $\boldsymbol{u}$ and $\boldsymbol{v}$ "stack" if $u_i \geq v_i$, $i = 1, \ldots, N$. Let us suppose that both signals are filtered with a binary window filter of width L [i.e., we use a Boolean function B: $\{0, 1\}^L \rightarrow \{0, 1\}$ for the filtering operation, which results in $B(\boldsymbol{u})$ and $B(\boldsymbol{v})$]. The binary filter B exhibits the stacking property if and only if $B(\boldsymbol{u}) \geq B(\boldsymbol{v})$ whenever $\boldsymbol{u} \geq \boldsymbol{v}$.

The stack filter $S_B(\boldsymbol{X})$ is defined by a binary filter $B(\boldsymbol{x})$ as follows:

$$S_B(\boldsymbol{X}) = \sum_{j=1}^{M-1} B(\boldsymbol{x}^j) \tag{4}$$

The threshold decomposition architecture of stack filters means that filtering an M-valued input signal by the stack filter S_B is equivalent to threshold decomposing the input signal to $M - 1$ binary threshold signals, filtering each binary signal separately with the binary filter B, and finally adding the binary output signal together to reconstruct the M-valued signal. As stack filters possess the stacking property, this reconstruction section needs only to detect the level just before the transition from 1 to 0 takes place.

Figure 15 illustrates the threshold decomposition architecture of a stack filter with a window width of 3 for the four-valued input signal shown at the upper left corner. The binary signals are obtained by thresholding the input signal at levels 1, 2, and 3. Binary filtering is independently performed

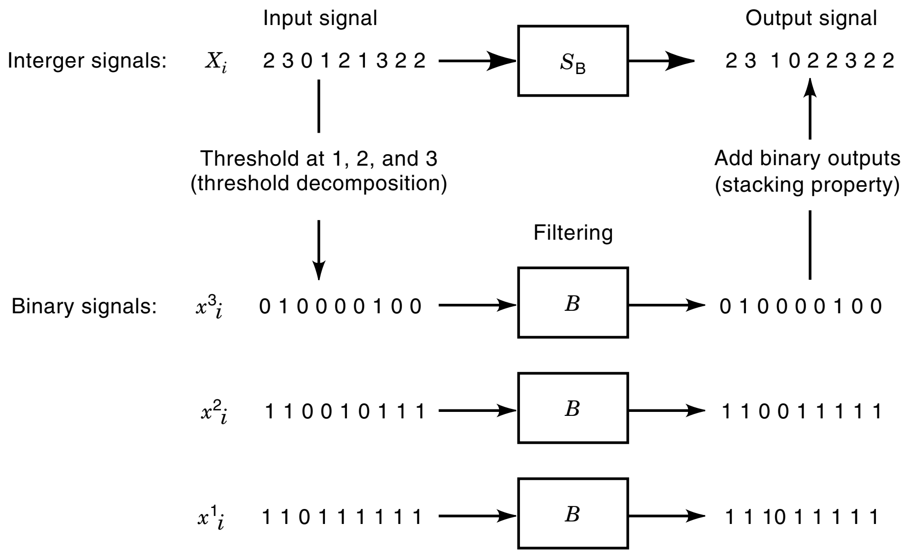


Figure 15. Illustration of threshold decomposition and the stacking property.

by the digital function $B(a, b, c) = ac + b$, in which a, b, and c are the bits, in time order, appearing in the filter's window.

In the original integer domain of the M-valued input signal, the stack filter corresponding to a positive Boolean function (PBF) can be expressed by replacing logical operators AND and OR with MIN and MAX operations, respectively. In consequence, the output of a stack filter is a composition of maximum and minimum operations on the samples in the window. For the example in Fig. 15, this means that the operation performed by S_B is $S_B(A, B, C) = \text{MAX}\{\text{MIN}\{A, C\}, B\}$. Both filtering operations are represented in Fig. 15: by threshold decomposition if the lightface arrows are followed and directly, by the stack filter S_B, following the boldface arrows.

The next question is to know which binary functions possess the stacking property. It has been shown that the necessary and sufficient condition for this is that the binary function is a PBF, that is, positive in all its variables. These functions are a subset of unate functions that have the property that each one possesses a unique minimum sum-of-products (SOP) expression, and hence each stack filter can be described in terms of a unique minimum SOP Boolean expression. Finally, as shown previously, threshold functions are a subset of unate functions. Stack filters that are based on TGs with nonnegative weights and nonnegative threshold values are called weighted-order statistics filters.

It can be very instructive to show the relations of the more usual members of the class of stack filters, namely, weighted-order statistic (WOS), weighted-median (WM), order-statistic (OS), and standard-median (SM) filters. In Fig. 16 these relations are shown by means of boxes and arrows. Each box corresponds to a filter subclass specified by the integer domain filter and the binary domain filter. The arrows indicate the containing conditions among classes of filters.

From a practical point of view, there are several options for the very-large-scale integrated circuit (VLSI) implementation of the PBFs of a stack filter: binary logic gates, sort-and-select circuits, or count-and-compare circuits. If logic gates or a programmable logic array (PLA) is used, a number of terms bounded by

$$\binom{n}{\lfloor n/2 \rfloor} \quad \text{or} \quad \binom{n}{\lceil n/2 \rceil}$$

for a window width of n can be obtained. The hardware complexity for sort-and-select circuits is $O(n \log n)$ and $O(n)$ for count-and-compare circuits. The PBF can also be realized as a look-up table by a 2^n-sized random-access memory (RAM) or read-only memory (ROM), and if a RAM is used, programmable PBF-based filters can be made.

In the case of a WOS filter, its PBF can be realized by a TG. It constitutes a great advantage because the number of product terms or sum terms of the PBF can be as large as

$$\binom{n}{T}$$

while the representation of a TG needs only $n + 1$ components, the n weights and the threshold T. Therefore, while the implementation of a generic stack filter can be very diffi-

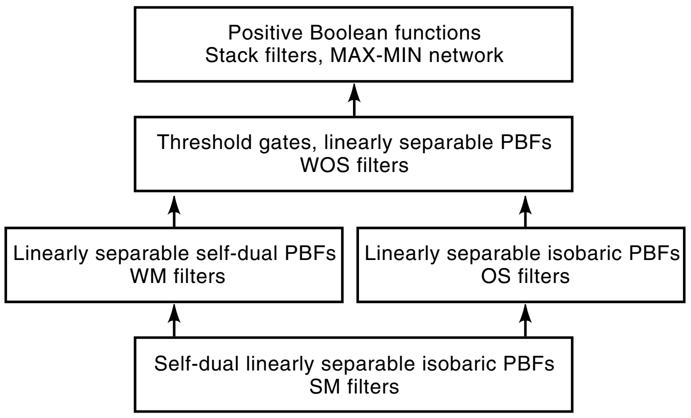

Positive Boolean functions
Stack filters, MAX-MIN network
Threshold gates, linearly separable PBFs
WOS filters
Linearly separable self-dual PBFs
WM filters
Linearly separable isobaric PBFs
OS filters
Self-dual linearly separable isobaric PBFs
SM filters

Figure 16. Relations of linearly separable subclasses of stack filters.

cult, the implementation of the subclass of WOS filters can be affordable if an efficient realization of TGs is used.

BIBLIOGRAPHY

1. S. Muroga, *Threshold Logic and its Applications,* New York: Wiley, 1971.
2. Z. Kohavi, *Switching and Finite Automata Theory,* New Delhi: Tata McGraw-Hill, 1978.
3. E. J. McCluskey, *Logic Design Principles: With Emphasis on Testable Semicustom Circuits,* Englewood Cliffs, NJ: Prentice Hall, 1986.
4. K. J. Schultz, R. J. Francis, and K. C. Smith, Ganged CMOS: Trading standby power for speed, *IEEE J. Solid-State Circuits,* **25** (3): 870–873, 1990.
5. C. L. Lee and C-W. Jen, CMOS threshold gate and networks for order statistic filtering, *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.,* **41** (6): 453–456, 1994.
6. J. M. Quintana, M. J. Avedillo, and A. Rueda, Hazard-free edge-triggered D flipflop based on threshold gates, *Electron Lett.,* **30** (17): 1390–1391, 1994.
7. J. M. Quintana et al., Practical low-cost CMOS realization of complex logic functions, *Proc. Eur. Conf. Circ. Theor. Design,* pp. 51–54, 1995.
8. T. Shibata and T. Ohmi, A functional MOS transistor featuring gate level weighted sum and threshold operations, *IEEE Trans. Electron Devices,* **39** (6): 1444–1445, 1990.
9. T. Shibata and T. Ohmi, Neuron MOS binary-logic integrated circuits—Part I: Design fundamentals and soft-hardware-logic circuit implementations, *IEEE Trans. Electron Devices,* **40** (3): 570–576, 1993.
10. T. Shibata and T. Ohmi, Neuron MOS binary-logic integrated circuits—Part II: Simplifying techniques of circuit configuration and their practical applications, *IEEE Trans. Electron Devices,* **40** (5): 974–979, 1993.
11. W. Weber et al., On the application of the neuron MOS transistor principle for modern VLSI design, *IEEE Trans. Electron Devices,* **43** (10): 1700–1708, 1996.
12. N. Balabbes et al., Ratioed voter circuit for testing and fault-tolerance in VLSI processing arrays, *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.,* **43** (2): 143–152, 1996.
13. M. J. Avedillo et al., Low-power CMOS threshold-logic gates, *Electron. Lett.,* **31** (25): 2157–2159, 1995.
14. J. Fernández Ramos et al., A threshold logic gate based on clocked coupled inverters, *Int. J. Electron.,* **84** (4): 371–382, 1998.
15. H. Özdemir et al., A capacitive threshold-logic gate, *IEEE J. Solid-State Circuits,* **31** (8): 1141–1150, 1996.
16. M. Furst, J. B. Saxe, and M. Sipser, Parity, circuits and the polynomial-time hierarchy, *Proc. IEEE Symp. Found. Comp. Sci.,* 1981, pp. 260–270.
17. K-Y. Siu, V. P. Roychowdhury, and T. Kailath, Depth-size tradeoffs for neural computation, *IEEE Trans. Comput.,* **40** (12): 1402–1412, 1991.
18. M. Goldmann, J. Hastad, and A. Razborov, Majority gates vs. general weighted threshold gates, *Proc. 7th Annu. Conf. Struct. Complexity Theory,* 1992, pp. 2–13.
19. M. Goldmann and M. Karpinski, Simulating threshold circuits by majority circuits, *Proc. 25th Annu. ACM Symp. Theory Comput.,* 1993, pp. 551–560.
20. K. Siu and V. P. Roychowdhury, An optimal-depth threshold circuit for multiplication and related problems, *Soc. Ind. Appl. Math. J. Discrete Math.,* **7** (2): 284–292, 1994.
21. K. Siu and J. Bruck, Neural computation of arithmetic functions, *Proc. IEEE,* **78** (10): 1669–1675, 1990.
22. K. Siu, V. P. Roychowdhury, and T. Kailath, Toward massively parallel design of multipliers, *J. Parallel Distr. Comput.,* **24**: 86–93, 1995.
23. A. L. Oliveira and A. Sangiovanni-Vincentelli, LSAT—An algorithm for the synthesis of two level threshold gate networks, *Proc. Int. Conf. Comput. Aided Design,* 1991, pp. 130–133.
24. P. D. Wendt, E. J. Coyle, and N. C. Gallagher Jr., Stack filters, *IEEE Trans. Acoust. Speech Signal Process.,* **34** (4), 898–911, 1986.
25. B. I. Justusson, Median filtering: Statistical properties, in T. S. Huang (eds.), *Two-Dimensional Digital Signal Processing II,* New York: Springer, 1981.
26. L. Yin et al., Weighted median filters: A tutorial, *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.,* **43** (3), 157–192, 1996.

María J. Avedillo
José M. Quintana
Adoración Rueda
University of Seville

THYRISTOR. See Thyristor phase control.